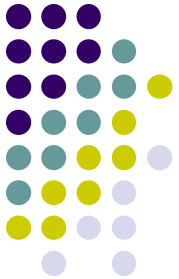
ROAD MAP

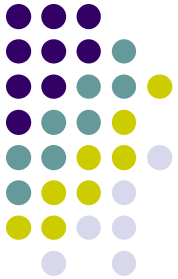
- **Matematiksel Altyapı**
- **Temel Veri Yapıları**
  - **Linear Data Structures**
  - **Graphs**
  - **Trees**
- Algoritma nedir?
- Algoritma Analizi
- Farklı Problemler ve bunların analizi
- Çalışma Zamanı Fonksiyonları

Matematiksel Altyapı



- Fonksiyonlar
- Logaritma
- Toplama
- Olasılık
- Asymptotic Notasyonlar
- Recursion
  - Recurrence equation

Temel Veri Yapıları



- Bir veri yapısı birbiriyle ilişkili veri elemanlarını organize etmeye yarar.

- Linear Veri Yapıları
  - Array
  - Linked list
  - Stack
  - Queue
  - Prioritiy Queue
- Graphs
- Trees

Linear Veri Yapıları

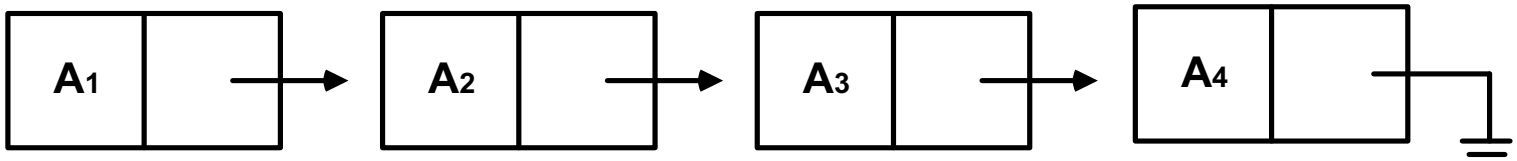
- **Array (Dizi)**
  - Bir dizi, n aynı veri türünün ardışıl olarak bilgisayar belleğinde saklandığı veri yapısıdır.
  - Bir indexi (indisi) belirtilerek diziye erişilebilir.

Item[0]	*Item[1]*	…	*Item[n-1]*

n elemanlı bir array

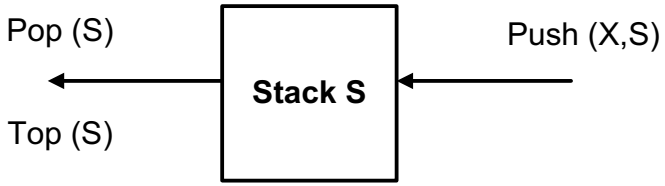
Linear Veri Yapıları

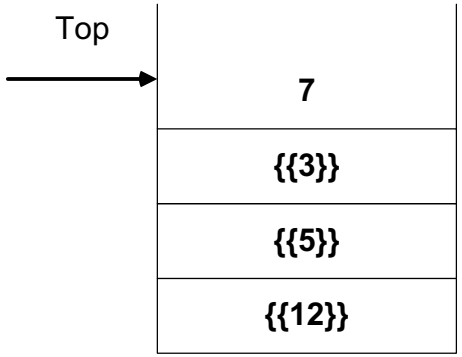
- **Linked List (Bağlı Liste)**
  - Bağlantılı bir liste, düğüm adı verilen sıfır veya daha fazla öğenin dizilimidir.
  - Her düğüm iki tür bilgi içerir :
    - Veri
    - Bağlantılı listenin diğer düğümlerine işaretçiler (pointer) olarak adlandırılan bir veya daha fazla bağ



Linear Veri Yapıları

- **Stack (Yığın)**
  - Bir yığın, eklemelerin ve silme işlemlerinin yalnızca sondan yapılabileceği (üst olarak adlandırılır) bir listedir.
  - Last In First Out(LIFO))

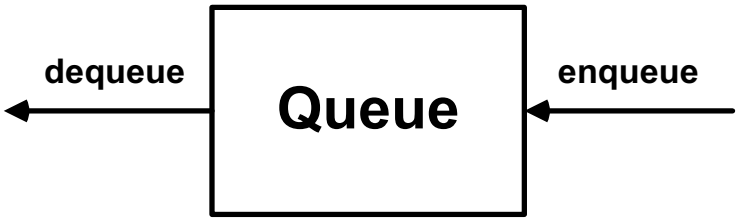




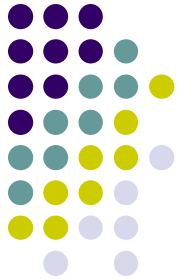
Linear Veri Yapıları

- **Queue (Kuyruk)**
  - **Queue**, elemanların yapının ön tarafından silindiği bir yapıdır.

    dequeue işlemi
  - *Sıranın , arka adı verilen diğer ucuna yeni elemanlar eklenir*
    - *enqueue operation*
  - First In First Out (FIFO)

