Data Analitiği

" Matematiksel Modelleme ve Algoritma "

Dr. Cahit Karakuş

“ Başka bir gezegendeki yaşama hazırlanma amacıyla bilinçlenme okuludur; bu dünya”

CKK

İçerik

- Giriş
- Matematiksel Modelleme
- Simülasyon

Problemin Çözümü

İnsanların yaşamları boyunca karşılaştıkları problemlere çözüm arayışları zamanla bu çözümleri modeller üzerinde arama yaklaşımını doğurmuştur.

Karşılaşılan herhangi bir problemi çözmek ve sonuç elde edebilmek için;

Deneysel Yöntem: Çözülmesi istenen problemle ilgili neden sonuç ilişkilerini saptamak üzere araştırmacının hazırladığı deneysel yöntemdir.

Sezgisel Yöntem (Deneme - Yanılma Yöntemi): problemin çözümüne yönelik hazırlanan deneme yanılma metotlarıyla sonucun elde edilmesini sağlayan yöntemdir.

Deneyimsel Yöntem: Çözülecek probleme yönelik daha önceki çözüm çalışmalarından kazanılan deneyimler ve yetenekler ile geliştirilen çözümleme yöntemine deneyimsel yöntem adı verilir.

Analiz - Modelleme

- **Analiz:** Bir bütün içerisindeki bileşiklerin hepsinin veya bir kaçının özelliklerinin neler olduğunu ortaya koymak. Çözümlemek, Tahlil etmek.
- **Modelleme:** Bilgisayardaki gelişmeler, problemlerin matematiksel olarak çözümlerinin modellenmesi ve bu modellerin gerçek hayata yansıtılması olanağını vermiştir. Matematiksel modelleme tekniğinde doğrusal ve doğrusal olmayan modeller kullanılmaktadır.
- **Program** : Belirli bir problemi çözmek için bir bilgisayar dili kullanılarak yazılmış komutlar dizisi.
- **Algoritma:** Bir sorunun çözüm sürecinde tasarlanan yollar ve yöntemlerdir.

Model Kavramı

- Model, kısaca, “gerçek sistemlerin temsili” olarak tanımlanabilir.
- Bu tanımda, “gerçek” ifadesi hem şu anda mevcut sistemleri hem de şu anda mevcut olmayan ancak gelecekteki olası sistemler için kullanılır.
- **Mevcut bir sistem için model geliştirmenin amacı:** sistemin performansını analiz etmek, riskleri ve sapmaları önceden belirlemek iken, gelecekteki olası sistemler modellenirken amaç, kurulacak sistemin ideal yapısını tanımlamak ve hassas noktlarını saptamaktır.
- Modelden elde edilecek çıktıların karar durumlarını değiştirmedeki olası değişimleri ve sonuçları da nasıl değiştireceğini göstermesini sağladığından karar vericiye kararında yardımcı olur.

Neden model?

- Karmaşık bir sistemin basitleştirilmesi
- Manipülasyon kolaylığı (deneme yerine simülasyon)
- Sistemi anlamada kazanımlar: Yeni hipotezler oluşturmaya yardımcı olur. Yeni deneylerin tasarımına yardımcı olur.
- Mükemmel bir model yok! Bir model doğruluğu, esnekliği ve maliyeti dengelemelidir.
- Genel bir kural, artan doğruluk esnekliği azaltır ve maliyeti artırır
- Hedef, yüksek esneklik ve düşük maliyetle "yeterince" doğru bir model oluşturmak.
- Sorun asla "tamamen" çözülemeyebilir, araştırma yapacaksanız buna alışmanız gerekir

Modelleme

1) **Problemin Tanımlanması**
   Problemin tanımlanması evresi, ele alınan problemin incelenip izlenerek tanımlanmasını kapsar. Bu aşamada çalışmanın amacı/amaçları, bu amaca/amaçlara ulaşmada etkili olan sistem kısıtları ve karar seçenekleri belirlenmelidir.
2) **Model Kurma**
   Problem tanımlandıktan sonra problemi etkileyen parametre değerlerinin belirlenerek problemin matematiksel modeli kurulur. Bir başka deyişle problem matematik diline tercüme edilir.
3) **Modelin Çözümü**
   Model kurulduktan sonra optimizasyon (eniyileme) algoritmaları kullanılarak çözümlenir. Bu evrede aynı zamanda, model çözümlendikten sonra duyarlılık analizleriyle parametrelerdeki değişimlerin optimal çözüm üzerindeki değişimleri incelenir.
4) **Modelin Geçerliliği**
   Bu aşamada çözümlenen modelin gerçeği doğru bir şekilde temsil edip etmediği araştırılır. Modelden elde edilen sonuçlarla sistemin gözlenmesiyle elde edilen sonuçlar karşılaştırılır. Böylece modelin beklenen davranışları sergileyip sergilemediği incelenir.

Modelleme Aşamaları

Modelleme Aşamaları

Pratik Modelleme:

1. Modellerin Oluşturulması (Building of Models)
2. Tahminler ve Analizler Oluşturulması (Generate Predictions & Analysis)
3. Validate (Doğrulama)
4. Apply (Uygulama)

Deneysel modelleme:

1. Deneyi tasarlayın (Design the experiment)
2. Veri oluşturulur (Generate data)
3. Deneysel veriler analiz edilir (Analyse experimental data)
4. Deneysel bulguların doğrulanması (Validate experimental findings)
5. Sonuçların pratiğe uygulanması (Apply results in practice)

Modellerin Oluşturulması (Building of Models)

1. Model hedeflerinin tanımlanması
2. Uygun seviye ve anahtar model bileşenlerinin belirlenmesi
3. Varsayımların tanımlanması
4. Bir akış diyagramının oluşturulması
5. Model denklemlerinin yazılması

Model tahminleri oluşturulması ve analiz edilmesi

There are 2 ways of solving the model equations for given parameter values

1. **Analytically (using mathematical principles)**
   - İdealdir, kesin çözümler ve dolayısıyla model davranışının tam bir içgörüsünü sağlar.
   - Ancak genellikle yalnızca çok basit sistemler için mümkündür (örneğin, bir denklem veya doğrusal denklem sistemi)

2. **Numerically (using computers)**
   - Çoğu matematiksel model için geçerlidir
   - Hesaplama rutinlerinde uygulanan sayısal algoritmaların kullanılmasını gerektirir (örn.Euler yöntemi, Runge-Kutta, Monte-Carlo)
   - Yaklaşık çözümler sağlar
   - Yerleşik kod kullanılmalıdır, kendi sayısal çözücü yazılmaktan kaçınılmalıdır !!!

Model tahminleri oluşturulması ve analiz edilmesi

- Uygun model giriş ve çıkışlarının belirlenmesi
- Model girdi parametresi değerlerinin tahmin edilmesi
- İlgili model senaryolarının ve çıktılarının seçilmesi
- Anlamlı model çıktıları üretilmesi
- Modelin analiz edilmesi

Analysis techniques:

1. Asimptotik davranış
2. Başlangıç değerinin sapma davranışı
3. Duyarlılık ve Belirsizlik analizi

Modeli doğrulama

Validating the model

- Ideally (but not necessarily!) involves comparison of model predictions to experimental data
- Important to use independent data to those used for parameter estimation
  - If independent data don't exist, split your data into training and validation set
- Useful summary statistics for comparing model predictions (P_i) to observations (O_i):

$$\textit{Bias (B)} = \frac{1}{n}\sum_{i=1}^{n}(P_i - O_i)$$

$$\textit{Standard deviation (SD)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_i - O_i - B)^2}$$

$$\textit{Prediction mean square error (MSE)} = \frac{1}{n}\sum_{i=1}^{n}(P_i - O_i)^2$$

Modeli doğrulama

What if model predictions are different to the observations?

Identify potential reasons for imperfect predictions:

1. Natural variability in the real system and environment
   - Equates to experimental measurement errors
   - Obtain confidence intervals directly from the data; if model predictions fall within these limits, don't worry
2. Mis-specifications in the model
   - Wrong parameter values → extend parameter range, use fitting algorithms
   - Errors in the choice of model equations
   - Restrict the scope of the model or look for better equations and start again
3. Effects of factors ignored in the model
   - Increase model complexity and start again

Modeli doğrulama

Comparing alternative models

Independent models:

- Subjective choice: no objective model selection criterion available
- Balance between generality, flexibility, predictive ability, computing requirements

Related (e.g. nested) models:

- For models with likelihood (L), k parameters and n available independent data points, use information criteria (IC) such as
  - AIC (Akaike IC): -2log(L) + 2k; defined for nested models
  - BIC (Bayesian IC): - 2log(L) + k log(n); penalizes models with more parameters

Modeli uygulamak

Applying the model

- Mathematical models can be a valuable decision support tool
  - For risk assessment – particularly important in infectious disease context
  - To predict consequences of various (disease) control strategies

- It requires trust that the model predictions are valid
  - It is crucial to keep the purpose of the model and the end user of the model in mind at all modelling stages
  - The user should have a thorough understanding of the model assumptions, model predictions (with uncertainty estimates) and limitations

4. Modeli uygulamak

What is a good model?

Key attributes of a good model:

1. Fit for purpose
   - As simple as possible, but sufficiently complex to adequately represent the real system without obstructing understanding
   - Appropriate balance between accuracy, transparency and flexibility
2. For predictive models: Parameterizable from available data

Matematiksel Modelleme

Matematiksel Modelleme

- Bir sistemin girdi-çıktı davranışını tanımlayan matematiksel denklemlerdir.
- Bir sistemin girdi-çıktı davranışını tanımlayan iç bloklarının matematiksel ifadeleridir.
- Gerçek sistemlerin matematiksel olarak temsil edilmesidir.
- Bir sistemin girdi-çıktı davranışını tanımlayan matematiksel denklemlerdir.
- Sistemin giriş ve çıkışlarından deneyim kazanarak sistemin işlemsel bloklarını ve katsayılarını öğrenen matematiksel modellerdir.
- Matematiksel modellemede parametreleri tanımlayan değişkenler en iyileme (Optimizasyon) amacıyla güncellenir.
- Matematiksel modellemede denklemler sistemin davranışını matematiksel olarak tanımlar, değişkenleri ve parametreleri ilişkilendirir.
- Matematiksel modellemede, açıklamak veya tahmin etmek; yorumlama ve doğrulama yapılır.

Mathematical Modeling

- Matematiksel modelleme, gerçek dünya sistem davranışını açıklamak ve / veya tahmin etmekle ilgili bir uygulamalı matematik alanıdır.
- Matematiksel modelleme bilimsel araştırmada, teori ve deneyleri tamamlayıcı olarak kullanır ancak onların yerine geçmez.
- Matematiksel modelleme analiz ve tahmin etmektir.
- Pratiği anlatan teori

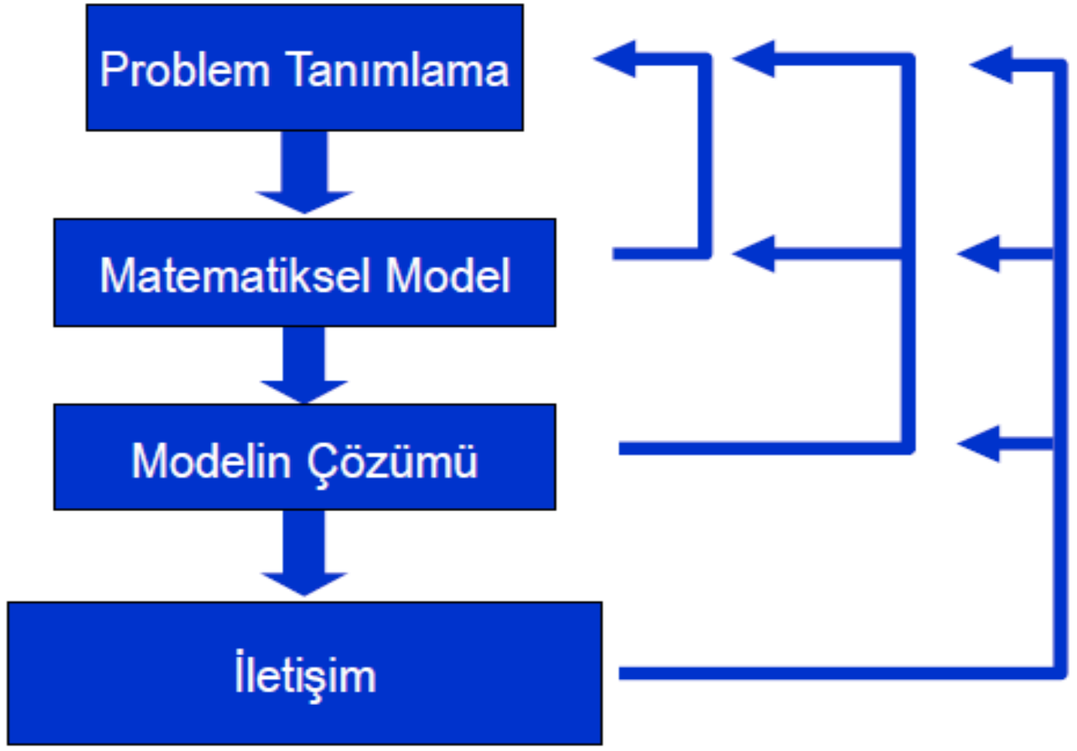
Matematiksel Modelleme Süreci ve Geri Bildirim Mekanizması

1. **Problemi belirleme ve tanımlanma,**

2. **Matematiksel Modelleme:**
   Amaçların, değişkenler ve kısıtların analizi
   Değişkenler arası ilişkilerin tanımlanması
   Formülasyon (Sembolleştirme),
   Modelin seçimi veya geliştirilmesi,

3. **Modelin çözümü ve test edilmesi,**

4. **Sonuçların sunumu ve uygulanması,**



Matematiksel Modelleme Uygulama Alanları

- Bilimsel araştırma, ürün ve süreç geliştirme ve üretimde çeşitli problemleri incelemek için güçlü ve vazgeçilmez bir araç olarak ortaya çıktı.
- Sismoloji: petrol arama, deprem tahmini (Paralel hesaplama, hesaplama süresini haftalardan saatlere düşürdü)
- Küresel okyanus / iklim modellemesi: küresel ısınma, hava durumu tahmini
- Ekonomi: yerel veya ulusal bir ekonominin büyümesi (Temsilciye dayalı modelleme), kaynakların yönetimi, vergi stratejilerinin analizi
- Çevresel: kaynakların kullanımı, popülasyon modellemesi, böcek kontrolü
- Malzeme araştırması: yeni malzemelerin tasarımı, akıllı malzemeler; sıcaklık malzemeleri tarafından yönlendirilen şekil; malzeme yaşlanma sorunları (Stok sahası yönetimi)
- İlaç tasarımı: anti-kanser ilaçlarının tasarımı vb.
- Üretim: üretim süreçlerinin optimizasyonu (En iyi süreci belirleme) otomasyon
- Tıp: Tıbbi görüntüleme, MRI'lar
- İnsan genomu: gen haritası, hastalığı anlamak ve tedavi etmek için uygulamalar

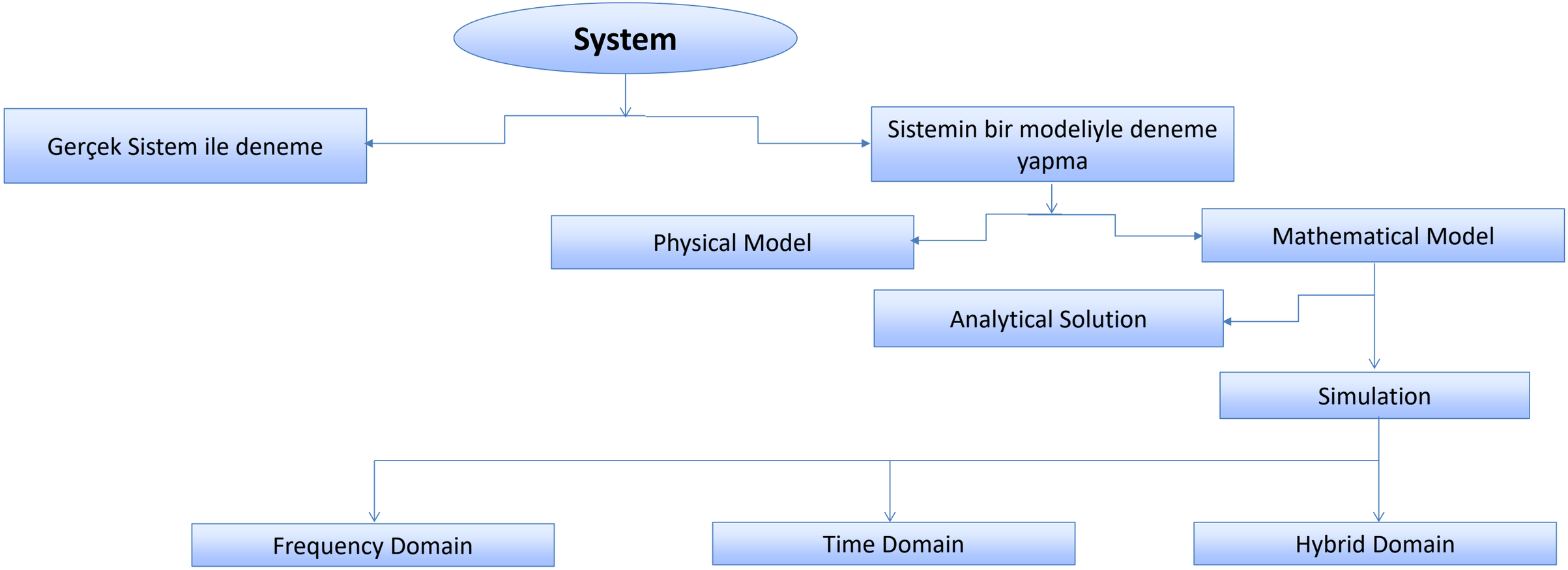
Matematiksel Modelin Temel Bileşeneleri

- **Sınırlamalar:** Değişkenlerin alabilecekleri değerleri ve limitleri gösterirler (olabilirlik / kabul edilebilirlik arralığı). Sistemin içinden veya dışından olabilirler.
- **Sabitler**: Çözümlenemez ve parçalara ayrılamaz sınırlamalar, zaman içinde değişmeyen ölçümler, katsayılardır.
- **Değişkenler:** Problemin faktörlerinin etkileşimlerini yansıtırlar, operasyonel çalışma sırasında farklı değerler alabilirler. Genelde giriş değerleri bağımsız değişkenlerdir. Bazı sistemlerde giriş değerleri çıkışların fonksiyonel ifadelerine de bağımlı olabilirler. Bunların ölçü birimlerinin ve probleme ne ölçüde dahil edileceklerinin tespiti gereklidir. Problemin özelliğine göre gereksiz faktör ve ilişkiler modele alınmamalı, ama önemli hiçbir unsur unutulmamalıdır.
- **Karar Değişkenleri:** Bir değişkenin karar değişkeni olup olmadığını belirlemek için “Bir değişkenin değeri değiştirildiğinde farklı karar çıktıları elde edilebilir mi?” sorusuna yanıt aramak yeterlidir. Eğer cevap evet ise karar değişkenidir.

Matematiksel Modelin Temel Bileşenleri

- **Karar Değişkenleri:** Amaca ulaşmak için kontrol edilen faktörler
- **Amaç Fonksiyonu:** Ulaşılmak istenen amacın karar değişkenlerinin fonksiyonu olarak matematiksel ifadesi
- **Kısıtlamalar:** Karar değişkenlerin alabileceği değerler üzerindeki sınırlama ya da gereksinimler. Kısıtlar da amaç fonksiyonu gibi karar değişkenlerinin içerildiği matematiksel fonksiyonlar olarak ifade edilir.
- **Parametreler:** Modeli etkileyen ancak karar vericinin kontrol edemediği faktörler.
- **Varsayımlar:** Model oluşturulurken doğru oldukları kabul edilen olgular.

Bir sistemin model çıktıları



What is Mathematical Model?

Bir sistemin girdi-çıktı davranışını tanımlayan bir dizi matematiksel denklemlerdir.

Matematiksel Modellemenin Kullanım Biçimleri:

- Simülasyon
- Optimizasyon (Eniyileme)
- Kalibrasyon
- Kestirim / Tahmin
- Belirtiler / Teşhis
- Tasarım / Performans Değerlendirmesi
- Kontrol Sistemi Tasarımı

Sistemlerin Matematiksel Modellerinin Sınıflandırılması

- Deneysel (Ampirik) – Mekanik (Empirical vs mechanistic models)
- Deterministik - Stokastik
- Sistemler - Moleküler model (Systems vs molecular model)
- Statik - Dinamik
- Doğrusal - Doğrusal Olmayan
- Ayrık - Sürekli
- Beyaz kutu, kara kutu ve gri kutu

Matematiksel Modelleme

Deterministik Modeller:

-Doğrusal Programlama

- Tamsayılı Programlama

-Doğrusal Olmayan Programlama

-Oyun Kuramı

- Envanter Modelleri

-Dinamik Programlama

- Çok Amaçlı Karar Verme

Stokastik Modeller:

- Belirsizlik Altında Karar Verme

- Oyun Kuramı

- Envanter Kuramında Yeni Gelişmeler

- Markov Zincirleri

- Dinamik Programlama

- Kuyruk Kuramı

- Benzetim

- Tahmin Modelleri

- Çok Amaçlı Karar Verme

Matematiksel Modellerin Türleri

Amaca göre,

1) Optimizasyon (Eniyileme) Modelleri: Matematiksel modelin bileşenlerini değiştirerek çıkış değerini maksimize ya da minimize edilerek ulaşılmak istenen bir –optimal-amaç ve bu amaca ulaşılmasını sınırlayan kısıtları içeren modellerdir.
2) Tahmin Modelleri: Bir amaca ulaşmak yerine bulunulan durumun belli şartlar altında tahmin edilmesi ya da açıklanmasıdır Örn. Geçmiş satış verileri ile gelecek satış verilerinin tahmini.