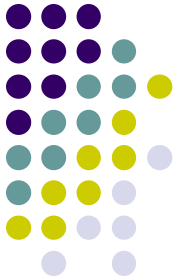


Linear Veri Yapıları



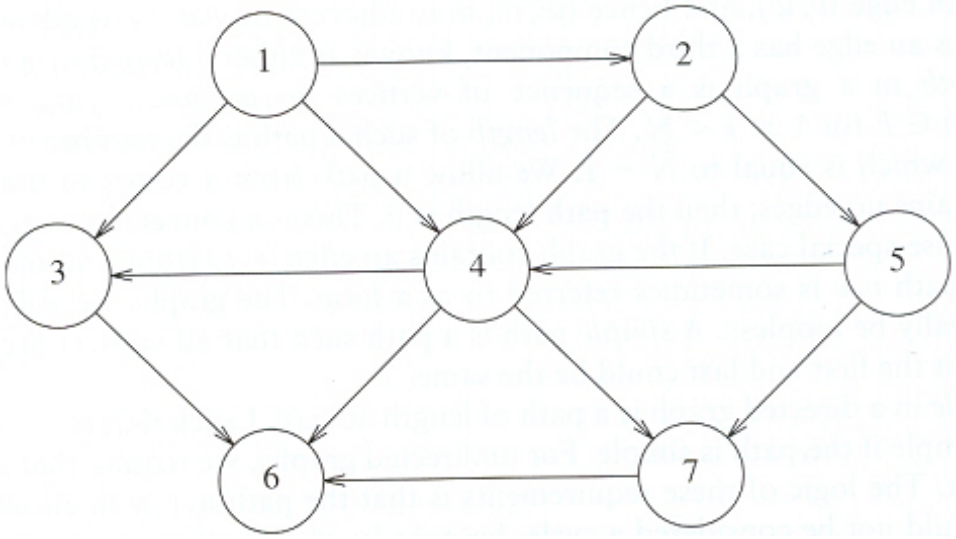
- **Priority Queue (Öncelik Kuyruğu)**
  - Bir öncelik sırası, tamamen karşılaştırılabilir bir evrenin veri öğelerinin bir listesidir.
    - Örneğin. tam sayı veya reel sayı
    - Dinamik olarak değişen adaylar grubu arasında en yüksek önceliğe sahip bir öğe seçilmesi gerekir
    - İşlemler:
    - Insert → adding a new element
    - Delete → deleting largest/smallest element
    - Search → find largest/smallest element
  - Bir priority queue uygulaması da, heap adı verilen usta bir veri yapısına dayalıdır

Graphs (Çizgeler)



- Bir *graph G=(V,E)* ikilisini içerir.
  - V : *vertices (köşeler)* , *nodes (düğümler)*
  - E : *edges (kenarlar)*
    - Her kenar bir ikilidir (*v,w*), | *v,w* Є V
  - Eğer bu ikililer sıralı ise çizge yönlü bir çizge yani directed bir graphtır.
    - Bazen *digraphs olarak adlandırılırlar.*
  - Eğer kenar ikilileri sıralı değilse çizge yönsüz undirected'dır.
  - *W düğümü* v ye komşu ise *(v,w) Є E olmalıdır.*

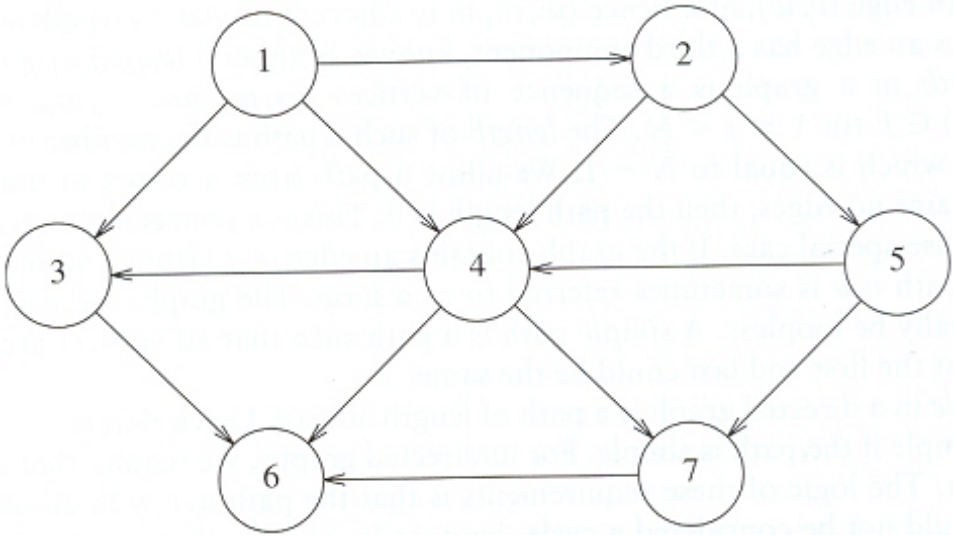
Graphs



7 köşe (vertex) ve 12 kenarlı (edges) bir graph

Graphs

- Bir path (yol), v'den w'ye kadar bir düğümler (vertices, nodes) dizisidir
- Bir path'in tüm kenarları farklıysa, bu pathin basit olduğu söylenir
- *Uzunluk, bir path'in kenarlarının (edgelerinin) toplam sayısıdır*
- Bir cycle (döngü) uzunluğu ≥ 1 olan bir yoldur; burada v= w olmalıdır.
- Bir çizgede döngü yoksa bu çizge *acyclictir*.

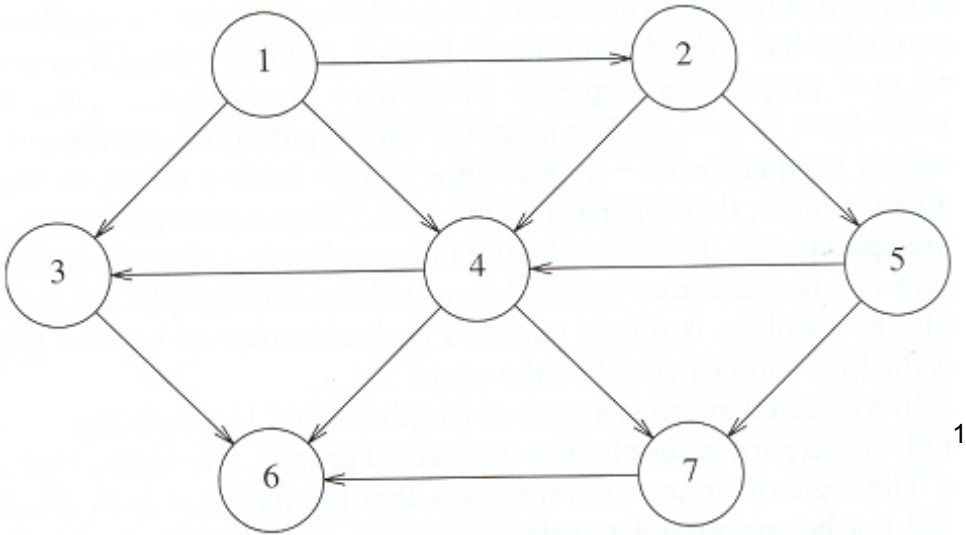


Graphs (Çizgeler)

- Her düğümün her düğümebağlı olduğu çizgeye *complete graph denilmektedir.*

- Çok yüksek sayıda edge'e sahip olan çizgeye (dense) graph denilmektedir.