İçerdiği Belirsizliklere göre,

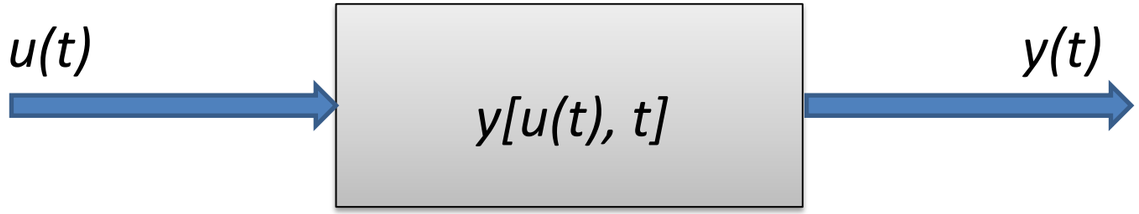
1) Deterministik Modeller: modeli oluşturan tüm parametreler kesin olarak bilinmektedir.
2) Olasılıklı (Stokastik) Modeller: Bazı parametrelerin değerleri bilinmemekte ancak bu parametreler olasılık dağılımları ile açıklanabilmektedir.

Black Box Model

- Yalnızca girdi ve çıktı bilindiğinde.İç dinamikler ya çok karmaşıktır ya da bilinmemektedir.
- Easy to Model

Grey Box Model

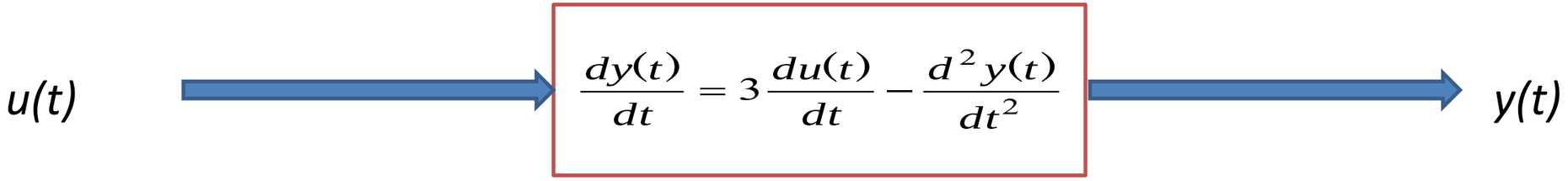
- Giriş ve çıkış ve sistemin iç dinamikleri hakkında bazı bilgiler bilindiğinde.



- Easier than white box Modelling.

White Box Model

- Sistemin girdi, çıktı ve iç dinamikleri bilindiğinde.



- Bir beyaz kutu modeli elde etmek için sistem hakkında tam bilgiye sahip olunması gerekir.

Sistemi Temsil Eden
Matematiksel Model Oluşturma

Sistem Çözüm Parametreleri

- **Analiz:** Bir bütün içerisindeki bileşiklerin hepsinin veya bir kaçının özelliklerinin neler olduğunu ortaya koymak. Çözümlemek, Tahlil etmek.
- **Sentez:** Bütünleşik yapıyı oluşturan bileşenlerdeki değişimlerin incelenmesidir.
- **Kalibrasyon:** Ölçümler arasında bir karşılaştırma. Bir cihazla yapılan veya ayarlanan bilinen büyüklük veya doğruluktan biri ve ikinci bir cihazla mümkün olduğunca benzer şekilde yapılan başka bir ölçüm. Doğruluğu bilinen veya atanan cihaza standart denir. İkinci cihaz, test edilen birim, test cihazı veya kalibre edilmekte olan cihaz için diğer birkaç addan herhangi biridir.
- **Çözünürlük:** Ölçülen miktarında algılanabilecek en küçük değişikliği ifade eder.
- **Doğruluk:** Ölçülen değerin gerçek / gerçek değere ne kadar yakın olduğunu ifade eder.
- Algoritma: Bir problemin çözüm sürecindeki yollar, yöntemler ve matetatiksel modellerdir.
- Program: Problemi çözmek için bir bilgisayar dili kullanılarak yazılmış komutlar dizisidir.
- **Update:** Güncelleme
- **Upgrade:** Yükseltme, iyileştirme
- **Modify:** Kurulumu değiştirme

Simülasyon (Benzeri Oluşturma)

- Bir sistemi temsil edebilecek bir matematiksel model oluşturma işlemidir.
- Ucuş simülasyonu, anten tasarım HFSS
- Gerçek sistemin modelinin tasarımlanması ve bu model ile sistemin işletilmesi amacına yönelik olarak , sistemin davranışını anlayabilmek veya değişik stratejileri değerlendirebilmek için deneyler yürütülmesi sürecidir.

- Geliştirilen veya yeniden düzenlenen süreçleri tamamlamada ve deneme çalışmalarını yürütmede ve süreçlerin hata zamanlarını tahmin etmek için yapılan deneysel çalışmadır .Yeni sürecin değişikliklere gösterdiği olası reaksiyonları da anlayabiliriz.

Optimizasyon (Eniyileme)

- Makine öğrenmesi ile yakın bağı vardır: birçok öğrenme probleminde, bir öğrenme seti örneğindeki işlevlerin en aza indirilmesine odaklanır.
- Her hangi bir dalda (örneğin bilgisayar yada telefonda) performansı artırmak ve daha iyi verim alabilmek için yani sistemi daha iyi bir hale getirmek için yapılan işlemlerin tümüne optimize (Eniyileme) ya da optimize etmek denir.
- Yeni girdiye aşırı duyarlılık gösteren parametreler; kaotik davranış.
- Giriş değerinde en küçük değişim, çıkışta çok büyük sapmaya neden olursa
- Bir sistemin işlevini tanımlayan çok sayıda değişkenleri olan bir denklemde, değişken değerlerinin belirlenmesidir.
- Bir değişkenle eniyileme kolay, onlarca değişken eniyileme çok zordur. Burda optimizasyon devreye girer. Optime – kabul edilebilir değerlenden koabul edilebilir bir sonuç elde edilir.

Sentez

- Bileşenlerinden bütünün davranışını incelemek.
- Bir sistemin öğelerini mantıksal bir tarzda bir araya getirme işlemi.
- Birleştirme faaliyetleri
- Bir madde ya da sistemi oluşturmaktır.
- Element veya başka maddeleri bir araya getirerek yapay olarak bileşik cisimler oluşturma, bireşim.
- Varılan sonuca gidilme, bireşim

Algoritma

Algoritma geliştirmede uzman olunması gereken alanalar

Matematiksel modellerin bilgisayar çözülmesinde ne çok sık kullanılır.

- Senaryolar
- Yazılım kodu, Yazılım dili
- Algoritma ve Matematiksel modelleme
- Donanım: CPU, Bellek, I/O, Bellek Organizasyonu, Girdiler, Çıkışlar,
- İş süreçleri analizi

What is Program

- A Set of Instructions
- Data Structures + Algorithms
- Data Structure = A Container stores Data
- Algorithm = Logic + Control

Algoritma Nedir?

- **Algoritma**, bir sorunun çözüm sürecinde tasarlanan yollar ve yöntemlerdir.
- Algoritma: Mevcut giriş bilgilerden istenilen sonuca erişebilmek amacıyla planlanan çözüm yöntemlerinin semboller ile açıklanmasıdır.
- Algoritma, verilerin, bilgisayara hangi çevre biriminden girileceğinin, problemin nasıl çözüleceğinin, hangi basamaklardan geçirilerek sonuç alınacağının, sonucun nasıl ve nereye yazılacağının semboller ile fade edilmesi biçiminde tanımlanabilir.
- Program geliştirme sürecinde işlemlerin hangi sırada ve nasıl gerçekleşeceğini belirleyen planların yapılması gerekir. Algoritma doğru bir şekilde oluşturulduktan sonra istenen programlama dili ile kodlama yapılabilir.
- Algoritma oluşturulduktan sonra problemin çözüm basamaklarında birbirleri ile ilişkili bilgi akışı daha kolay görülebilir ve yanlışlıklar düzeltilebilir. Bir giriş verisine karşılık, çıkış verisi elde edilir.
- Matematiksel modelin çözülmesinde algoritmalar çok sık kullanılır.
- **Algoritma bir programlama dili değildir.** Programlama dillerine yol gösteren bir semboller dizisidir. Algoritma, sıralı olmalıdır, belirli olmalıdır, sonlu olmalıdır.

Bir çözüm yönteminin algoritma olabilmesi için gerekli koşullar:

- İşlemler yeterli, sonuca uygun ve işleyici için geçerli ve genel olmalıdır.
- İşlem dizisinde beklenen sonuca en kısa yoldan ulaşmalı, el ile ya da mantıksal olarak ya da kağıt kalemle doğruluğu izlenebilir olmalıdır.
- İşlem tekrarlarının önüne geçebilmek için kütüphane ve alt parogramlardan yararlanılmalıdır.
- Girdileri ve çıktıları tanımlı olmaldır.

Bir algoritma oluştururken dikkat edilecek özellikler

I. Kesinlik : Algoritma içindeki adımların herkes tarafından anlaşılabilir olması, içerisinde farklı anlamlara gelebilecek bulanık ifadeler içermemesi gerekir.

I. Sıralı Olma : Yapılacak işlemlerin hangi adımda gerçekleştirileceği algoritma içerisinde net bir şekilde belirtilmelidir.

II. Sonluluk : Algoritma mutlaka sonlu sayıda adımdan oluşmalıdır. Her algoritmanın bir son noktası ve sınırlı bir zaman dilimi olması gerekir.

Algoritma geliştirmenin temel adımları

1. Problemin Tanımlanması : Algoritmanın amacı belirli bir problemi çözmektir. bu nedenle problem ne kadar anlaşılırsa algoritmanın geliştirilmesi de o kadar kolaylaşır.

2. Girdi ve Çıktıların Belirlenmesi : Problemin iyi tanımlanabilmesi için başlangıç ve bitiş noktalarının çok net bilinmesi gerekir.

3. Çözüm Yolları Bulmak : Bir problemin çözümü için birden fazla çözüm alternatifi olabilir. Bu noktada programcının en sade çözümü tercih etmesi gerekir. Çünkü karmaşık çözümlemeler programa dönüştürüldüğünde anlaşılabilirliğini kaybedebilir.

4. Çözümün Kontrolü ve Testi : Algoritma oluşturulduktan sonra mutlaka kontrol edilmelidir. Kontrol esnasında bir eksiklik ya da bir hata ile karşılaşılır ise bu sorunun düzeltilmesi gerekir. Bu eksiklikler ve hatalar giderildikten sonra mutlaka algoritma kağıt üzerinde değerler vererek test edilmelidir.

Algoritma geliştirmenin temel adımları

5. Algoritmanın Kodlanması: Geliştirilen algoritma bilgisayar üzerinde bir programlama dili ile yazılır. Böylece kağıt üzerinde geliştirilen algoritma bilgisayar ortamında çalışabilecek hale gelmiş olur.

6. Kodun Sınanması ve İyileştirilmesi: Yazılan kod algoritmada olduğu gibi test edilir. Bu test aşamasında bir hata meydana gelir ise hatanın bulunduğu kod bloğunda iyileştirilme yapılır.

7. Algoritma oluşturma ve iyileştirme aşamalarında beyin fırtınasına yönelik ekip çalışmasına gereksinim vardır. Unutulan, hatalı oluşturulan ve farklı yorumlanan ifadeler sorgulanarak düzeltilmelidir.

Algoritmanın temel öğeleri

- Tanımlayıcı : Değişken, sabit , alt yordam gibi programlama birimlerine yazılımcı tarafından verilmiş isimlere tanımlayıcı adı verilir.

- Değişken : Programın akışı içinde farklı değerleri tutmak üzere ayrılmış bellek bölümlerine değişken adı verilir.

- Sabit : Program her çalıştırıldığında ve programın içinde herhangi bir aşamada hep aynı değeri döndüren tanımlayıcılara sabit adı verilir.

- Gömülü değer : Kod içinde yazılmış olan metinsel, sayısal ya da diğer veri tiplerindeki sabit değerlere denir.

Algoritma geliştirmede önemli noktalar

- Belirsizlik
- Giriş aralığı
- Hassasiyet
- Aynı algoritma farklı şekillerde gösterilebilir
- Aynı problemi çözmek için birkaç algoritma
- Tekrarlar

Algoritma Geliştirmede Kullanılan Temel Kavramlar

Yapısal İşlem Blokları

- Programming Constructs:
  - decision structures: **if … then … [else …]**
  - while-loops: **while … do**
  - repeat-loops: **repeat … until …**
  - for-loop: **for … do**
  - array indexing: **A[i], A[i,j]**
- Methods:
  - calls: object method(args)
  - returns: **return** value

Algoritma Geliştirmede Kullanılan Temel Kavramlar

1 – Değişkenler

2 – Atama Operatörü

3 – Sayaçlar

4 – Döngüler

Algoritma Geliştirmede Kullanılan Temel Kavramlar

- **1- Değişken :** Bir program içerisinde bilgileri tutmak ve bu bilgiler üzerinde işlem yapmak için değişkenlerden yararlanılır.
- Örneğin c=a+b ifadesindeki "a","b" ve "c" değişkenlerdir.
- Değişkenler, farklı zamanlarda farklı değerler alabilen bilgi sahalarına verilen sembolik adlardır.
- Bilgisayar işlem yaparken RAM belleği(geçici bellek) kullanır. İşte program yazılırken programcının Ram belleği kullanmasını sağlayan değişkenlerdir.
- Değişkenler Ram bellekte tahsis edilmiş odacıklar olarak düşünülebilir. Yani bir değişken tanımlandığında ram bellekte bir odacık (bir bölüm) açılır ve bu bölüme değişken ismiyle ulaşılır.
- Program içinde kullanılacak olan değişkenler problemin tanımı ve girdi-çıktı belirleme aşamalarında belirlenmelidir.

Algoritma Geliştirmede Kullanılan Temel Kavramlar

- **2-Atama:** Herhangi bir değişkenin içine bir değeri veya ifadenin/işlemin sonucunu aktarma işlemidir.
- değişken = ifade
- Satırında 'değişken' yazan kısım, herhangi bir değişkenin adıdır. 'ifade' yazan kısımda ise matematiksel, mantıksal veya alfa nümerik bir ifade olabilir. Aradaki '=' sembolü, 'atama operatörü' olarak adlandırılır ve sağdaki ifadenin/işlemin sonucunu soldaki değişkene aktarır. Bu durumda değişkenin -eğer varsa- bir önceki değeri (eski değeri) silinir.
- X=3
- Y=X+5
- işleminin sonucunda Y'nin bir önceki değeri silinerek yerine 8 değeri yazılır.

Algoritma Geliştirmede Kullanılan Temel Kavramlar

- **3- Sayaç :** Programlarımızda bazı işlemlerin belirli sayıda yaptırılması veya işlenen/üretilen değerlerin sayılması gerekebilir.
- Örneğin: Klavyeden girilen bir cümlede kaç sesli harf olduğunu bulan programda, cümlenin her harfi sırayla çağrılır ve sesli harfler kümesine ait olup olmadığı araştırılır. Eğer çağrılan harf bu kümeye ait ise bunları sayacak olan değişkenin değeri bir artırılır.
- sayac = sayac + 1
- Şeklindeki işlemde sağdaki ifadede değişkenin eski(önceki) değerine '1' eklenmekte; bulunan sonuç yine kendisine, yeni değer olarak aktarılmaktadır. Bu tür değişkenlere, algoritmada "sayaç" veya "sayıcı" (counter) adı verilir.Yani "sayaç"; işlem akışı kendisine her geldiğinde, belirtilen adım değeri kadar artan/azalan değişkendir.

Algoritma Geliştirmede Kullanılan Temel Kavramlar

4- Döngü :

- Birçok programda, bazı işlemler belirli ardışık değerlerle gerçekleştirilmekte veya belirli sayıda yapılmaktadır.
- Programlardaki belirli işlem bloklarını, belirli sayıda tekrarlayan işlem akış çevrimlerine döngü denir.

Algoritma Geliştirmek

Problem çözmenin 3. adımı algoritma geliştirmektir. Geliştirilen algoritma kağıt ya da bilgisayar ortamında yazılı hale getirilmelidir. Algoritmayı yazılı hale getirebilmek için 3 yöntem kullanılır.

- Satır Algoritma Yöntemi
- Akış Diyagramları Yöntemi
- Sözde Kod Yöntemi

Satır algoritma ile Akış diyagramı yöntemleri matematik, inşaat, vb. herhangi bir konuda problem çözülürken kullanılabilir. Ancak sözde kod tamamen programlamaya yönelik bir gösterimdir. Ve kodlamaya çok yakın bir yöntemdir.

Akış Diyagramları

Akış Şeması

- Akış Şeması, algoritmanın görsel ya da şekilsel olarak ortaya koyulmasıdır. Problemin çözümü için yapılması gerekenleri başından sonuna kadar geometrik şekillerden oluşan simgelerler gösterilir.

- Algoritma geliştirildikten sonra, daha iyi anlaşılabilir olması ve programlama dillerine aktarımı daha kolay olması nedeniyle, akış şeması oluşturulur.

- Akış şeması bir problemin çözüm sürecinin semboller ile gösterilmesidir.

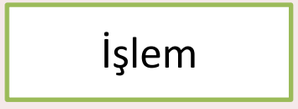
Akış Diyagramları

- Geliştirilecek olan yazılımın genel yapısının şematik gösterimine *akış diyagramı* veya *blok diyagramı* adı verilir.
- Akış diyagramları, yazılımı oluşturacak program parçalarını ve bu parçaların birbirleri ile olan ilişkilerini belirler.
- Bir bilgisayar programının oluşturulmasında akış diyagramlarının hazırlanması, algoritma oluşturma aşamasından sonra gelmektedir.
- Bilgisayar programının oluşturulması sırasında algoritma aşaması atlanarak, doğrudan akış diyagramlarının hazırlanmasına başlanabilir.
- Programlama tekniğinde önemli ölçüde yol almış kişiler bu aşamayı da atlayarak direkt olarak programın yazımına geçebilirler.
- Akış diyagramlarının algoritmadan farkı, adımların simgeler şeklinde kutular içinde yazılmış olması ve adımlar arasındaki ilişkilerin (iş akışı) oklar ile gösterilmesidir.

59

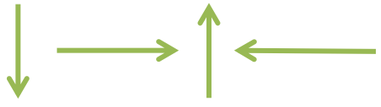
Akış Diyagramları

Akış Diyagramları, algoritmanın belirli grafikler kullanılarak ifade edilme şeklidir.

Şekil	Anlamı
	Algoritmanın Başlangıcını ve bitişini göstermekte kullanılır.
	Kullanıcıdan bilgi alınacağı zaman kullanılır.
	Aritmetiksel, Mantıksal vb. işlemleri ifade etmek için kullanılır.
	Kullanıcıya bilgi gösterileceği zaman kullanılır.

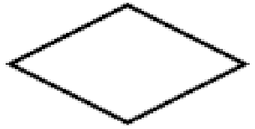
Akış Diyagramları

Akış Diyagramları, algoritmanın belirli grafikler kullanılarak ifade edilme şeklidir.

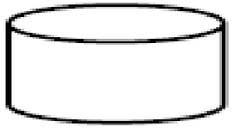
Şekil	Anlamı
	Algoritma içinde belirli bir koşula bağlı olarak akışın dallanmasını sağlamak için, karar yapılarında kullanılır.
	Tekrarlı işlemleri ifade etmek için kullanılır.
	Aynı sayfaya sığmayacak algoritmaların devamıyla bağını göstermek için kullanılır.
	Adımlar arasındaki bağlantıyı ve akış yönünü göstermek için kullanılır.

Akış Çizelgesinde Kullanılan Semboller

Başlama / Bitiş



Karar Verme



Dosya İşlemi

İşlemler

Döngü

Giriş (Klavyeden ..)

Çıkış (Ekrana...)

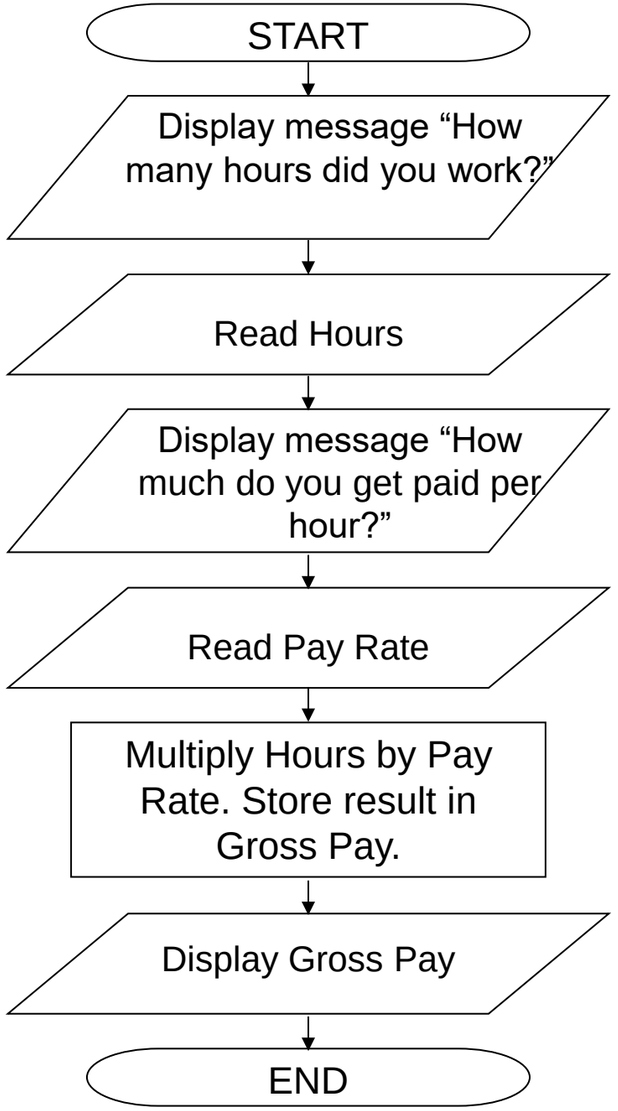
Çıkış (Kağıt...)

Akış Şeması Yapısı

- Sıra
- Karar
- Tekrarlama
- Durum

What is a Flowchart?

- A flowchart is a diagram that depicts(“yosor”) the “flow” of a program.
- The figure shown here is a flowchart for the pay-calculating program in Chapter 1.

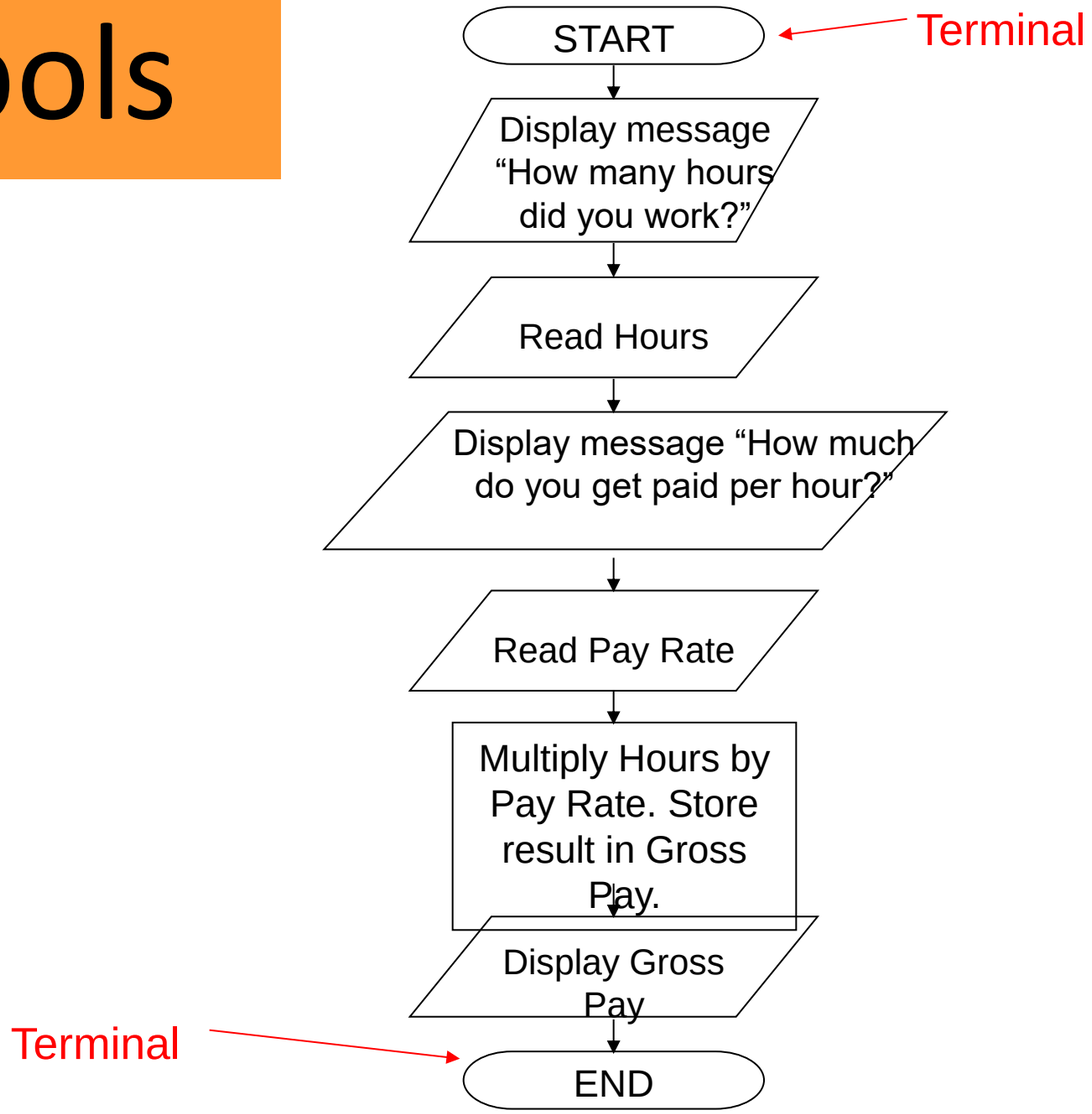


Basic Flowchart Symbols

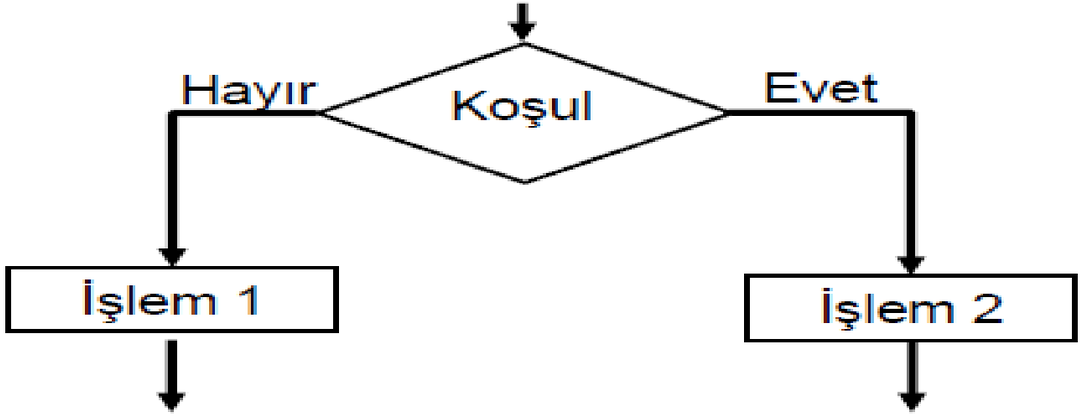
- Terminals
  - represented by rounded rectangles
  - indicate a starting or ending point

START

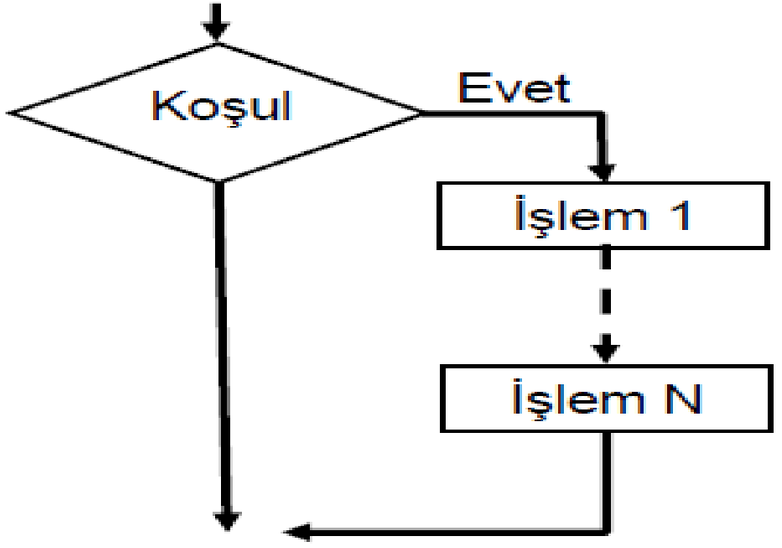
END



Karar Verme – Koşul ileri sürme



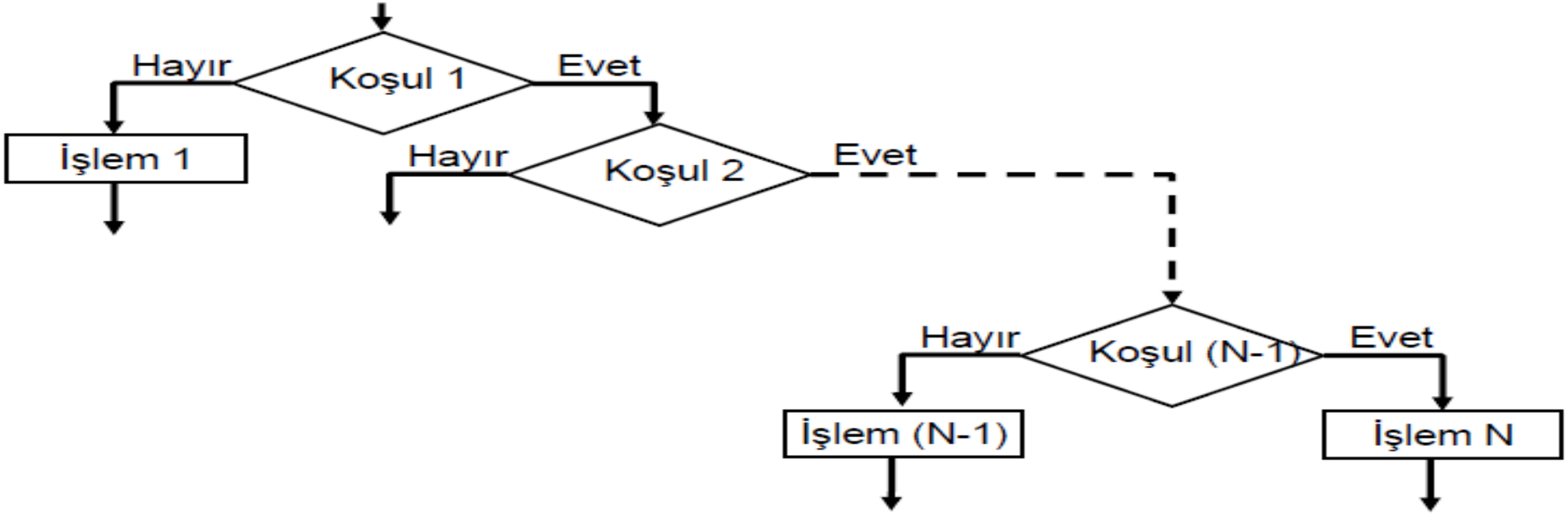
a) Koşulun durumuna bağlı olarak 2 farklı işlem vardır.



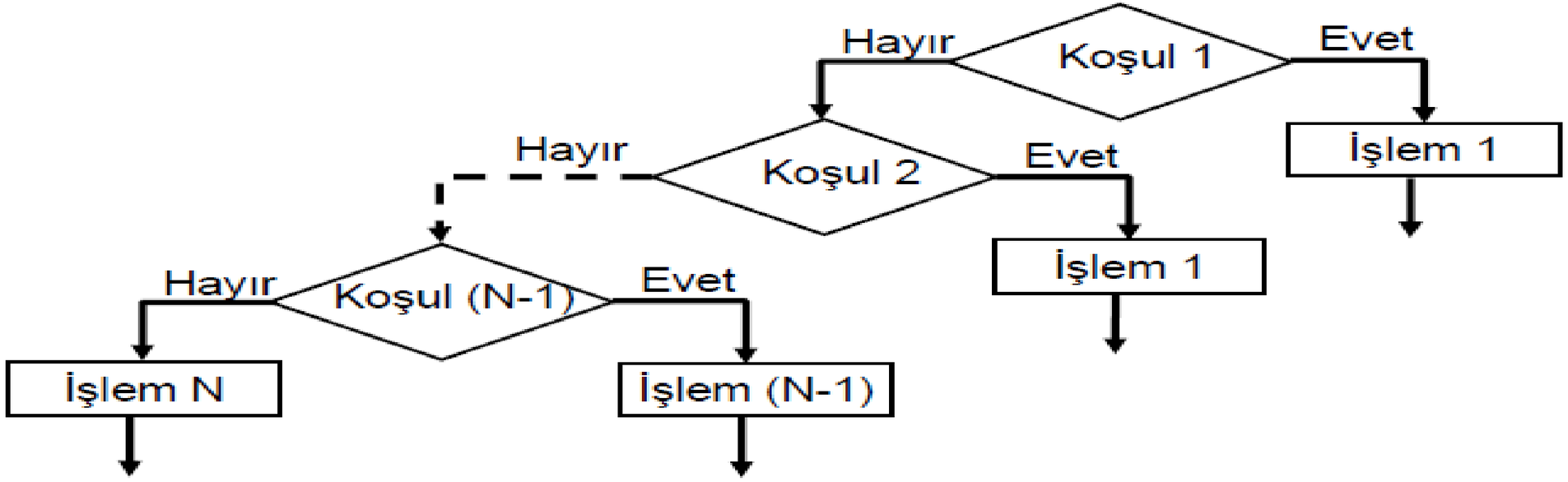
b) Olumsuz koşulda yapılacak işlem yoktur; olumlu olması durumunda ise N adet işlem yapılacaktır.

Karar Verme

Bu yapıyı art arda birden çok kez kullanıp aşağıdaki gibi bir kaşılaştırma dizisi oluşturulabilir.



Karar Verme



Olumsuz taraftan içiçe karşılaştırma simgesi örneği

Döngü Yapısı

- Bu yapı kullanılırken, döngü sayacı, koşul bilgisi ve sayacın artım bilgisi verilmelidir. Döngü sayacı kullanılmıyorsa sadece döngüye devam edebilmek için gerekli olan koşul bilgisi verilmelidir.

Genel olarak çoğu programlama dilinin döngü yapıları:

- While
- Do-while
- For

gibi yapılar üzerine kurulmuştur. Farklı dillerde bu yapılara farklı alternatifler olsa da döngülerin çalışma mantığı genel olarak benzerdir.

Döngü Oluşturma Kuralları:

1- Döngü değişkeninin başlangıç değeri belirlenir.

2- Döngü değişkeninin bitiş değeri belirlenir.

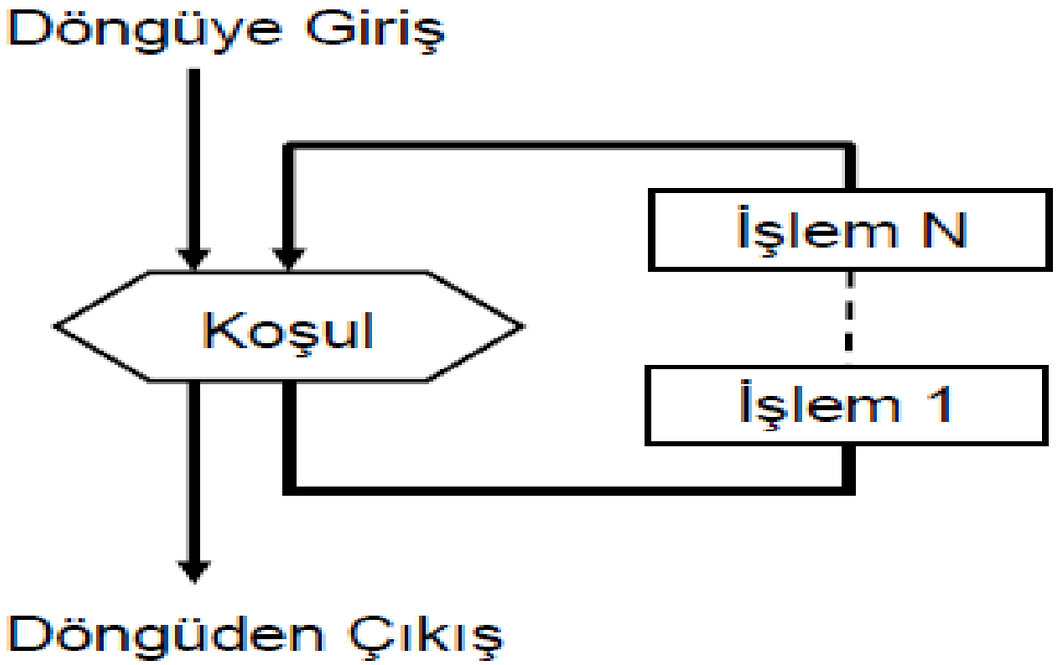
3- Döngü değişkeninin bitiş değerine ulaşıp ulaşmadığı test edilir.

4- İstenen işlem gerçekleştirilir.

5- Döngü değişkeni, döngü içinde adım miktarı kadar artırılır ya da azaltılır.

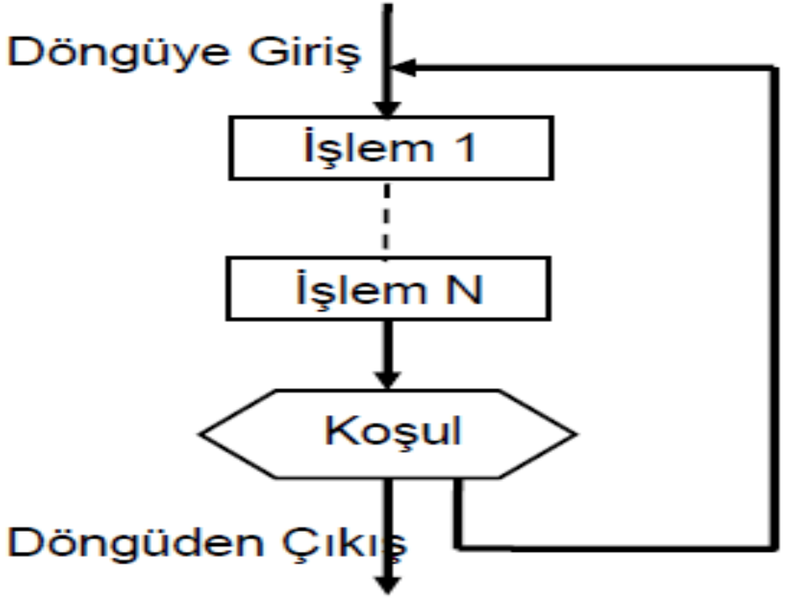
While Deyimi

- Koşul daha çevrim içerisine girmeden sınanır. Koşul olumsuz olduğunda çerime hiç girilmez ve döngü içerisinde yapılması gerekenler atlanır.



Do-While Deyimi

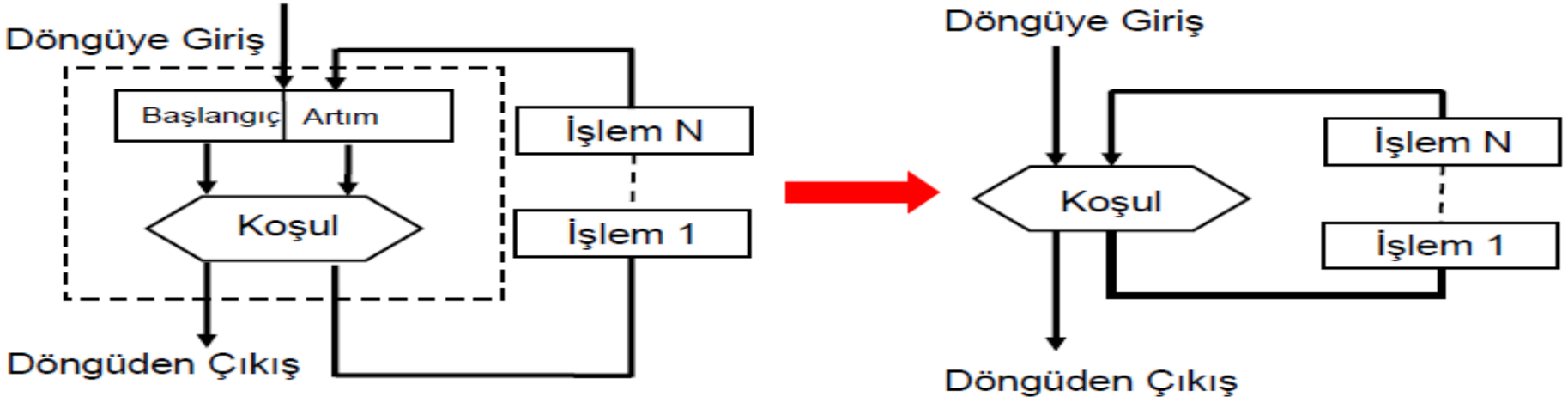
- Bu döngüdeyiminde, çevrim en az bir defa olmak üzere gerçekleşir. Çünkü koşul sınaması döngü sonunda yapılmaktadır. Eğer koşul sonucu olumsuz ise bir sonraki çevrime geçilmeden döngüden çıkılır. Çevrimin devam edebilmesi için her döngü sonunda yapılan koşul testinin olumlu sonuçlanması gerekir.