- Vertex sayısına oranla çok az sayıda edge'e sahip olan graph'a ise sparse graph denilmektedir.
- A graph is *connected* if for every pair of vertices *u* and *v* there is a path from *u* to *v*



Graph Gösterimleri

Adjacency Matrix Gösterimi

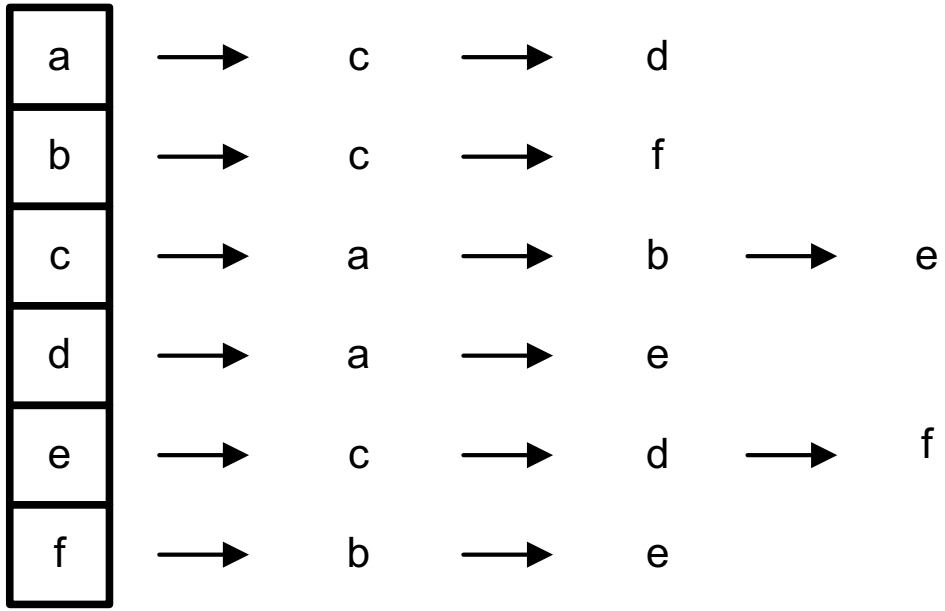
- n x n bir binary matristir.
  - *i. satır ve j. sütun, i. vertexten j. vertex'e bir kenar varsa 1'e eşittir*
  - *ikininci satır ve j sütun, eğer böyle bir kenar yoksa 0'a eşittir*

	a	b	c	d	e	f
a	0	0	1	1	0	0
b	0	0	1	0	0	1
c	1	1	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	1	0	1
f	0	1	0	0	1	0

Graph Gösterimleri

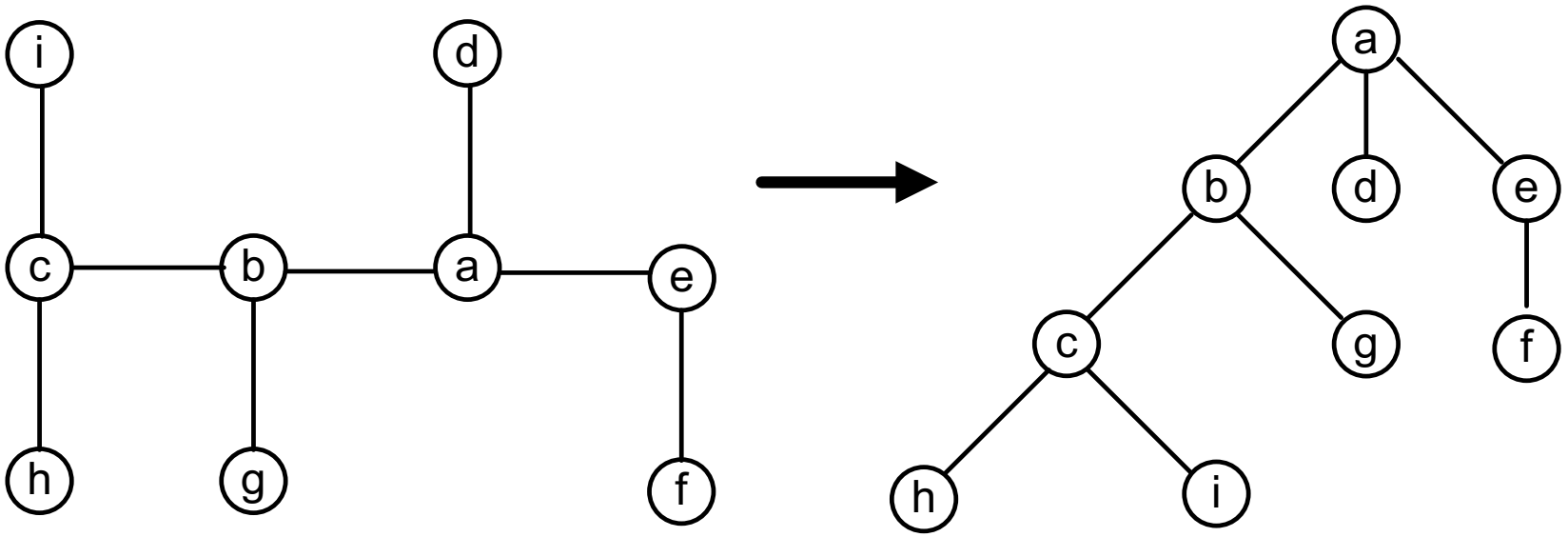
Adjacency List Gösterimi

- Listenin köşesine bitişik tüm köşeleri içeren, bağlantılı listelerden oluşan ve her köşe başlığına ait bir koleksiyon
- Grafik yoğun değilse (seyrektir) bitişik liste gösterimi daha iyi bir çözümdür.



Trees (Ağaçlar)

- Ağaç, bağlı bir acyclic bir graftır
- .*rooted tree*
  - Root denilen özel bir düğüme sahiptir.



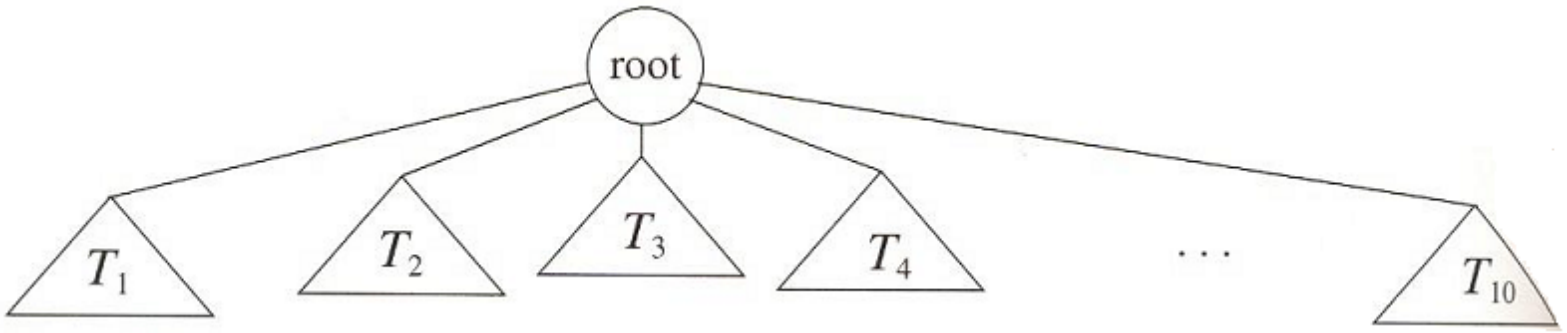
tree

rooted tree

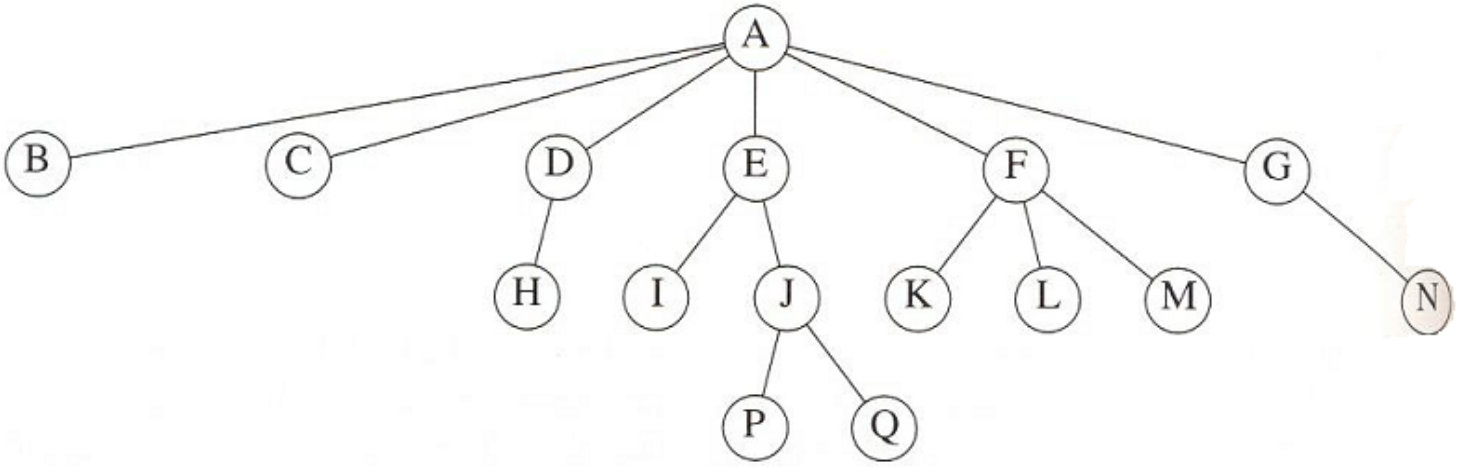
Trees (Ağaçlar)

Recursive Definition of Rooted Trees:

- **Tree** bir **node'lar** koleksiyonudur**.**
  - A tree can be empty (Tree boş olabilir)
  - Bir tree 0 veya daha çok **subtree'ye sahip olabilir** T_1, T_2,… T_k connected to a **root node** by **edges**

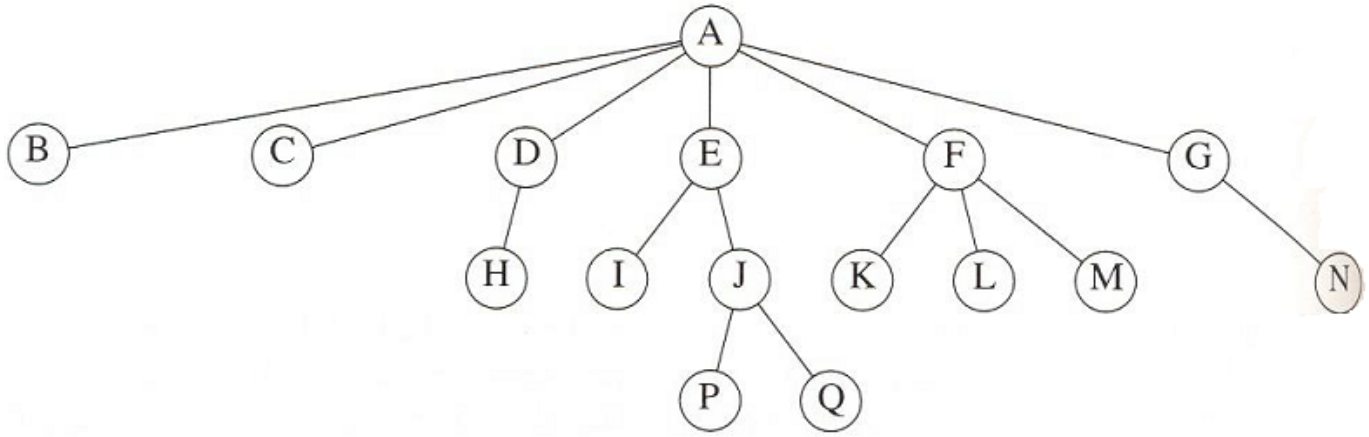


Trees - Terminology



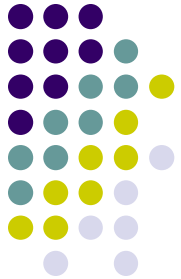
Family Tree Terminolojisi

- child → F A'nın child'idir.
- parent → A, F'nin parent'ıdır.
  - Root dışındaki her düğüm bir parenta bağlıdır.
  - sibling → Aynı parent'a sahip düğümler(K, L, M)
- leaf → Çocuğu olan düğümler (P, Q)
- Ancestor / Descendant