For Deyimi

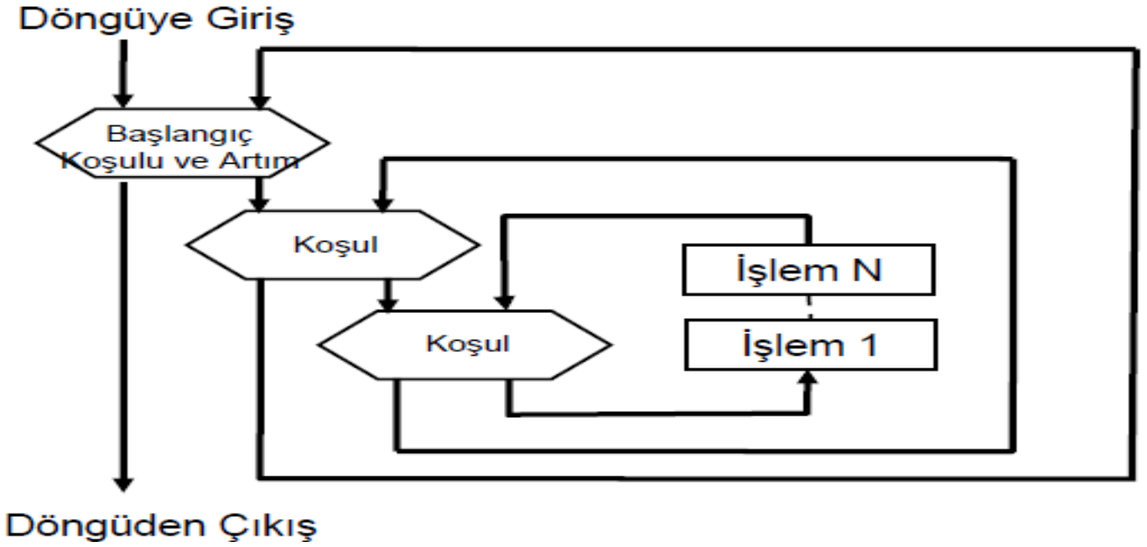
I=1,20,3 J=30,4,-2 K=1,80

- Diğer deyimlerden farklı olarak, döngü sayacı doğrudan koşul parametreleri düzeyinde verilir.
- Döngü girmeden önce sayaç değişkenine başlangıç değeri atanmakta ve daha sonra koşula bakılmaktadır. Döngü içerisinde belirtilen işlemler yapıldıktan sonra sayaç değişkeni arttırılmaktadır.



İç İçe Döngüler

- İçiçe döngü kurulurken en önemli unsur, içteki döngü sonlanmadan bir dıştaki döngüye geçilmemesidir. Diğer bir deyişle döngüler birbirlerini kesmemelidir.
- Aşağıdaki gösterimde en içteki döngü bir dıştaki döngünün her adımında N kez tekrarlanır.



Matrix Operations

- Matrix addition/subtraction
  - Matrices must be of same size.

- Matrix multiplication

m x n q x p m x p

$$\begin{bmatrix} a_{11} & a_{12} & . & a_{1n} \\ a_{21} & a_{22} & . & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & . & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & . & b_{1p} \\ b_{21} & b_{22} & . & b_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ b_{q1} & b_{q2} & . & b_{qp} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & . & c_{1p} \\ c_{21} & c_{22} & . & c_{2p} \\ \cdots & \cdots & c_{ij} & \cdots \\ c_{m1} & c_{m2} & . & c_{mp} \end{bmatrix}$$

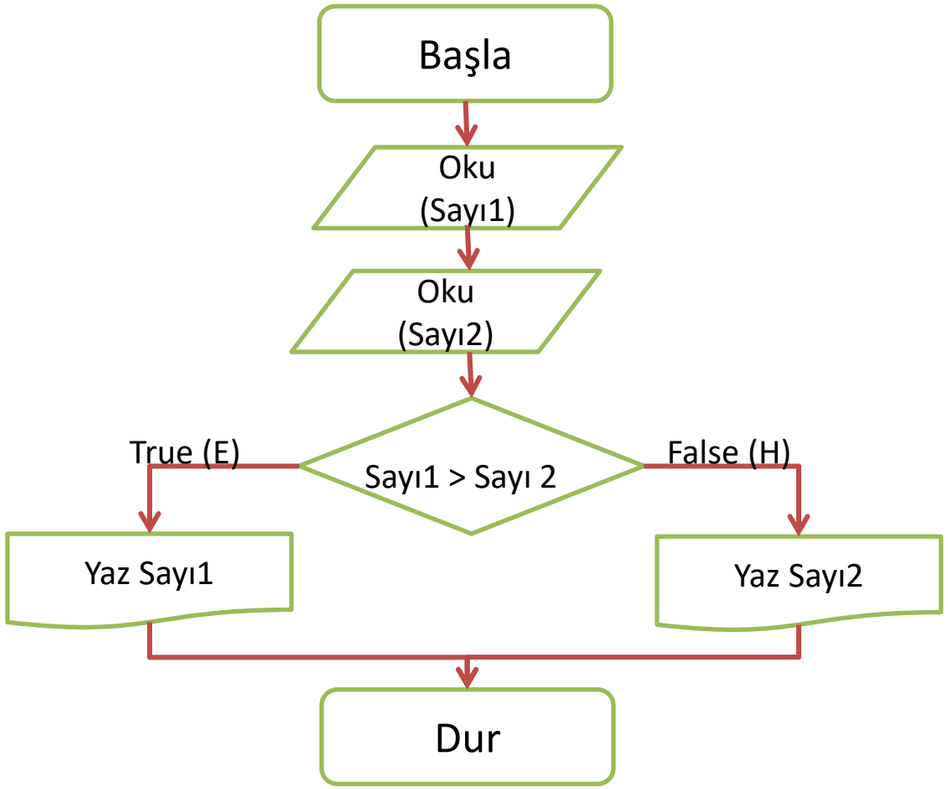
Condition: n = q

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

$$AB \neq BA$$

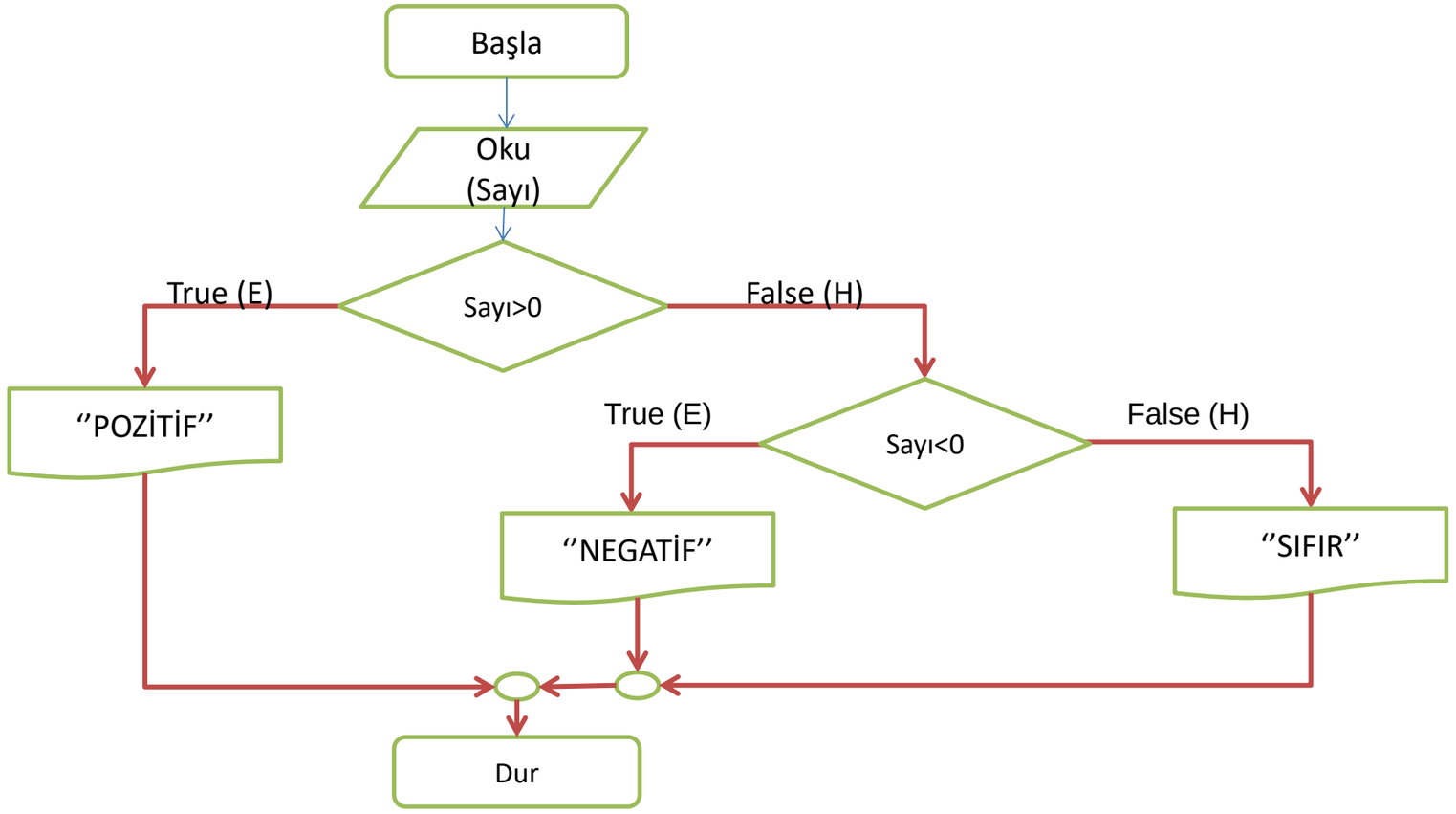
Akış Diyagramları

Klavyeden girilen iki adet sayıdan büyük olanını ekrana yazan algoritmayı oluşturunuz.



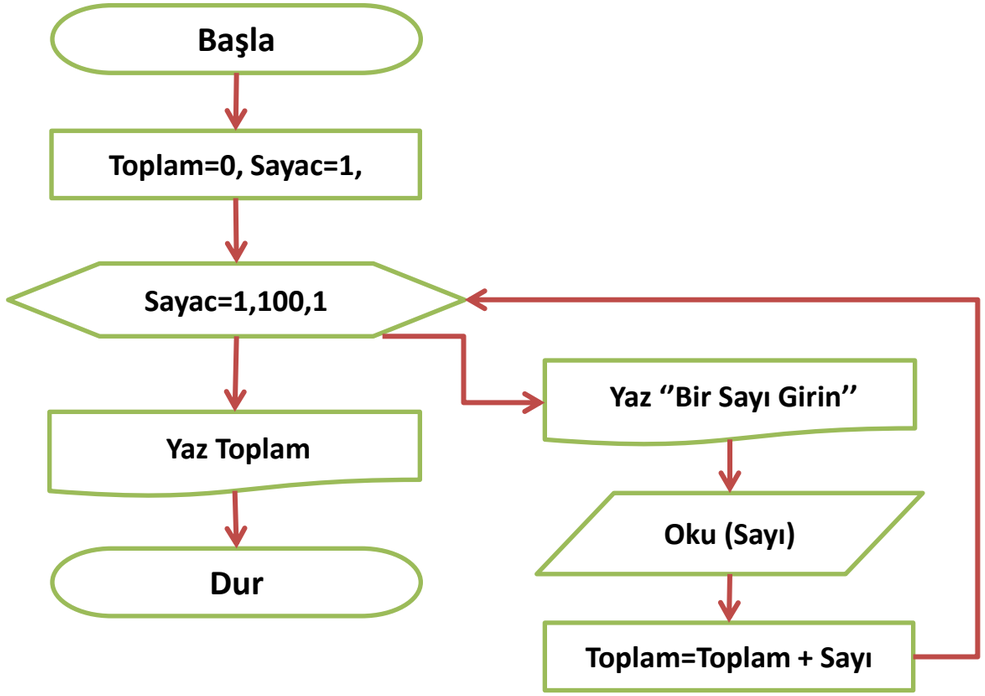
Akış Diyagramları

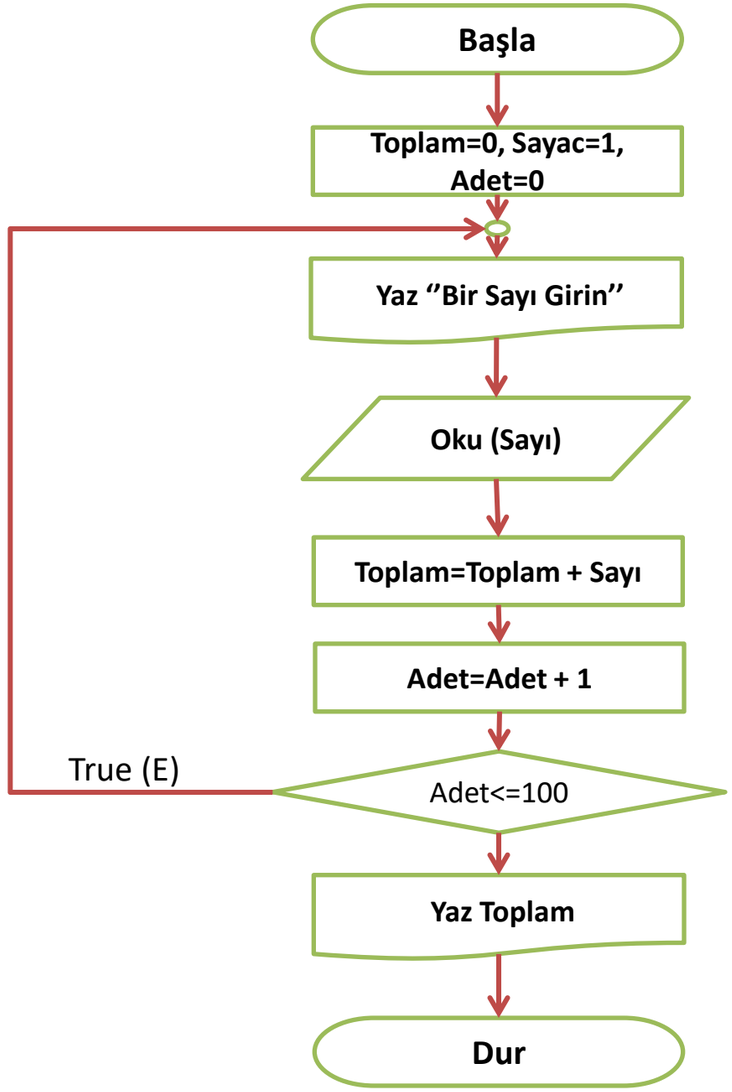
Klavyeden girilen sayının pozitif ya da negatif, ya da sıfır olduğunu ekrana yazan algoritmayı oluşturunuz.



Akış Diyagramları

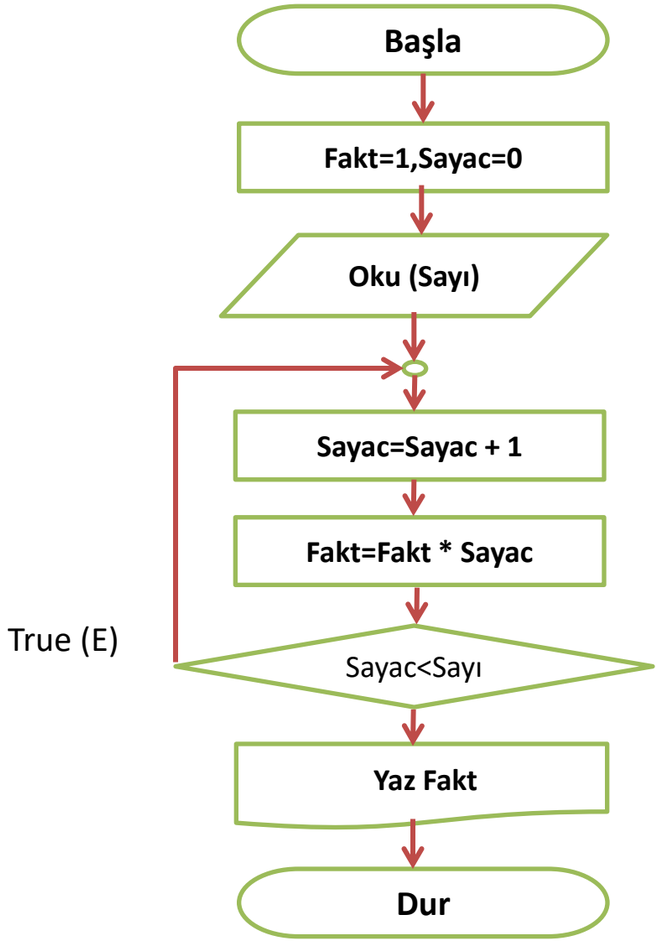
Örnek: Kullanıcıdan 100 adet sayı alarak, bu sayıların toplamını ekrana yazan algoritmayı akış diyagramları ile oluşturunuz.





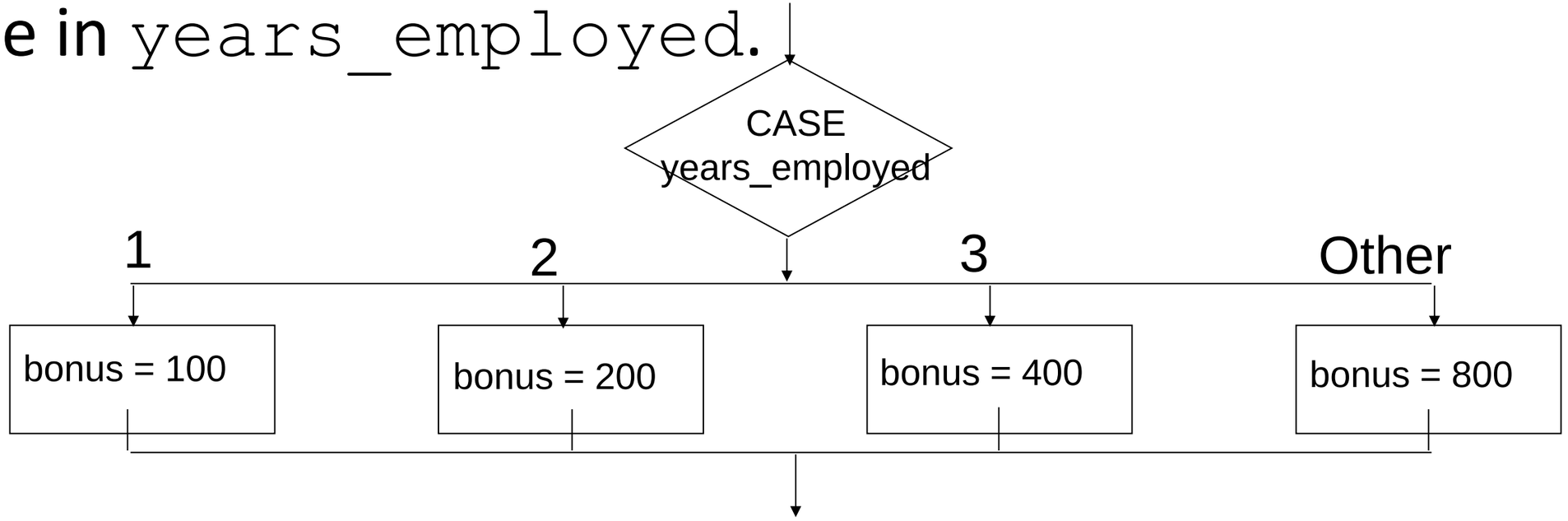
Akış Diyagramları

Klavyeden girilen sayının faktöriyelini hesaplayan algoritmayı oluşturunuz.

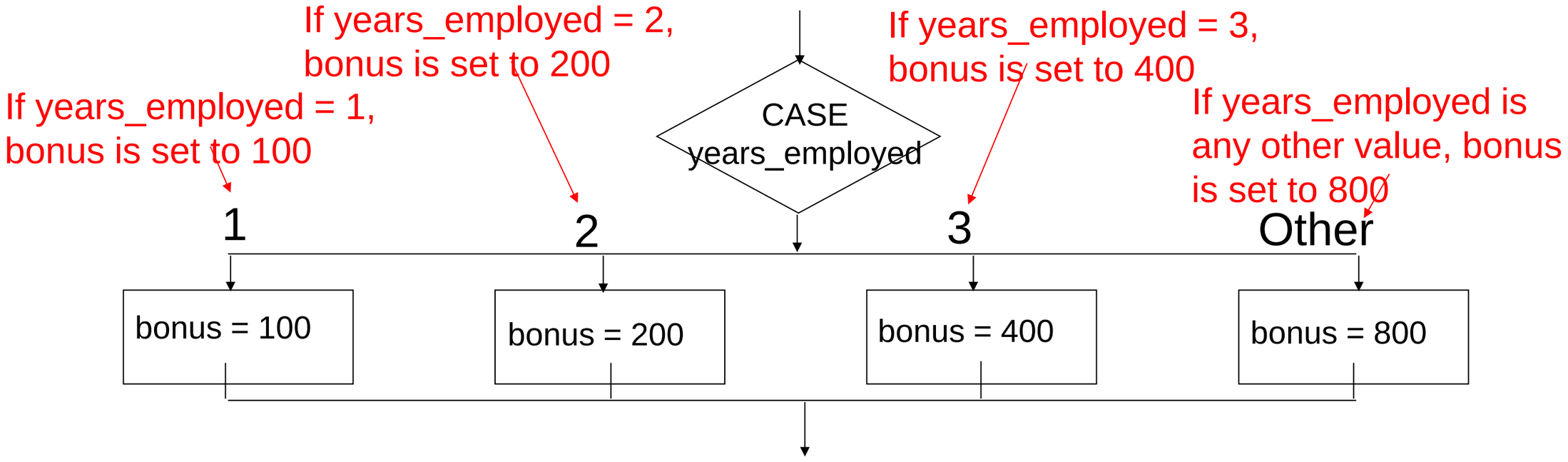


Case Structure

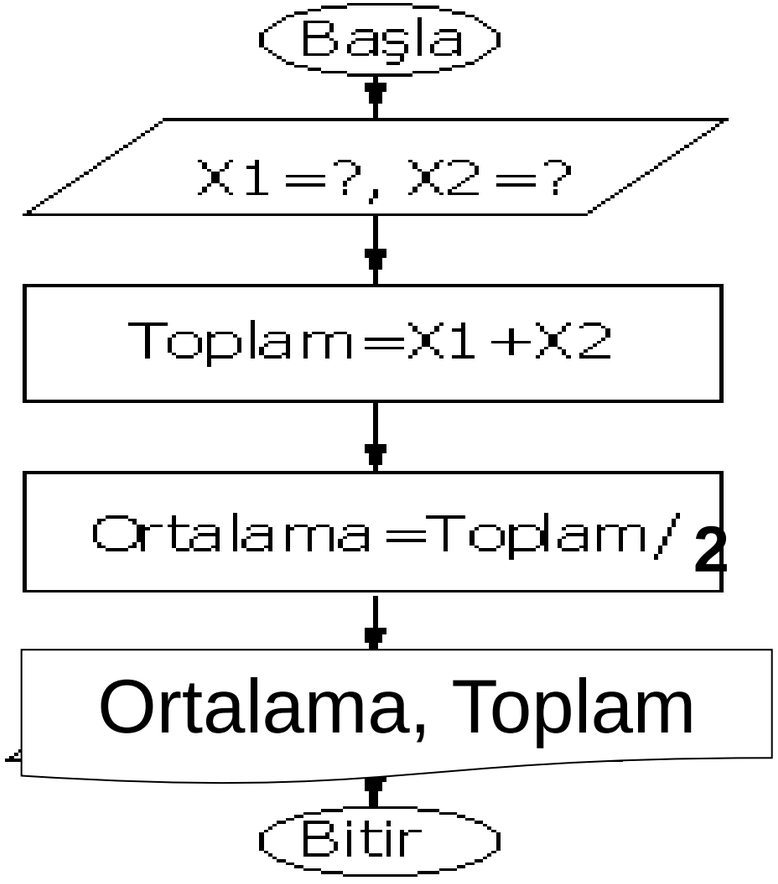
- One of several possible actions is taken, depending on the contents of a variable.
- The structure below indicates actions to perform depending on the value in `years_employed`.



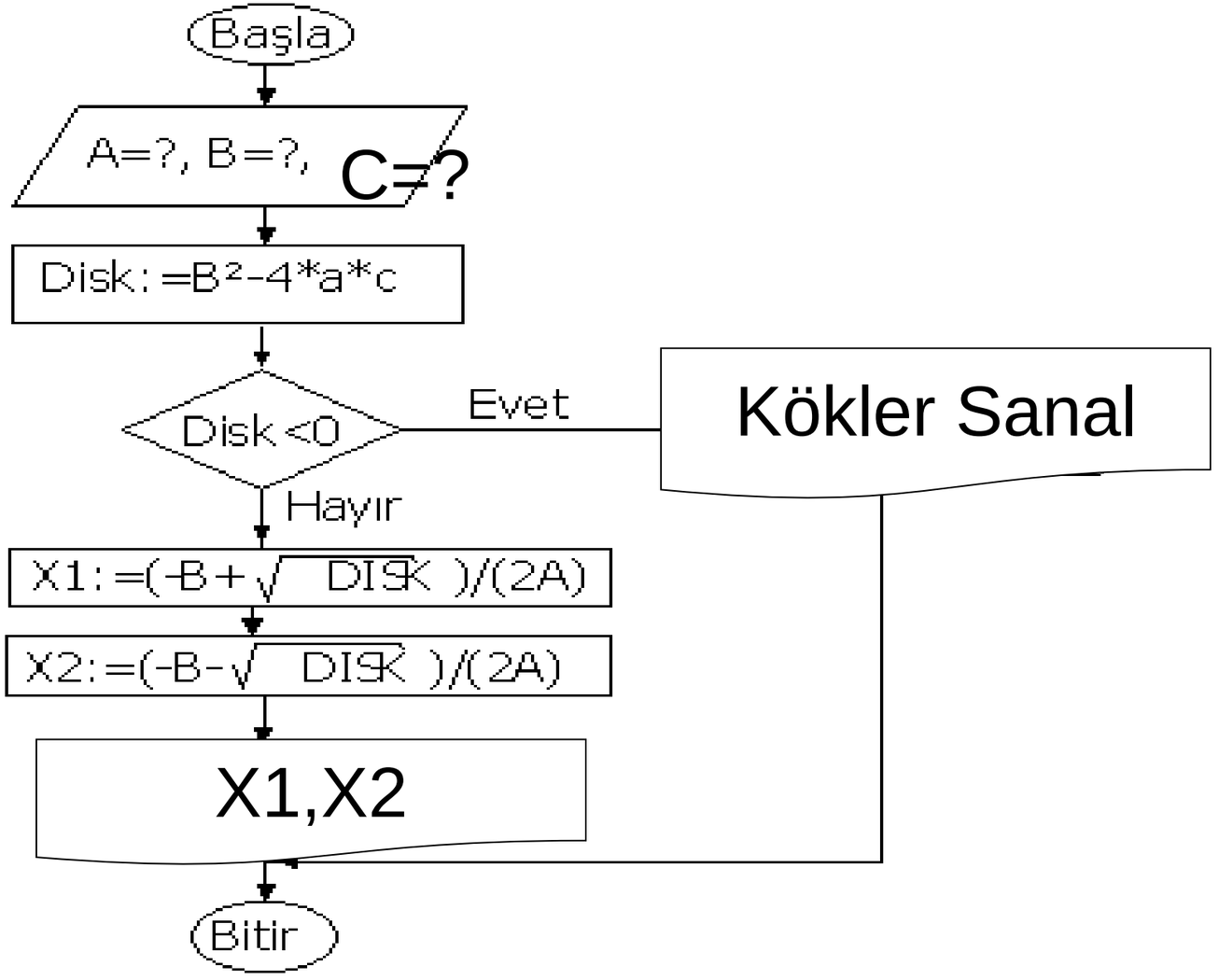
Case Structure



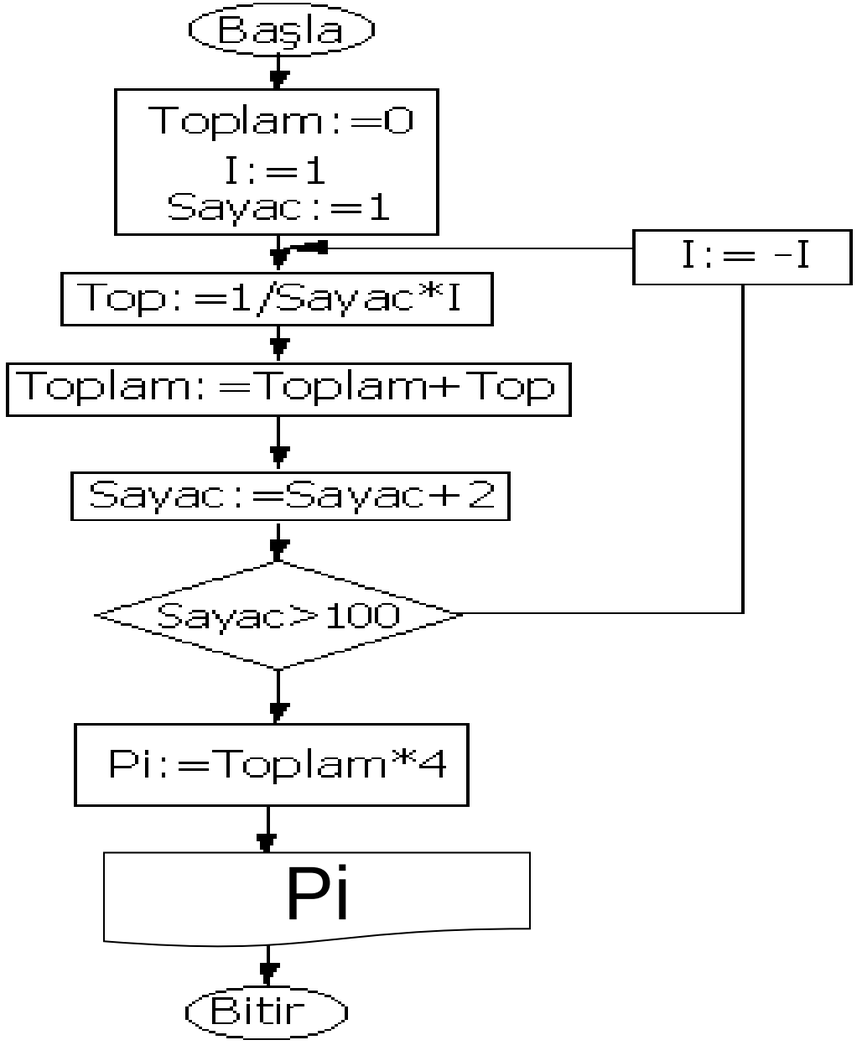
- İki sayının toplamını ve ortalamasını yapan bilgisayar programının akış diyagramını çiziniz.

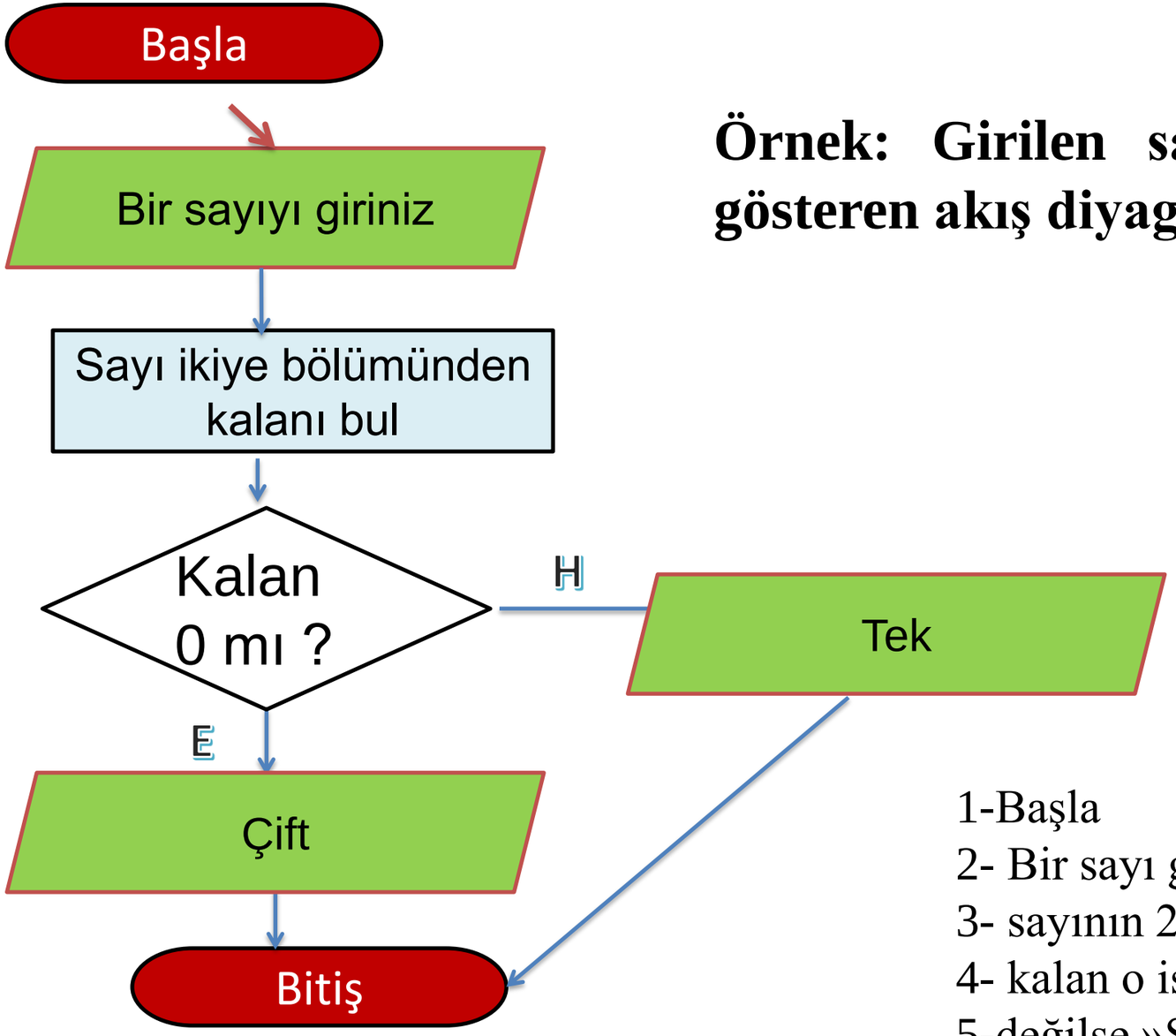


- $A^2+Bx+C=0$ şeklinde verilen 2. derece denklemin köklerini bulan programın akış diyagramını



- Örnek 12: p sayısının formülü $((1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots).4)$ serinin paydasındaki ifade 100 oluncaya kadar pi sayısını hesaplayan bilgisayar programının akış diyagramını çizınız.

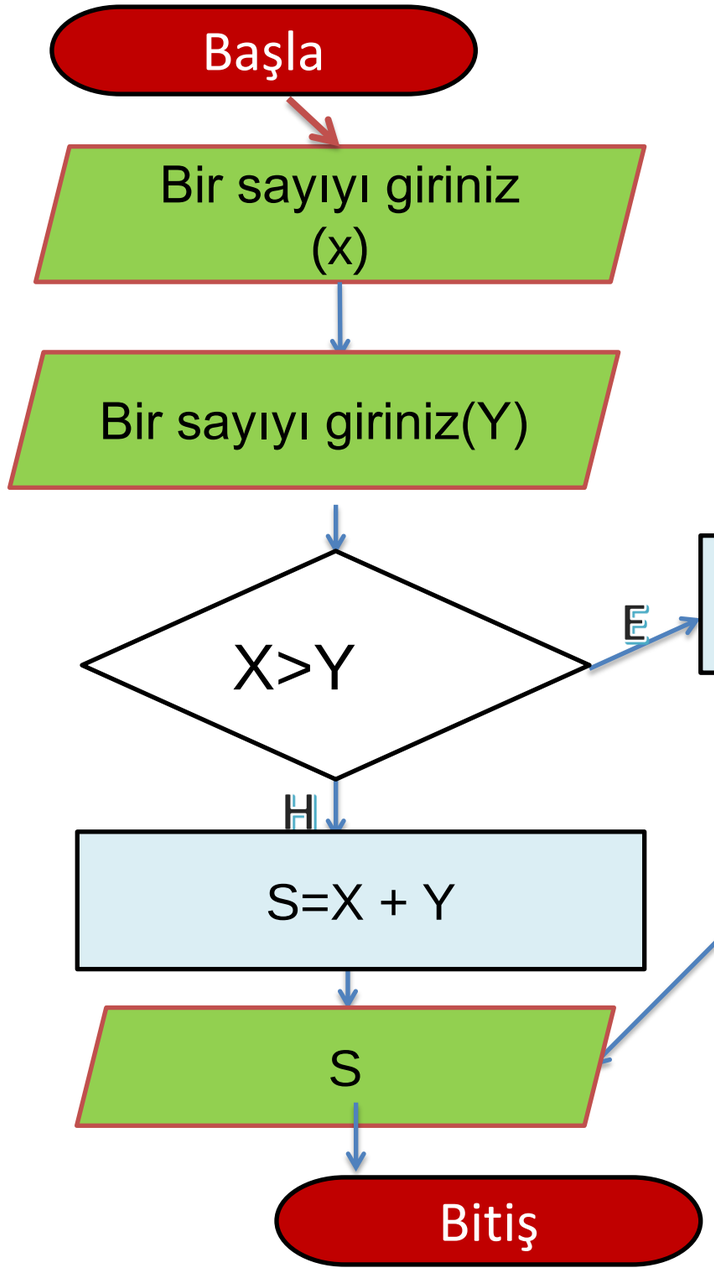




Örnek: Girilen sayının tek mi, çift mi olduğunu gösteren akış diyagramı.

Algoritması

1-Başla
2- Bir sayı giriniz
3- sayının 2 ye bölümünden kalanı bul
4- kalan o ise «sayı çifttir» yaz
5-değilse »Sayı çift «yaz
6- Bitir



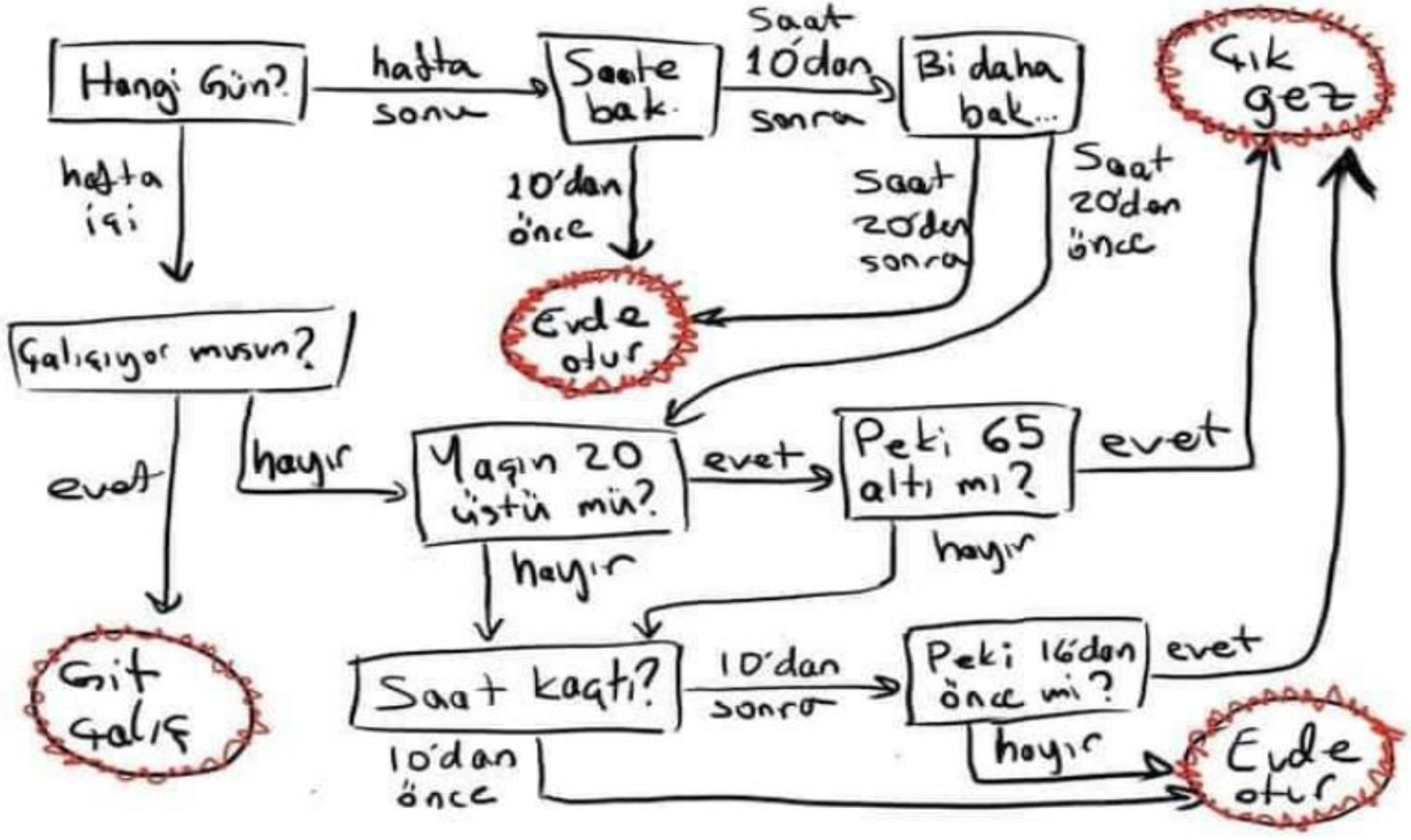
Girilen iki sayıdan birincisi ikincisinden büyükse
İkisini çarpan küçük veya eşitse toplayıp sonucunu
Gösteren programın algoritması

Değişkenler:
Birinci sayı = X
İkinci sayı = Y
Sonuç = S

Algoritma :

1.Adım : Başla
2.Adım : Birinci sayıyı oku (X)
3.Adım : İkinci sayıyı oku(Y)
4.Adım : X > Y ise S=X*Y ve 6. adıma git
5.Adım : X <= Y ise S= X +Y
6.Adım : S yi göster
7.Adım : Bitir.