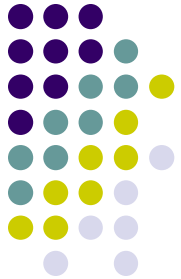
- **Path :** $n_1, n_2, \ldots n_k$ şeklinde bir düğüm dizilimidir. Burada bu dizilim n_i düğümü n_{i+1} düğümünün parentıdır. $1 \leq i < k$
- **Lenght :** path üzerindeki edge sayısıdır. (k-1)
- **Depth :** roottan n_i 'ye olan tek (unique) yolun length'idir. Root'un depthi 0'dır.
  - Bir treenin depth'i ise roota en uzak olan düğümün depth'idir. **Height :** n_i yüksekliği, n_i'den bir yaprağa uzanan en uzun yolun uzunluğudur.
  - height of a tree = height of the root = depth of the tree

Ağaçlar



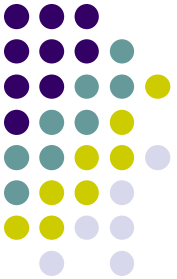
- Ordered (Sıralı) tree
  - Her köşenin tümünün sıralı olduğu rooted bir tree.
  - Binary tree
    - Hiçbir düğümün ikiden fazla olmadığı sıralı bir ağaç.
  - Her child parentının sol ya da sağ çocuğu olmaktadır.

Ağaçlar



- Binary Search Tree (BST)
  - Bir binary tree
    - Hiç tekrarnanan elemanı yok.
  - Search Tree özelliğini sağlamaktadır.
    - Sol alt ağaçtaki öğeler kökten daha küçük
    - Sağ alt ağaçtaki öğeler kökten daha büyük
    - Sağ ve sol alt ağaçların kendisi de bir BST
- Heap
  - Priority queue uygulamak için
  - İkili bir ağaç yani binary bir tree.
  - Heap order özelliğini sağlamaktadır.
    - Her eleman parentından büyüktür.

ROAD MAP



- Matematiksel Altyapı
- Temel Veri Yapıları
  - Linear Data Structures
  - Graphs
  - Trees
- Algoritma nedir?
- Algoritma Analizi
- Farklı Problemler ve bunların analizi
- Çalışma Zamanı Fonksiyonları

Algoritma nedir ?

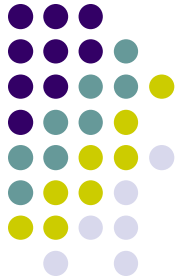
Bir algoritma, bir problemi çözmek ya da bir işlevi hesaplamak için izlenecek sonlu, açıkça belirtilen talimat dizisidir

Bir algoritma genel olarak

- Bir (birkaç) girdi alır.
- Sınırlı bir süre içerisinde komutları yerine getirmelidir.
- Bir çıktı üretmektedir.

Etkili bir komut, temelde kalem ve kağıt kullanarak gerçekleştirmenin mümkün olduğu kadar basit bir işlemdir.

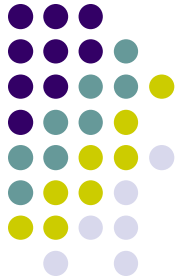
Algoritmaları İfade Etmek



Algoritmalar şu şekilde gösterilebilir:

- **doğal diller**
  - ayrıntılı ve belirsiz
  - nadiren karmaşık veya teknik algoritmalar için kullanılır
- **pseudocode, flowcharts**
  - algoritmaları ifade etmek için yapısal yöntemler.
  - doğal dilde ifadelerde belirsizliklerden kaçınır.
  - belirli bir uygulama dilinden bağımsız
- **programming languages**
  - algoritmaları bir bilgisayar tarafından yürütülebilecek biçimde ifade etmeyi amaçlayar
  - algoritmaları belgelemek için kullanılabilir

Örnek:



Problem: Sıralanmamış bir listede en büyük elemanı bulmak

Fikir: Look at every number in the list, one at a time.

Natural Language:

- Listedeki ilk elemanın en büyük olduğunu varsay.
- Listenin sonuna kadar daha büyük bir sayı var mı diye ara.
- Liste tarama işlemi bittiğinde en son not edilen en büyük elemandır.

Örnek:

Pseudocode:

Algorithm LargestNumber

Input: A non-empty list of numbers *L*.

Output: The *largest* number in the list *L*.