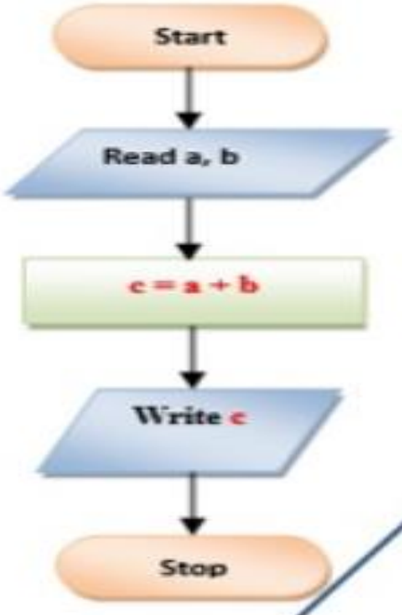
Korona evde kal algoritması



Algorithm

1. Start
2. Read a, b
3. c = a + b
4. Print or display c
5. Stop

Flowchart



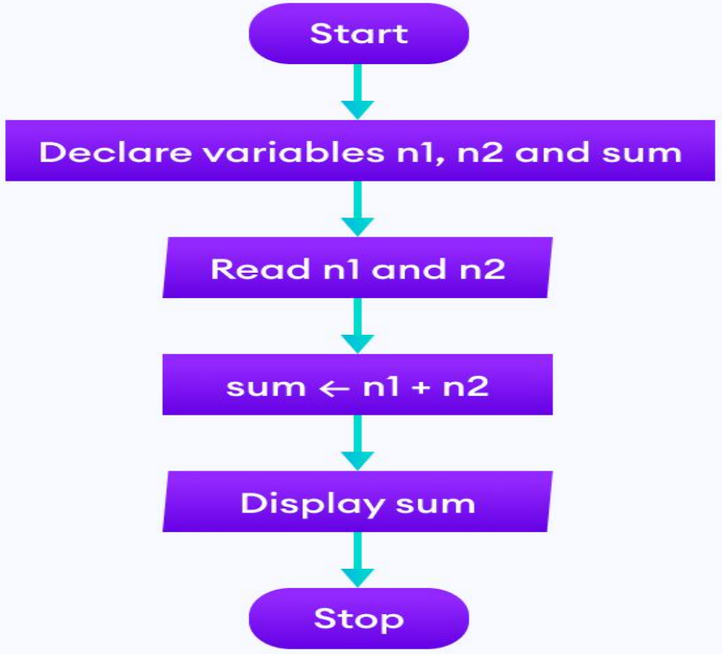
Program

1. Open **MATLAB** software
2. **File > New > Blank m-file**
3. Type below program

```
a = input(' Enter value of a: ');
b = input(' Enter value of b: ');

c = a+b;
disp(c);
```

Refer to video lecture: Introduction to MATLAB for explanations

Flowchart to add two numbers

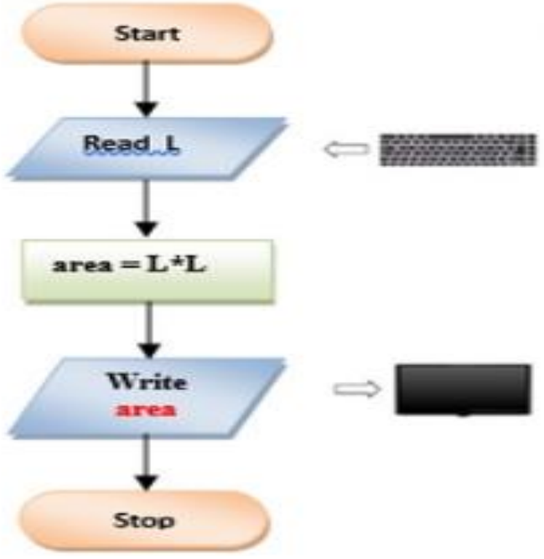


Finding Area of the square

Algorithm

1. Start
2. Read length, L
3. area = L*L
4. Print or display area
5. Stop

Flowchart



Program

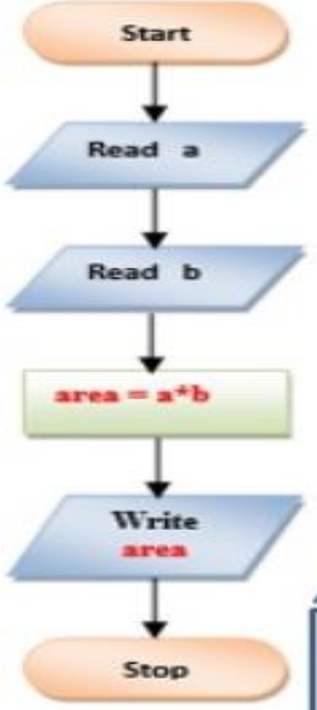
```
%Program to find area of a square

L = input(' Enter length of square L: ');

area = L * L;
disp(' Area of square is: ');
disp(area)
```

Finding Area of the rectangle

Algorithm

1. Start
2. Read side length, a
3. Read side length b
4. area = a*b
5. Print or display area
6. Stop

Flowchart



Program

```
%Program to find area of a rectangle

a = input(' Enter side length a: ');
b = input(' Enter side length b: ');

area = a * b;
disp('Area of rectangle is: ');
disp(area);
```

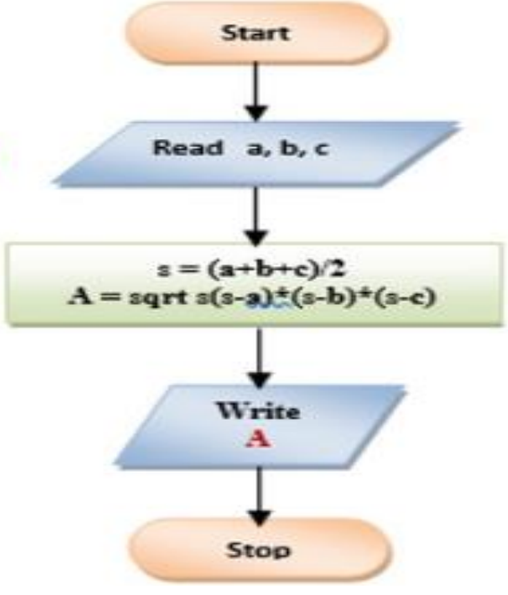
Refer to video lecture: Introduction to MATLAB for explanations

Area of a triangle where three sides are given

Algorithm

1. Start
2. Read a, b, c
3. s = (a+b+c)/2
4. A=sqrt (s *(s-a)*(s-b)*(s-c))
5. Print or display A
6. Stop

Flowchart



Program

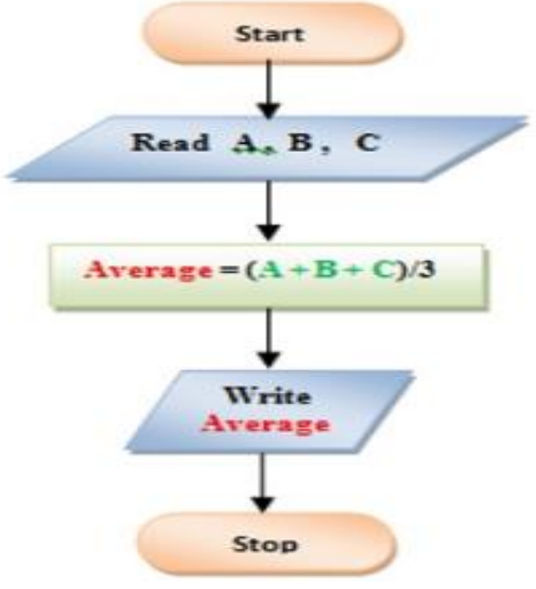
```
%Area of a triangle with 3 sides

A = input(' Enter value of a: ');
A = input(' Enter value of b: ');
A = input(' Enter value of c: ');

S = (a+b+c)/2;

A = sqrt(s*(s-a)*(s-b)*(s-c));

disp(' Area of triangle is: ');
disp(A);
```

Calculating the average for 3 numbers

Algorithm

1. Start
2. Read 3 numbers A, B, C
3. Calculate the average by the equation:
   Average = (A + B + C)/3
4. Print Average
5. Stop

Flowchart



Program

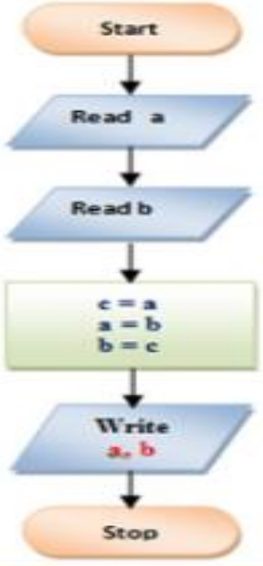
```
%Average of given 3 numbers

A = input(' Enter value of A: ');
B = input(' Enter value of B: ');
C = input(' Enter value of C: ');

Average = (A+B+C)/3;

disp(' Average of A, B, C is: ');
disp(Average);
```

Interchange the value of two numbers

Algorithm

1. Start
2. Read two values into two variables a, b
3. Declare third variable, c
   - c = a
   - a = b
   - b = c
4. Print or display a, b
5. Stop

Flowchart



Program

```
%Interchange values of two variables

a = input(' Enter value of a: ');
b = input(' Enter value of b: ');

c = a;
a = b;
b = c;

disp(' Values of a and b after swapping: ');
disp(' a = ');
disp(a);

disp(' b = ');
disp(b);
```

Greatest of two numbers

Algorithm

1. Start
2. Read A, B
3. If A > B then
   Print A is large
   else
   Print B is large
4. Stop

Flowchart

Program

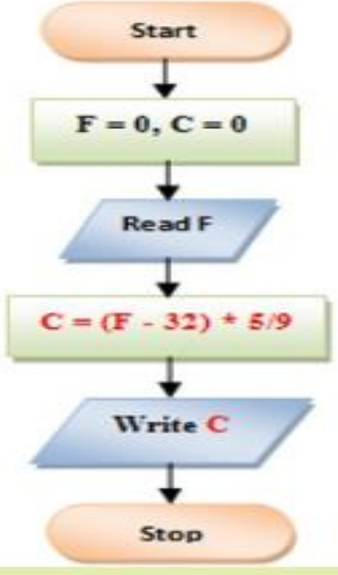
```
%Program to find greatest of two numbers

A = input(' Enter value of A: ');
B = input(' Enter value of B: ');

if (A>B)
        disp(' A is Larger: ');
else
        disp(' B is Larger');
end
```

Convert temperature from Fahrenheit to Celsius

Algorithm

1. Start
2. Initialize F = 0, C = 0
3. Read F
4. C = (F-32) * 5/9
5. Write C
6. Stop

Flowchart



Program

```
%Convert Fahrenheit to Celsius

F = input(' Enter Temp. in Fahrenheit: ');

C = (F-32) * 5/9;

disp(' Temp. in Celsius is: ');
disp(C);
```

Draw a flowchart for computing factorial N, where N! = 1 * 2 * 3 * ..... N

Algorithm

1. Start
2. Read N
3. Initialize F =1, i = 1
4. F = F * i;
5. Increment i by 1
6. Repeat step 4 & 5 until i = N
7. Print F
8. Stop

Flowchart

Program

```
%Compute factorial of given number N

N = input(' Enter value of N: ');
F = 1;

for i = 1 : N
F = F * i;
end

disp(' Factorial is: ');
disp(F);
```

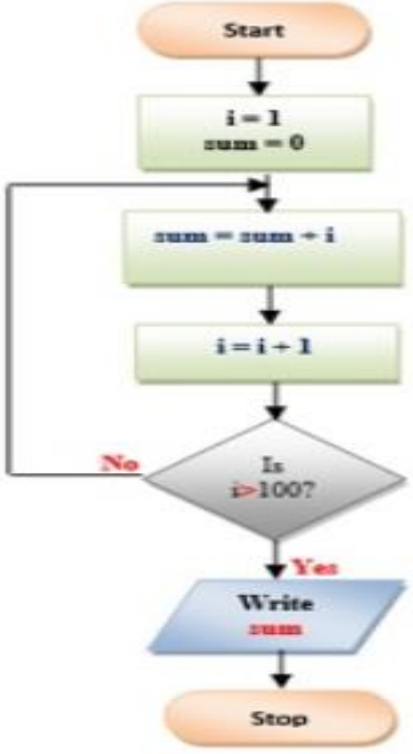
Refer to **video lecture**:
Introduction to MATLAB
for explanations

Calculating sum of integers from 1 to 100

Algorithm

1. Start
2. Initialize count i = 1, sum = 0
3. sum = sum + i
4. Increment i by 1
5. Repeat steps 3 & 4 until i > 100
6. Print sum
7. Stop

Flowchart



Program

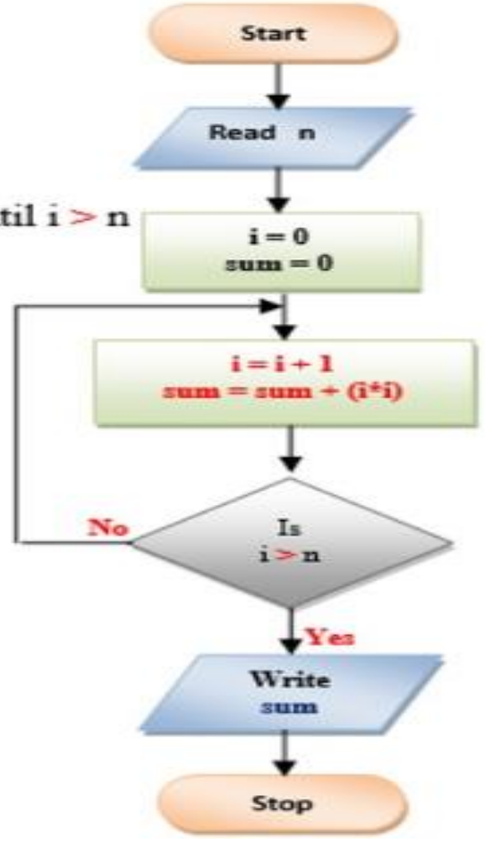
```
% Sum of integers from 1 to 100

sum = 0;
for count = 1 : 100
sum = sum + count;
end

disp(' Sum of integers from 1 to 100 is: ');
disp(sum);
```

Refer to **video lecture**:
Introduction to MATLAB
for explanations

Algorithm

1. Start
2. Read n
3. i = 0, sum = 0
4. i = i + 1
5. sum = sum + (i*i)
6. Repeat steps 4 and 5 until i > n
7. Print sum
8. Stop

Flowchart



Program

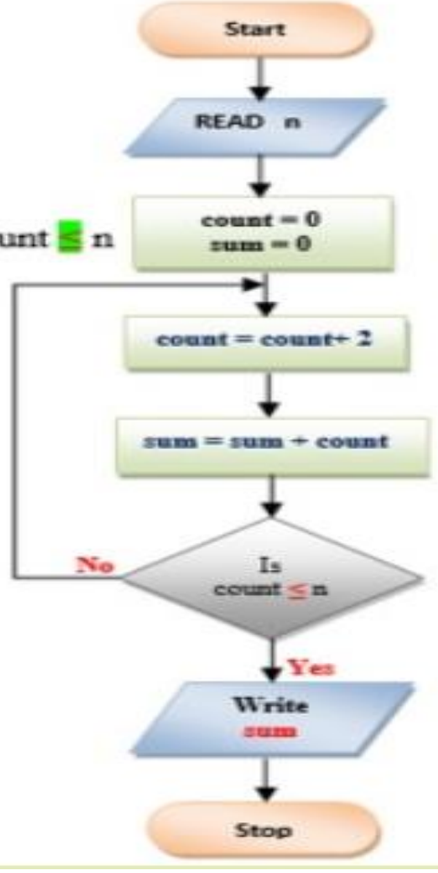
```
% Sum of squares of n natural numbers

n = input(' Enter value of n: ');

sum = 0;
for i = 1 : n
sum = sum + i*i;
end

disp(' Sum of squares of integers up to n: ');
disp(sum);
```

To find the sum of all even numbers up to 'n'

Algorithm

1. Start
2. Read n
3. count=0
4. sum=0
5. count = count + 2
6. sum = sum + count
7. Repeat steps 5 & 6 until count ≤ n
8. Print sum
9. Stop

Flowchart

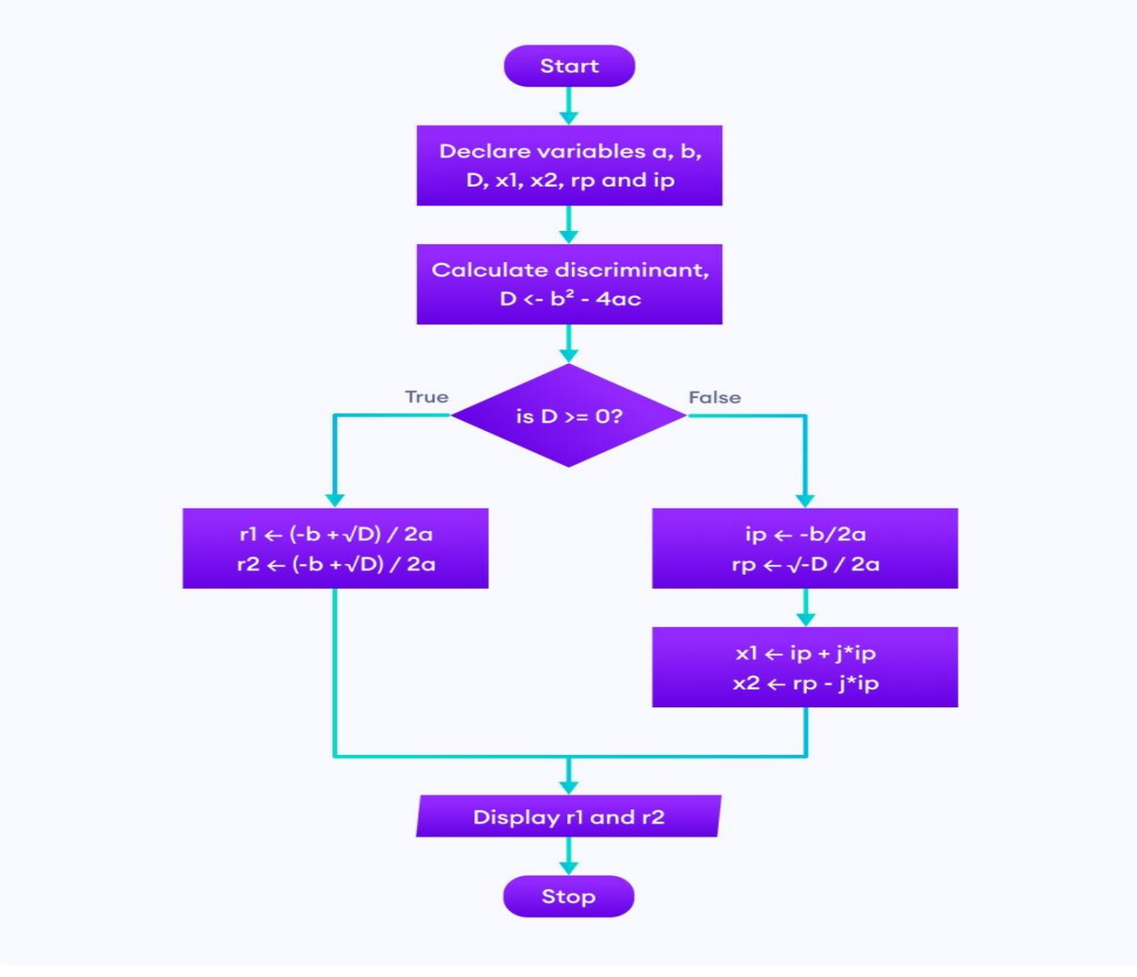


Program

```
%Sum of even numbers up to n

n = input(' Enter value of n: ');

sum = 0;
for i = 0 : 2: n
sum = sum + i;
end;

disp(' Sum of even numbers up to n: ');
disp(sum);
```

Refer to **video lecture**:
Introduction to MATLAB
for explanations

Find all the roots of a quadratic equation ax2+bx+c=0



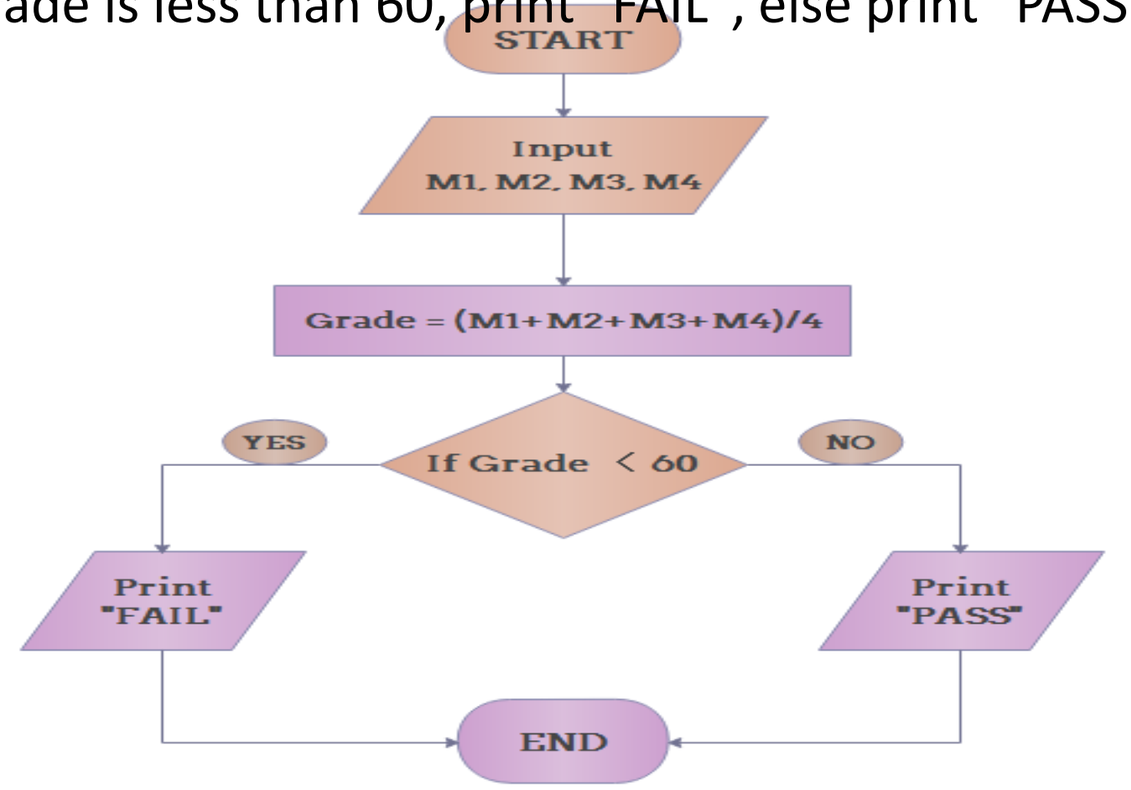
Determine Whether A Student Passed the Exam or Not:

Algorithm:

Step 1: Input grades of 4 courses M1, M2, M3 and M4,

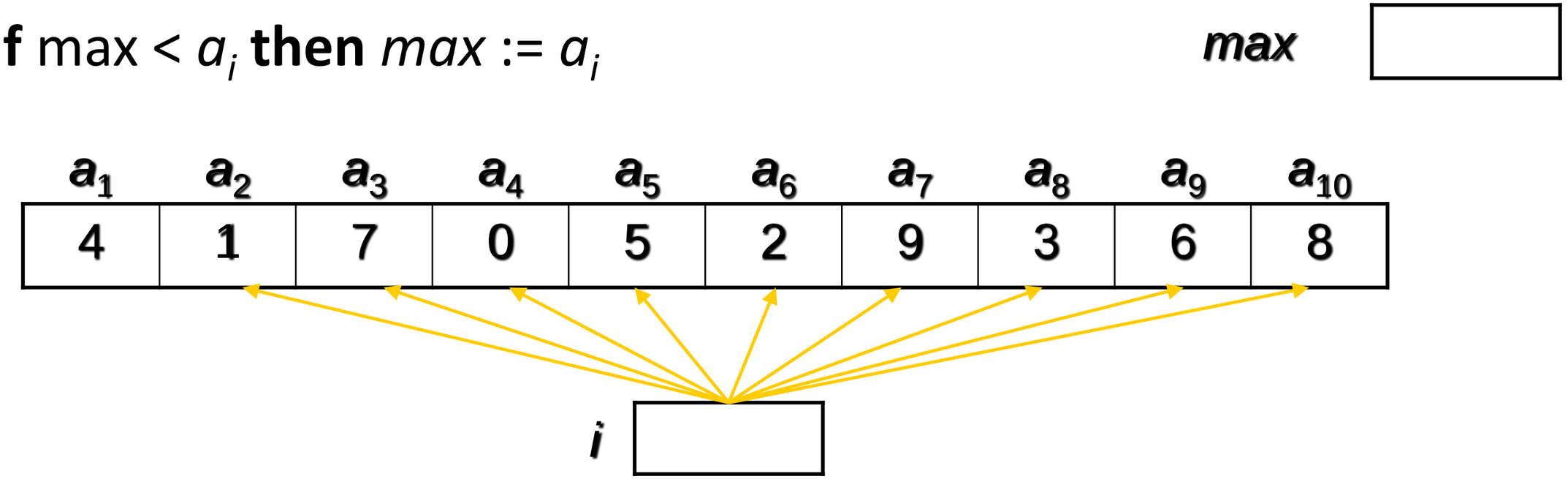
Step 2: Calculate the average grade with formula "Grade=(M1+M2+M3+M4)/4"

Step 3: If the average grade is less than 60, print "FAIL", else print "PASS".



Algorithm 1: Maximum element

procedure max (a_1, a_2, …, a_n: integers)
max := a_1
for *i* := 2 **to** n
  if max < a_i **then** *max* := a_i



Sözde Kodlamma

Sözde Kodlar

Bilgisayarda bir programlama dili olarak çalışmayan, ancak programlama dillerine yakın algoritma ifadelerine sözde kodlar(pseudo-code) denir. Bu yöntem farklı kullanım şekillerine sahiptir. Fakat genel kullanım dili İngilizce ve programlama dili olarak pascal diline çok benzerlik gösterir.

Kaba Kod veya Sözde Kod (Pseudo Code), bir algoritmanın yarı programlama dili kuralı, yarı konuşma diline dönük olarak ortaya koyulması ya da tanımlanmasıdır. Bu şekilde gösterim algoritmayı genel hatlarıyla yansıtır.

Sözde kodlar yapısal olarak 4 temel ögeye sahiptir.

1. Okuma / Yazma İşlemleri: Okuma işlemleri için GET, READ , Yazma işlemleri için WRITE,DISPLAY gibi komutlar kullanılır.
2. İşlemler :Sözde kod içinde gerçekleştirilen toplama, çıkarma, çarpma, bölme vb. aritmetiksel işlemler, bir değişkene değer atanması gibi işlemlerdir.
3. Karar Yapıları : Bir koşula bağlı olarak bir işin yapılıp yapılamayacağına karar verme işlemleridir.
4. Tekrarlı Yapılar : Program içinde bir koşula bağlı olarak ya da belirli bir sayıda tekrar edecek işlemler için kullanılırlar.

Pseudo code: basic notation

1. We will use **:=** for the assignment operator.
2. Method signatures will be written as follows:
   Algorithm name ({parameter list})
3. Programming constructs will be described as follows:
   - decision structures: **if ... then ... else ...**
   - while loops: **while ... do {body of the loop}**
   - repeat loops: **repeat {body of the loop} until ...**
   - for loops: **for ... do {body of the loop}**
   - array indexing: **A[i]**
4. Method calls: **Method name ({argument list})**
5. Return from methods: **return value**

‘Döngü’ Kavramına Örnek:

Örnek: Aşağıdaki algoritmada 1-10 arası tek sayıların toplamı hesaplanmaktadır. (Aşağıdaki algoritmayi çift sayıların toplamına hangi değişikliği yaparak dönüştürebilirsiniz?)

A1: Başla

A2: toplam=0

A3: sayac=1

A4: Eğer sayac>=10 ise adım 8 e git

A5: toplam=toplam+sayac

A6: sayac=sayac+2

A7: Adım 4 e git

A8: toplam ı ekrana yaz

A9: Bitir

Döngü

Hangi amaçlarla kaç değişkene ihtiyaç var?

Örnek : Dışarıdan girilen iki sayının toplamını bulan programın algoritması aşağıdaki gibi kurulur:

Algoritma:

A1 : Başla

A2 : A değerini gir

A3 : B değerini gir

A4 : C= A+B

A5 : C' yi ekrana yaz

A6 : Bitir

Hangi amaçlarla kaç değişkene ihtiyaç var?

Örnek : **Dışarıdan girilen 3 sayıdan en büyüğünü bulan algoritmayı kurunuz.**

Algoritma:

A1 : Başla
A2 : A, B ve C sayılarını dışarıdan gir
A3 : enBuyuk=A
A4 : Eğer enBuyuk<B ise enBuyuk=B yap
A5 : Eğer enBuyuk<C ise enBuyuk=C yap
A6 : enBuyuk değerini ekrana yaz
A7 : Bitir

Hangi amaçlarla kaç değişkene ihtiyaç var?

Örnek: 0'dan 100'e kadar olan sayma sayılarının kümülatif toplamını ekrana yazdıran algoritmayı geliştiriniz.

1. A1 : Başla
2. A2 : toplam=0;sayac=1 baslangıç değerlerini ata
3. A3 : Eğer sayac=100 ise 6. adıma git
4. A4 : toplam=toplam+sayac
5. A5 : sayac=sayac+1 yap ve 3. Adıma geri dön
6. A6 : toplam'ı ekrana yaz
7. A7 : Bitir

Hangi amaçlarla kaç değişkene ihtiyaç var?

Örnek: Verilen bir sayının faktöriyelini hesaplayan programın algoritmasını yazınız. (Faktöriyeli hesaplanacak sayı negatif girilmişse yeniden giriş istenmelidir.)

Değişkenler:

Sayının faktöriyeli : faktor , Faktöriyel Değişkeni :sayac
Faktöriyeli hesaplanacak sayı :Y

Algoritma:

A1: Başla

A2 : faktor =1; sayac =1

A3 : Y'yi gir

A4 : Eğer Y<=0 ise 3. adima git

A5 : Eğer sayac>Y ise adim 8 e git

A6: faktor=faktor*sayac

A7: sayac =sayac+1 yap ve adım 5 e git

A8 : faktor degerini ekrana yaz

A9 : Bitir

Anlamlı Değişken İsimleri Seçmek Çok Önemli !!!!!!!!!!!

Hangi amaçlarla kaç değişkene ihtiyaç var?

Satır Algoritmaları

Satır Algoritmaları

Satır Algoritmalar, problem çözümünü günlük yazı konuşma diliyle ifade ederek sıra numarasıyla yazılarak oluşturulur. Konuşma diline çok yakın olmasından dolayı bir algoritmayı ifade etmenin en basit yoludur.

Örnek : Kullanıcıdan iki sayıyı alıp, bu iki sayının toplamını ekrana yazdıran algoritmayı tasarlayın.

Çözüm :

1. Başla
2. Oku (Sayı1,Sayı2)
3. Sonuç=Sayı1 + Sayı2
4. Sonucu Ekrana Yaz
5. Dur

Not: Burada Sayı1 ve Sayı2 değişkenleri girdi, Sonuç değişkeni çıktı olarak kullanılmıştır. Yani programın iki girdisi ve bir çıktısı vardır.

Satır Algoritmaları

Örnek: Kullanıcıdan bir kenarı alınan karenin çevresini ve alanını hesaplayarak ekrana yazdıran algoritmayı tasarlayın.

1. Başla
2. Oku (Kenar)
3. Çevre=kenar * 4
4. Alan=kenar * kenar
5. Çevreyi ekrana yaz
6. Alanı ekrana yaz
7. Dur

Not : Burada kenar bilgisi girdi, çevre ve alan değerleri ise değişken olarak kullanılmıştır. Burada kenar, çevre, alan değişken olarak tanımlanmıştır.

Satır Algoritmaları

Örnek: Klavyeden yol ve aracın hız bilgisi alınarak ne kadar sürede yolun tamamlanacağını hesaplayan algoritmayı oluşturunuz.

Çözüm;

1. Başla
2. Oku (Yol)
3. Oku (Hız)
4. Süre=Yol / Hız (Y=V * t den)
5. Süreyi Ekrana Yaz
6. Dur

Satır Algoritmaları

Örnek: Klavyeden girilen iki adet sayıdan büyük olanını ekrana yazan algoritmayı oluşturunuz.

Çözüm;

1. Başla
2. Oku (Sayı1)
3. Oku (Sayı2)
4. Eğer

4.1. (Sayı1>Sayı2) Enbüyük=Sayı1

4.2. Değilse Enbüyük=Sayı2

5. Enbüyüğü ekrana yaz
6. Dur

Satır Algoritmaları

Örnek: Klavyeden girilen sayının tek ya da çift olup olmadığını ekrana yazdıran algoritmayı oluşturunuz.

Çözüm;

1. Başla
2. Oku (sayı)
3. Eğer

3. 1. ((Sayı % 2)==0) ise ekrana yaz ‘’çift”

3. 2. Değilse ekrana yaz ‘’tek”

4. Dur

Satır Algoritmaları

Örnek: Kullanıcıdan bir sayı alıp 1 den başlayarak kullanıcıdan aldığı sayıya kadar bir artırarak ekrana yazdıran algoritmayı oluşturunuz.

Çözüm;

1. Başla
2. Sayac=0, Toplam=0
3. Oku (sayı)
4. Eğer (Sayac >= Sayı), Adım 8 e git
5. Toplam = Toplam + Sayac
6. Sayac = Sayac + 1
7. Adım 4 e git
8. Yaz Toplam
9. Dur

Satır Algoritmaları

Örnek: 1 den 100 e kadar olan sayılardan 8 e tam bölünebilen sayıları ekrana yazdıran algoritmayı oluşturunuz.

Çözüm;

1. Başla
2. Sayı=0
3. Eğer

3. 1. (Sayı > 100) ise Adım 6 ya git

3. 2. Eğer

3. 2. 1. ((Sayı % 8)==0) ise Sayıyı ekrana yaz

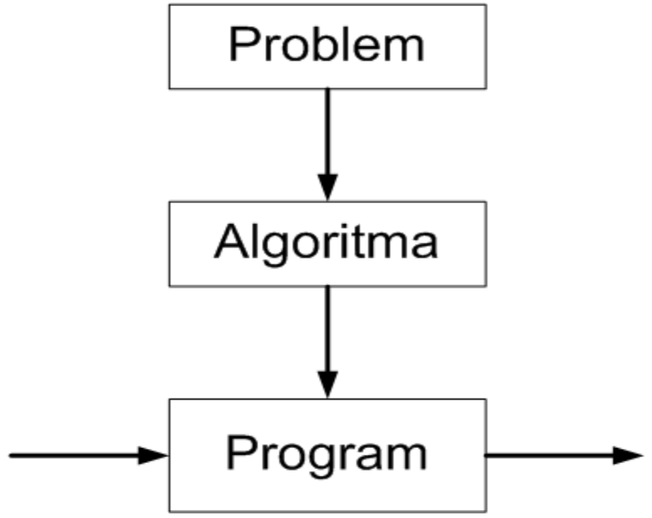
4. Sayı=Sayı + 1
5. Adım 3 e git
6. Dur

Algoritma Analizi

Algoritma Analizi Çerçevesi

Algoritma ile hedeflenen sonlu bir zaman içinde, belirli girdiler ile istenilen çıktıyı elde etmektir.

- Girdi Büyüklüğünün Ölçülmesi
- Çalışma Zamanı Ölçü Biriminin Belirlenmesi
- Büyüme Derecesi (Order of Growth)
- En kötü durum, en iyi durum, ortalama durum değerlendirmeleri



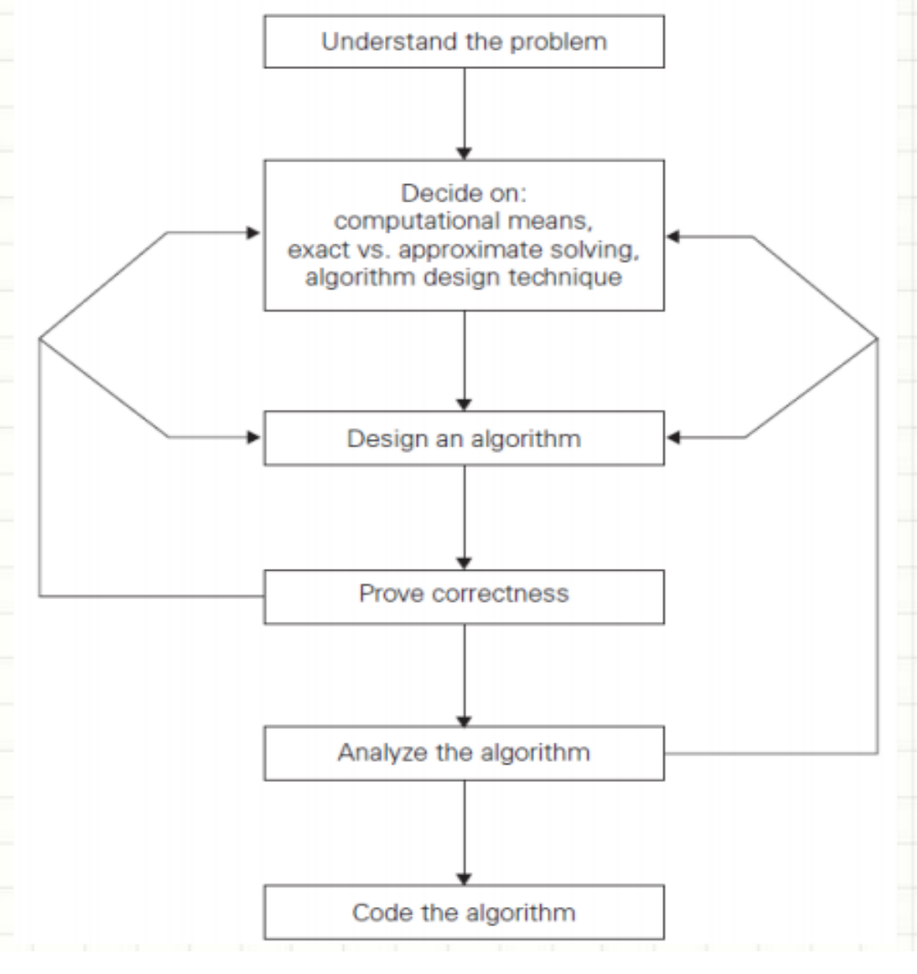
Algoritma Analizi

Bir algoritma birden fazla biçimde sunulabilmeli, net ve anlaşılır olmalıdır, etkin ve faydalı olmalıdır, sonlu veya sonlandırılabilir olmalıdır, geliştirildiği problem için doğru olmalıdır.

Algoritma Analizinde Göz Önünde Bulundurulması Gerekenler:

- Algoritmanın Doğruluğu (Correctness)
- Zaman Verimliliği (Time Efficiency)
- Bellek Alanı Verimliliği (Space Efficiency): Gelişen donanım teknolojisi ile artık çok önemli değil
- Uygunluk, en iyilik (Optimality)

Algoritma Tasarım ve Analiz Süreci



Algoritma Tasarım ve Analiz Süreci

Algoritma Tasarımı ve Veri Yapıları:

- Bellek kullanımı
- Veri türü
- Donanım?

Algoritma Açıklama Yöntemleri:

– Akış diyagramı

– Sözde kod

– Kelimeler ile anlatım

Algoritma Tasarım ve Analiz Süreci

Algoritmanın doğruluğunun kanıtlanması:

– Denemeler

– İstisnalar

– İspatlar

- Algoritmanın Analizi:

– Algoritmanın doğruluğu en önemli unsur

– Etkinlik

– Zaman Etkinliği

– Basitlik

Önemli Problem Türleri

- Sıralama
- Arama
- String İşlemleri
- Graph problemleri
- Kombinasyonel Problemler
- Geometrik problemler
- Nümerik problemler

Örnek-1: Dizideki sayıların toplamını bulma

```
int Topla(int A[], int N)
{
  int topla = 0;
  for (i=0; i < N; i++){
    topla += A[i];
  } //Bitti-for

  return topla;
} //Bitti-Topla
```

İşlem sayısı

1

N+1

N

1

Toplam: $1 + N + N + 1 + 1 = 2N + 3$

- Çalışma zamanı: $T(N) = 2N+3$
  - N dizideki sayı sayısı

Örnek-2: İç içe döngü

```
for (i=1; i<=N; i++){
    for (j=1; j<=N; j++){
        printf("Foo\n");
    } //bitti-içteki for
} //bitti-dıştaki for
```

- Prinf fonksiyonu kaç kez çalıştırıldı?
  - Veya Foo yazısı ekrana kaç kez yazılır?

$$T(N) = \sum_{i=1}^{N} \sum_{j=1}^{N} 1 = \sum_{i=1}^{N} N = (N+1)*(N+1) = N^2 + 2N + 1$$

Örnek-2: Matis Çarpımı

```
/* İki boyutlu dizi A, B, C.      Hesapla C = A*B    */
for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        C[i][j] = 0;
        for (int k=0; k<N; k++){
          C[i][j] += A[i][k]*B[k][j];
        } //bitti-en içteki for
    } //bitti-içteki for
} //bitti-dıştaki for
```

$$T(N) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(1 + \sum_{k=0}^{N-1} 1\right) = N^3 + N^2$$

Algorithm Analysis

Algorithm

- An ***algorithm*** is a set of instructions to be followed to solve a problem.
  - There can be more than one solution (more than one algorithm) to solve a given problem.
  - An algorithm can be implemented using different programming languages on different platforms.
- An algorithm must be correct. It should correctly solve the problem.
  - e.g. For sorting, this means even if (1) the input is already sorted, or (2) it contains repeated elements.
- Once we have a correct algorithm for a problem, we have to determine the efficiency of that algorithm.