largest ← L_0

for each *item* **in** the list $L_{i \geq 1}$, **do**

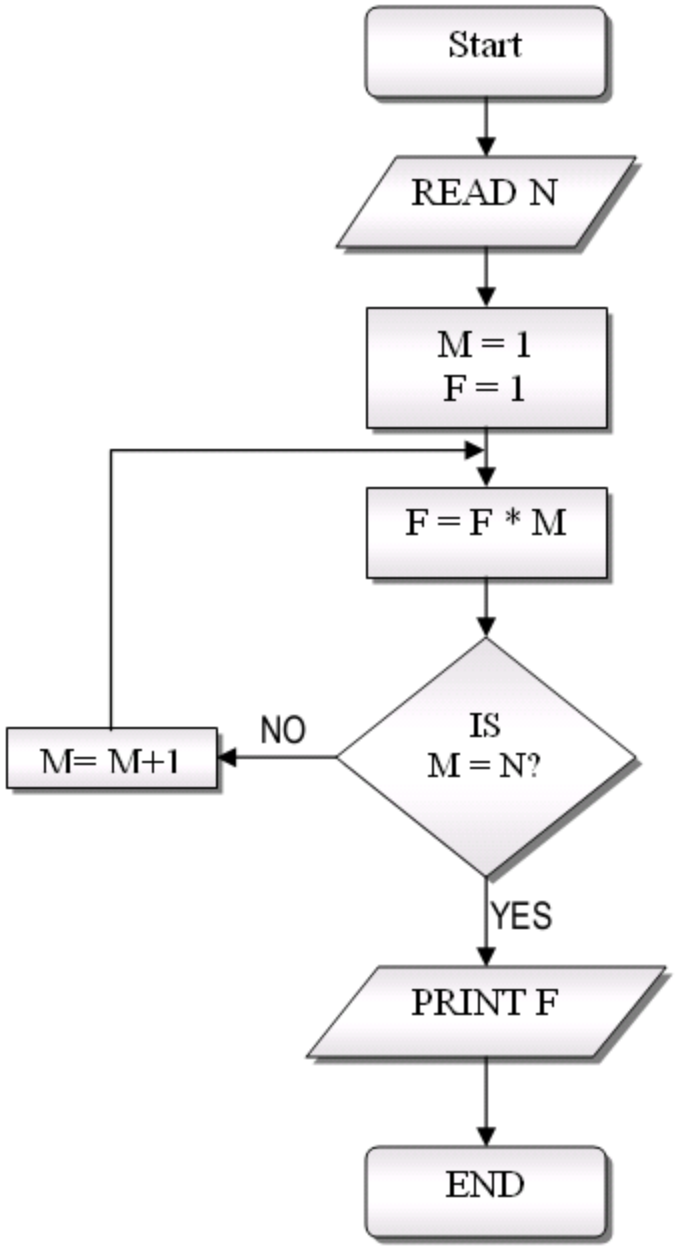
if the *item* > *largest*, **then**

largest ← the *item*

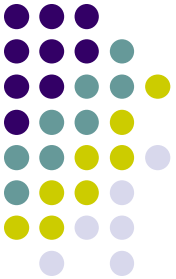
return *largest*

Example:

Flowchart:

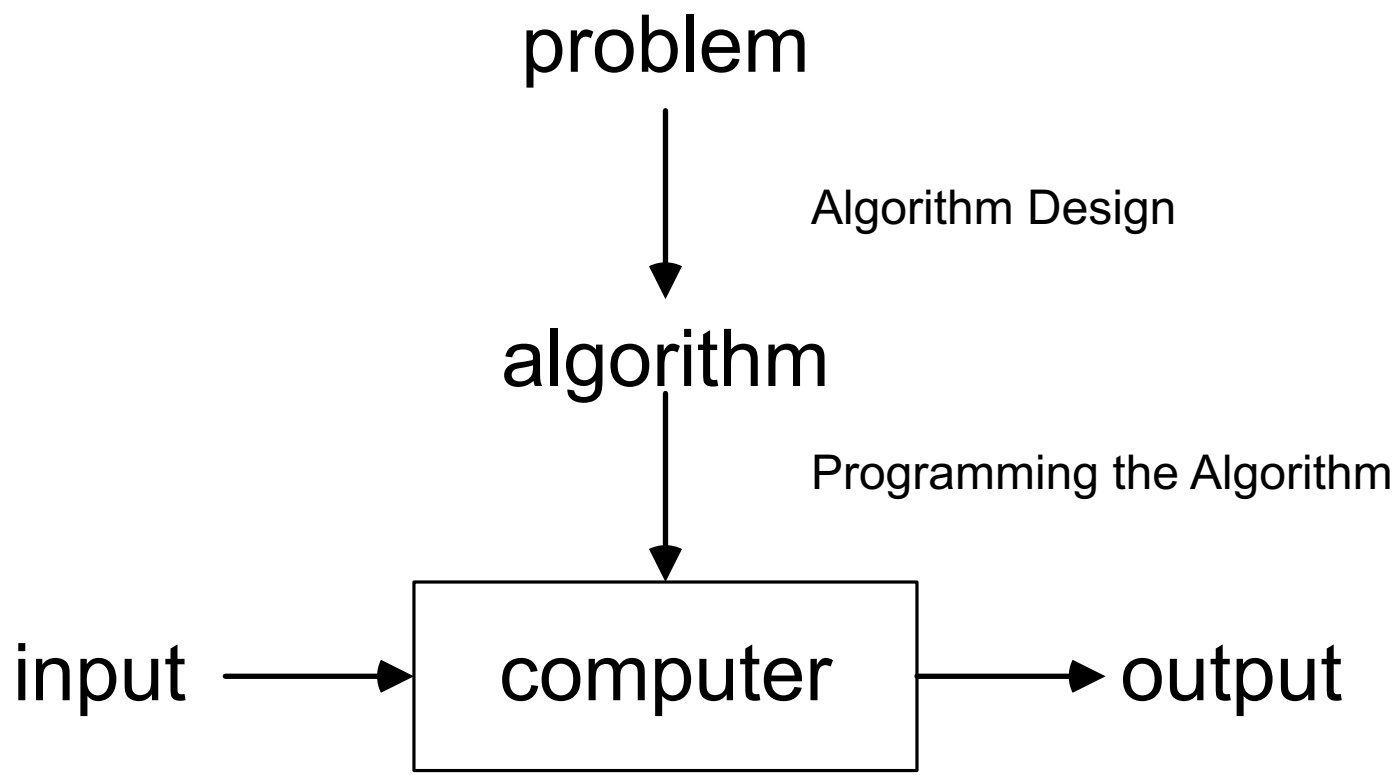


Algoritmanın Özellikleri

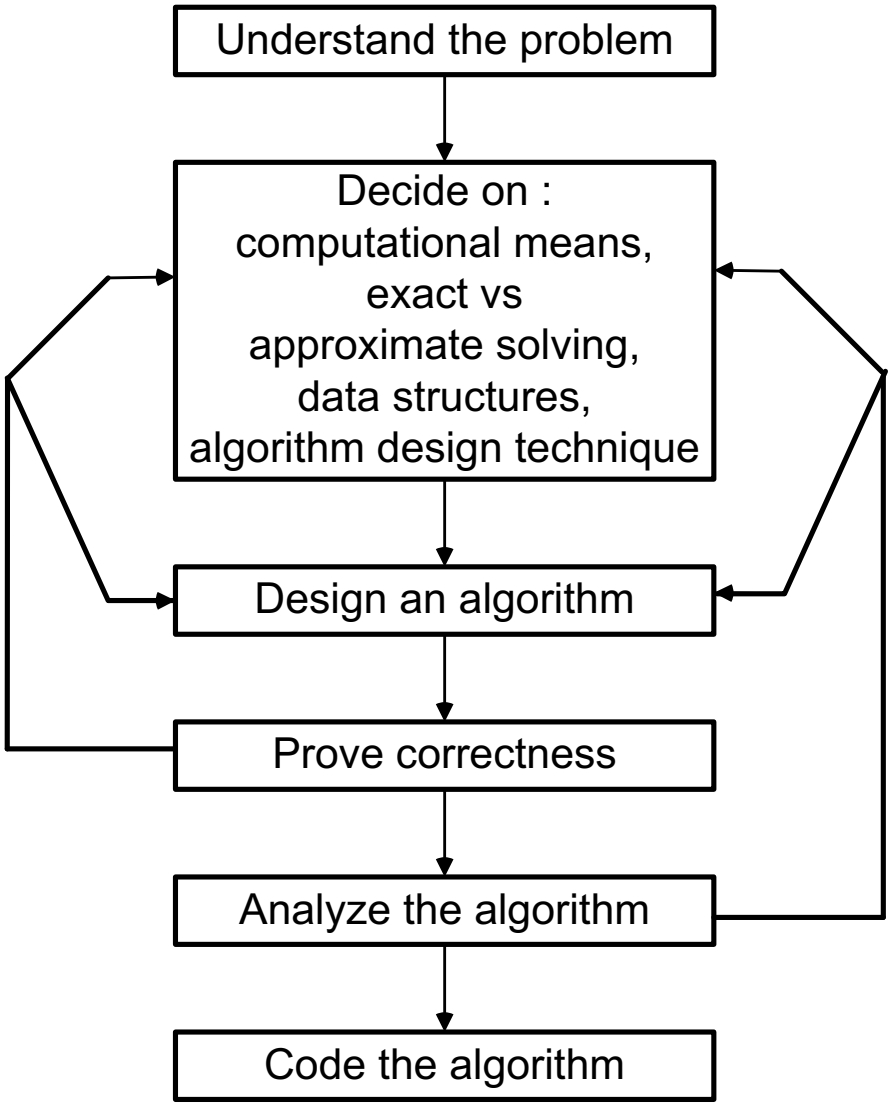


- **Effectiveness (Etkinlik)**
  - Talimatlar basit olmalı
  - Kalem ve kağıtla yazılabilir
- **Definiteness (Kesinlik)**
  - Talimatlar net
  - Anlamı tek olmalı
- **Correctness (Doğruluk)**
  - Algoritma doğru cevabı verir
  - Olası tüm durumlar için
- **Finiteness (Sonluluk)**
  - Algoritma makul sürede durmalı
  - Ve bir çıktı üretmelidir.

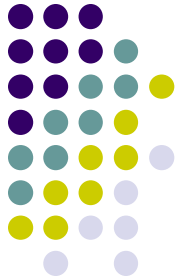
Notion of an Algorithm



Algoritma Tasarım Processi

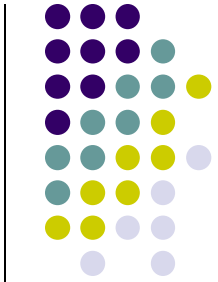


Algoritmaların Analizi



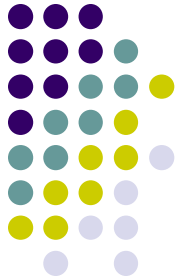
- Bir algoritmanın complexity'sini (karmaşıklığı) çalışma
  - Algoritma ne kadar iyi?
  - Diğer algoritmalarla karşılaştırma işlemi nasıl yapılacak?
  - En iyi yazılabilecek algoritma bu mudur?

Analysis of Algorithms



- Complexities
  - Space
    - Bit sayısı
    - Eleman sayısı
  - Time
    - Toplamda çalıştırılacak işlem sayısı
      - Modele göre değişir
      - RAM
      - Turing Machines

Algoritmaların Run-Time (Çalışma-Zamanı) Analizi



- Algoritma karmaşıklığı, problemin boyutunu gösteren parametre n'nin bir bir fonksiyonu olarak hesaplanabilmektedir.

- Zaman karmaşıklığı, T (n), algoritmanın en önemli işlemi olan - temel işlem olarak adlandırılan – işlemin çalıştırılma sayısı olarak hesaplanabilir.

- Space (Alan) karmaşıklığı, S (n), genellikle algoritmanın yürütülmesi sırasında kullanılan bellek alanının büyüklüğü olarak hesaplanır.

Complexity'lerin Tipleri

- Worst case

$$T(n) = max_{|I|=n}\{T(I)\}$$

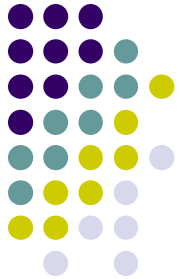
- Average case

$$T(n) = \sum_{|I|=n} T(I).Prob(I)$$

- Best case

$$T(n) = min_{|I|=n}\{T(I)\}$$

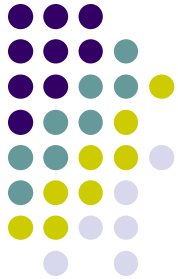
Tablo Yöntemi



- *Tablo Metodu, bir algoritmanın karmaşıklığını hesaplamak için kullanılır*
- Örnek: Bir dizinin elemanlarını toplama

	steps/exec	freq	total
`sum = 0`	1	1	1
`for i = 1 to n`	1	n+1	n+1
`    sum = sum + a[i]`	1	n	n
`report sum`	1	1	1
			2n+3

Tablo Metodu

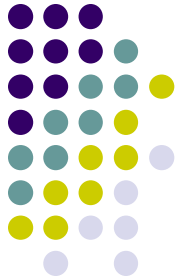


- *Tablo Metodu, bir algoritmanın karmaşıklığını hesaplamak için kullanılır*
- Örnek: Bir dizinin elemanlarını toplama

	steps/exec	freq	total
`sum = 0`	1	1	1
`for i = 1 to n`	1	n+1	n+1
`    sum = sum + a[i]`	1	n	n
`report sum`	1	1	1
			2n+3

Temel işlem nedir? Kaç defe işletilmiştir?