Algorithmic Performance

There are *two aspects* of algorithmic performance:

- Time
  - Instructions take time.
  - How fast does the algorithm perform?
  - What affects its runtime?
- Space
  - Data structures take space
  - What kind of data structures can be used?
  - How does choice of data structure affect the runtime?

➢ We will focus on time:

– How to estimate the time required for an algorithm

Analysis of Algorithms

- ***Analysis of Algorithms*** is the area of computer science that provides tools to analyze the efficiency of different methods of solutions.
- How do we compare the time efficiency of two algorithms that solve the same problem?

  Naïve Approach: implement these algorithms in a programming language (C++), and run them to compare their time requirements. Comparing the programs (instead of algorithms) has difficulties.
  - *How are the algorithms coded?*
    - Comparing running times means comparing the implementations.
    - We should not compare implementations, because they are sensitive to programming style that may cloud the issue of which algorithm is inherently more efficient.
  - *What computer should we use?*
    - We should compare the efficiency of the algorithms independently of a particular computer.
  - *What data should the program use?*
    - Any analysis must be independent of specific data.

Analysis of Algorithms

- When we analyze algorithms, we should employ mathematical techniques that analyze algorithms independently of *specific implementations*, *computers*, *or data*.

- To analyze algorithms:
  - First, we start to count the number of significant operations in a particular solution to assess its efficiency.
  - Then, we will express the efficiency of algorithms using growth functions.

General Rules for Estimation

- **Loops**: The running time of a loop is at most the running time of the statements inside of that loop times the number of iterations.
- **Nested Loops**: Running time of a nested loop containing a statement in the inner most loop is the running time of statement multiplied by the product of the sized of all loops.
- **Consecutive Statements:** Just add the running times of those consecutive statements.
- **If/Else**: Never more than the running time of the test plus the larger of running times of S1 and S2.

Algorithm Growth Rates

- We measure an algorithm's time requirement as a function of the *problem size*.
  - Problem size depends on the application: e.g. number of elements in a list for a sorting algorithm, the number users for a social network search.
- So, for instance, we say that (if the problem size is n)
  - Algorithm A requires **$5*n^2$** time units to solve a problem of size n.
  - Algorithm B requires **$7*n$** time units to solve a problem of size n.
- The most important thing to learn is how quickly the algorithm's time requirement grows as a function of the problem size.
  - Algorithm A requires time proportional to **n^2**.
  - Algorithm B requires time proportional to **n**.
- An algorithm's proportional time requirement is known as ***growth rate***.
- We can compare the efficiency of two algorithms by comparing their growth rates.

Common Growth Rates

Function	Growth Rate Name
c	Constant
log N	Logarithmic
$\log^2 N$	Log-squared
N	Linear
N log N	Log-linear
N^2	Quadratic
N^3	Cubic
2^N	Exponential

Order-of-Magnitude Analysis and Big O Notation

- If *Algorithm A requires time proportional to g(n)*, Algorithm A is said to be **order g(n),** and it is denoted as **O(g(n)).**
- The **function g(n)** is called the algorithm's **growth-rate function**.
- Since the capital O is used in the notation, this notation is called the **Big O notation**.
- If Algorithm A requires time proportional to $\mathbf{n^2}$, it is $\mathbf{O(n^2)}$.
- If Algorithm A requires time proportional to **n**, it is **O(n).**

Definition of the Order of an Algorithm

Definition:

Algorithm A is order g(n) – denoted as O(g(n)) – if constants k and n_0 exist such that A requires no more than k*g(n) time units to solve a problem of size $n \geq n_0$. ➔ $f(n) \leq k*g(n)$ for all $n \geq n_0$

- The requirement of $\mathbf{n \geq n_0}$ in the definition of O(f(n)) formalizes the notion of sufficiently large problems.
  - In general, many values of k and n can satisfy this definition.

Order of an Algorithm

- If an algorithm requires $f(n) = n^2 – 3*n + 10$ seconds to solve a problem size n. If constants k and n_0 exist such that

  $k*n^2 > n^2 – 3*n + 10$ for all $n \geq n_0$.

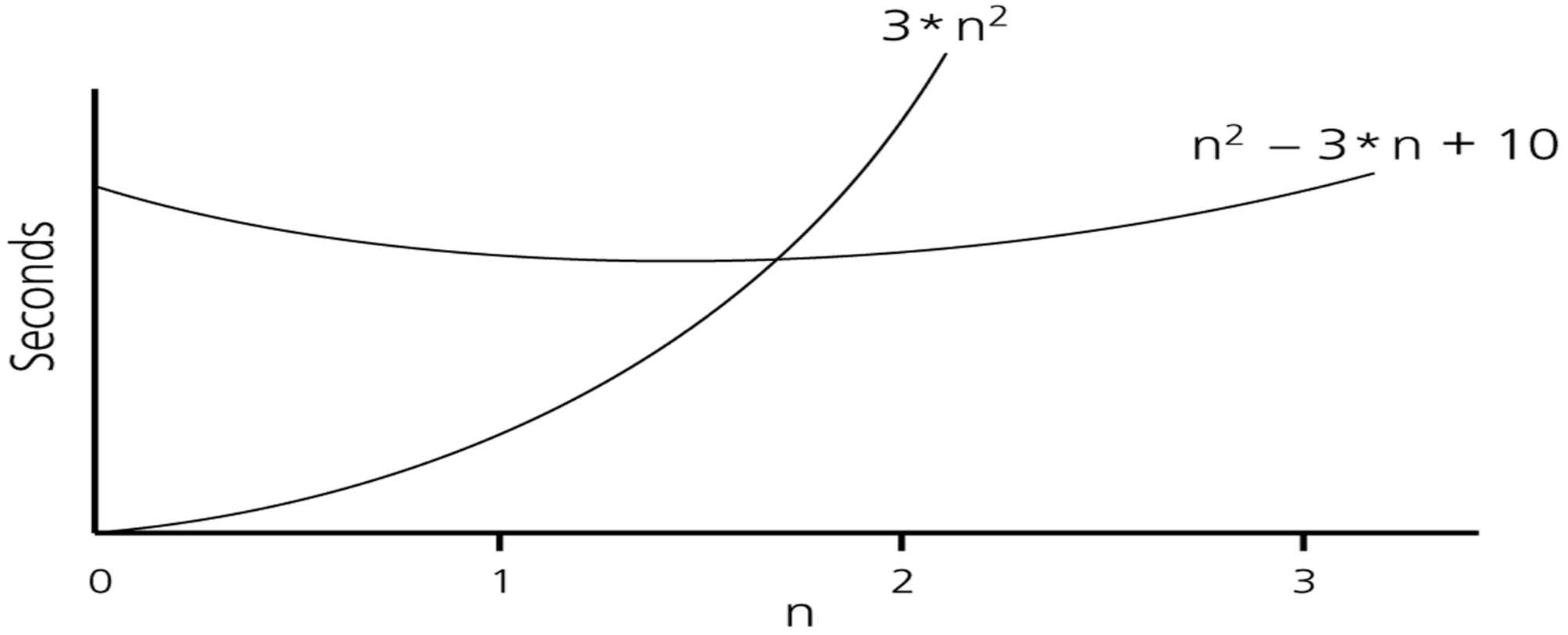
  the algorithm is order n^2 (In fact, k is 3 and n_0 is 2)

  $3*n^2 > n^2 – 3*n + 10$ for all $n \geq 2$.

  Thus, the algorithm requires no more than $k*n^2$ time units for $n \geq n_0$,

  So it is **$O(n^2)$**

Order of an Algorithm (cont.)



Order of an Algorithm

- Show $2^x + 17$ is $\mathbf{O(2^x)}$

- $2^x + \boldsymbol{17} \leq 2^x + \boldsymbol{2^x} = 2*2^x$ for $\boldsymbol{x > 5}$
- Hence k = 2 and $n_0 = 5$

Order of an Algorithm

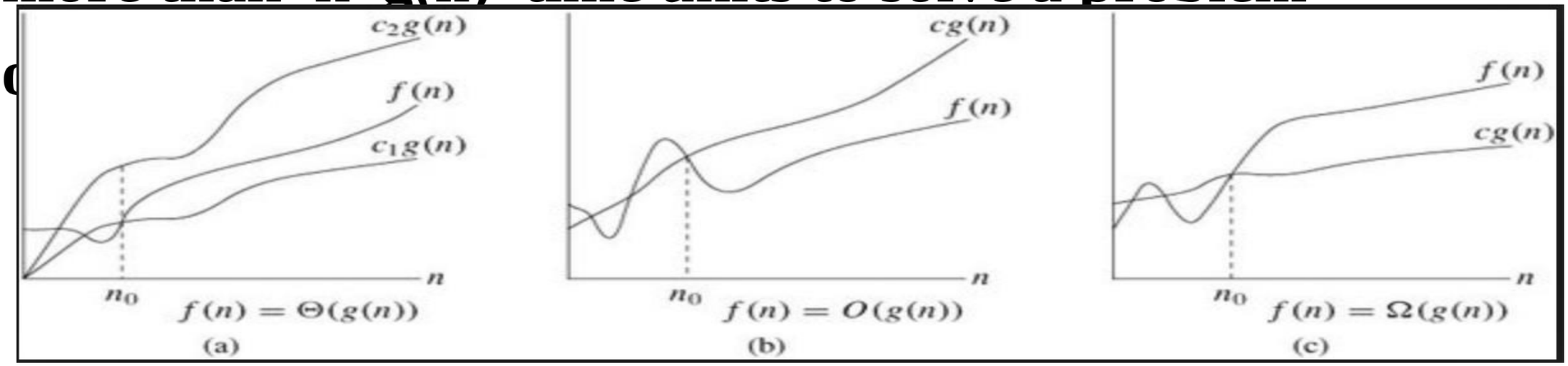
- Show $2^x + 17$ is $\mathbf{O(3^x)}$

- $2^x + \boldsymbol{17} \leq k3^x$
- Easy to see that rhs grows faster than lhs over time → k=1
- However when x is small 17 will still dominate → skip over some smaller values of x by using $n_0 = 3$
- Hence k = 1 and $n_0 = 3$

Definition of the Order of an Algorithm

Definition:

Algorithm A is omega g(n) – denoted as Ω(g(n)) – if constants k and n_0 exist such that A requires more than k*g(n) time units to solve a problem



Definition of the Order of an Algorithm

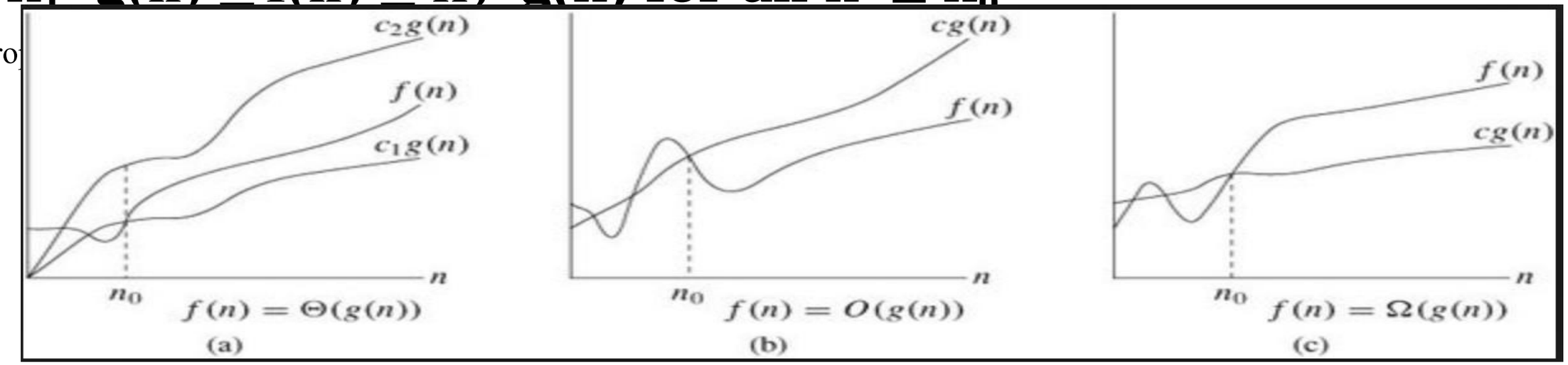
Definition:

Algorithm A is theta g(n) – denoted as Θ(g(n)) –

if constants k_1, k_2 and n_0 exist such that

$k_1 * g(n) \leq f(n) \leq k_2 * g(n)$ for all $n \geq n_0$

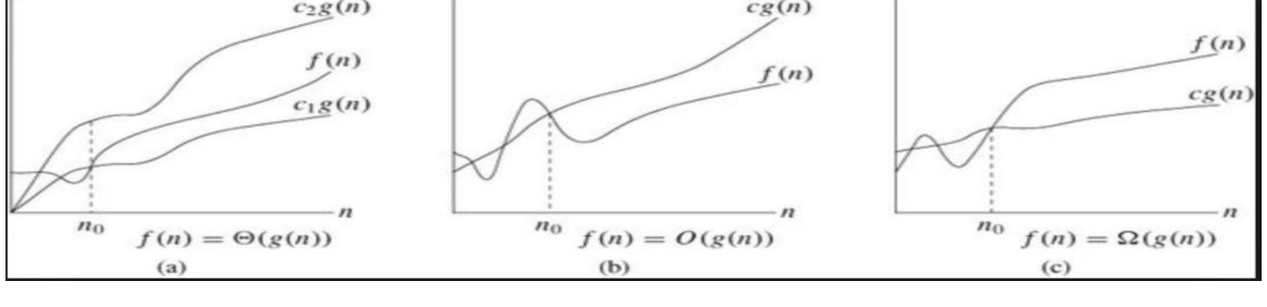
(Dro



Order of an Algorithm

- Show $f(n) = 7n^2 + 1$ is $\Theta(n^2)$

- You need to show $f(n)$ is $O(n^2)$ and $f(n)$ is $\Omega(n^2)$
- $f(n)$ is $O(n^2)$ because $7n^2 + 1 \leq 7n^2 + n^2 \; \forall n \geq 1$ ➔ $k_1 = 8$ $n_0 = 1$
- $f(n)$ is $\Omega(n^2)$ because $7n^2 + 1 \geq 7n^2 \; \forall n \geq 0$ ➔ $k_2 = 7$ $n_0 = 0$
- Pick the largest n_0 to satisfy both conditions naturally ➔ $k_1 = 8$, $k_2 = 7$, $n_0 = 1$



A Comparison of Growth-Rate Functions

(a)

	n					
Function	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

Growth-Rate Functions

O(1) Time requirement is **constant**, and it is independent of the problem's size.

O($\log_2 n$) Time requirement for a **logarithmic** algorithm increases increases slowly as the problem size increases.

O(n) Time requirement for a **linear** algorithm increases directly with the size of the problem.

O($n*\log_2 n$) Time requirement for a **$n*\log_2 n$** algorithm increases more rapidly than a linear algorithm.

O(n^2) Time requirement for a **quadratic** algorithm increases rapidly with the size of the problem.

O(n^3) Time requirement for a c**ubic** algorithm increases more rapidly with the size of the problem than the time requirement for a quadratic algorithm.

O(2^n) As the size of the problem increases, the time requirement for an **exponential** algorithm increases too rapidly to be practical.

Growth-Rate Functions

- If an algorithm takes 1 second to run with the problem size 8, what is the time requirement (approximately) for that algorithm with the problem size 16?
- If its order is:

  O(1) ➔ T(n) = 1 second

  O($\log_2 n$) ➔ T(n) = (1*$\log_2 16$) / $\log_2 8$ = 4/3 seconds

  O(n) ➔ T(n) = (1*16) / 8 = 2 seconds

  O(n*$\log_2 n$) ➔ T(n) = (1*16*$\log_2 16$) / 8*$\log_2 8$ = 8/3 seconds

  O(n^2) ➔ T(n) = (1*16^2) / 8^2 = 4 seconds

  O(n^3) ➔ T(n) = (1*16^3) / 8^3 = 8 seconds

  O(2^n) ➔ T(n) = (1*2^{16}) / 2^8 = 2^8 seconds = 256 seconds

Properties of Growth-Rate Functions

1. *We can ignore low-order terms in an algorithm's growth-rate function.*
   - If an algorithm is $O(n^3+4n^2+3n)$, it is also $O(n^3)$.
   - We only use the higher-order term as algorithm's growth-rate function.

2. *We can ignore a multiplicative constant in the higher-order term of an algorithm's growth-rate function.*
   - If an algorithm is $O(5n^3)$, it is also $O(n^3)$.

3. *$O(f(n)) + O(g(n)) = O(f(n)+g(n))$*
   - We can combine growth-rate functions.
   - If an algorithm is $O(n^3) + O(4n)$, it is also $O(n^3 +4n^2)$ ➔ So, it is $O(n^3)$.
   - Similar rules hold for multiplication.

What to Analyze

- An algorithm can require different times to solve different problems of the same size.
  - Eg. Searching an item in a list of n elements using sequential search. ➔ Cost: 1,2,...,n
- ***Worst-Case Analysis*** –The maximum amount of time that an algorithm require to solve a problem of size n.
  - This gives an upper bound for the time complexity of an algorithm.
  - Normally, we try to find worst-case behavior of an algorithm.
- ***Best-Case Analysis*** –The minimum amount of time that an algorithm require to solve a problem of size n.
  - The best case behavior of an algorithm is NOT so useful.
- ***Average-Case Analysis*** –The average amount of time that an algorithm require to solve a problem of size n.
  - Sometimes, it is difficult to find the average-case behavior of an algorithm.
  - We have to look at all possible data organizations of a given size n, and their distribution probabilities of these organizations.
  - ***Worst-case analysis is more common than average-case analysis.***

Sequential Search

```
int sequentialSearch(const int a[], int item, int n){
  for (int i = 0; i < n && a[i]!= item; i++);
  if (i == n)
     return -1;
  return i;
}
```

Unsuccessful Search: ➔ O(n)

Successful Search:

Best-Case: *item* is in the first location of the array ➔O(1)

Worst-Case: *item* is in the last location of the array ➔O(n)

Average-Case: The number of key comparisons 1, 2, ..., n

$$\frac{\sum_{i=1}^{n} i}{n} = \frac{(n^2+n)/2}{n}$$

Binary Search

We can do binary search if the array is sorted:

```
int binarySearch(int a[], int size, int x) {
   int low =0;
   int high = size -1;
   int mid;      // mid will be the index of
          // target when it's found.
   while (low <= high) {
     mid = (low + high)/2;
     if (a[mid] < x)
         low = mid + 1;
     else if (a[mid] > x)
       high  = mid - 1;
      else
       return mid;
   }
```

Binary Search – Analysis

- For an unsuccessful search:
  - The number of iterations in the loop is $\lfloor \log_2 n \rfloor + 1$

    ➔ $O(\log_2 n)$
- For a successful search:
  - ***Best-Case:*** The number of iterations is 1. ➔ $O(1)$
  - ***Worst-Case:*** The number of iterations is $\lfloor \log_2 n \rfloor + 1$ ➔ $O(\log_2 n)$
  - ***Average-Case:*** The avg. # of iterations $< \log_2 n$ ➔ $O(\log_2 n)$

0	1	2	3	4	5	6	← an array with size 7
3	2	3	1	3	2	3	← # of iterations

The average # of iterations = 17/7 = 2.4285 < $\log_2 7$

How much better is $O(log_2 n)$?

n	$O(log_2 n)$
16	4
64	6
256	8
1024	10
16,384	14
131,072	17
262,144	18
524,288	19
1,048,576	20
1,073,741,824	30

Usage Notes

- A lot of slides are adopted from the presentations and documents published on internet by experts who know the subject very well.
- I would like to thank who prepared slides and documents.
- Also, these slides are made publicly available on the web for anyone to use
- If you choose to use them, I ask that you alert me of any mistakes which were made and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides.

Sincerely,
Dr. Cahit Karakuş
cahitkarakus@gmail.com