Tablo Metodu

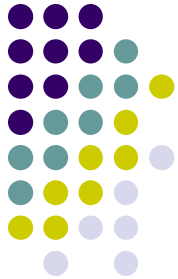


- *Tablo Metodu, bir algoritmanın karmaşıklığını hesaplamak için kullanılır*
- Örnek: Bir dizinin elemanlarını toplama

	steps/exec	freq	total
`sum = 0`	1	1	1
`for i = 1 to n`	1	n+1	n+1
`    sum = sum + a[i]`	1	n	n
`report sum`	1	1	1
			2n+3

Basic operation is executed n times and n is proportional to 2n+3

Tablo Metodu



- Örnek:

  Matris Toplama

  a, b, c 'nin mxn boyutunda matrisler olduğunu varsayalım.

	steps/exec	freq	total
for i = 1 to m	1	m+1	m+1
for j = 1 to n	1	m(n+1)	mn+m
c[i,j] = a[i,j] + b[i,j]	1	mn	mn
			2mn+2m+1

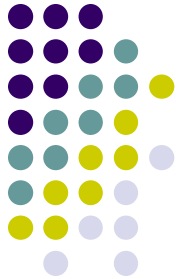
TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Örnek

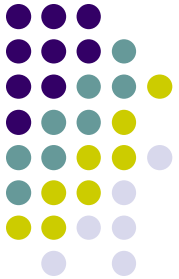
- Örneğin, 2^100'ü hesaplamak saniyede 10^{12} işlem yapan bir bilgisayar için 4x10^^{10} yıl alacaktır.
- 2^100 , 100! değerini hesaplamak için gereken süreden kısa, 100!'i hesaplamak ise dünya gezegeninin tahmini yaşından 4,5 milyar (4.5 .10^9) yıl daha uzun sürecektir.
- 2^n ve n! fonksiyonlarının büyüme sıraları arasında muazzam bir fark var.

Algoritmaların Time (Zaman) Complexity'si (Karmaşıklığı)

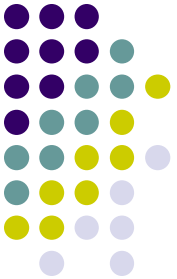


- Best Case (En iyi durum)
- Worst Case (En Kötü Durum)
- Average Case (Ortalamada)

ROAD MAP

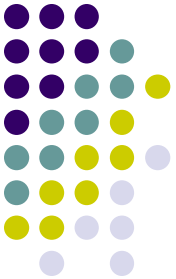


ROAD MAP



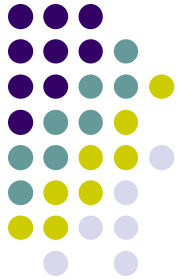
- Matematiksel Altyapı
- Temel Veri Yapıları
  - Linear Data Structures
  - Graphs
  - Trees
- Algoritma nedir?
- Algoritma Analizi
- Farklı Problemler ve bunların analizi
- Çalışma Zamanı Fonksiyonları

Önemli Problem Tipleri



- Sorting (Sıralama)
- Searching (Arama)
- String Processing (String İşleme)
- Graph Problems (Çizge Problemleri)
- Combinatorial Problems (Kombinasyon Problemleri)
- Geometric Problems (Geometrik Problemler)
- Numerical Problems (Nümerik Problemler)

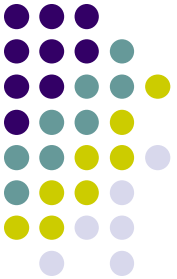
Searching



Problem tanımı:

- Bir listede aranan ‘key’’i bulmak.

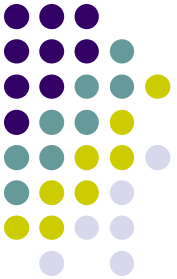
Lineer Arama (Sequential Search)



Yaklaşım

1. start with the first element
2. if the element is the search key
   return the position
3. otherwise continue with the next element
4. return fail if the end of list reached

Lineer Arama



Pseudo – code

Given A[1], … , A[n] and *search key*

```
1. i=1
2. while i ≤ n
3.       if A[i] = key return i
4.       else i = i+1
5. return fail
```

best, worst and average caseleri nelerdir ?

Lineer Arama

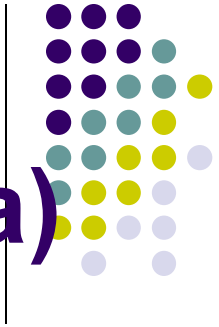
Algoritmanın Zaman (Time) Karmaşıklığı (Complexity)'si:

- Best case (En iyi Durum)

  A[1] = key
- Worst case (En kötü Durum)

  A[i] ≠ key for any key
  - Zamanı eleman sayısı ile orantılıdır.
  - Ardışıl aramanın zaman karmaşıklığı O(n)
- Average case (Ortalama Durum)
  - Tüm aramalar eş dağılımla gerçekleştiriliyorsa ~ n/2

Insertion Sort (Eklemeli Sıralama)

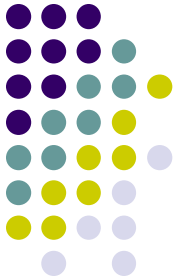


- **Hedef**
  - Verilen A[1] … A[n], listesini sıralamak

- **Yaklaşım (j. index açısından bakıldığında)**

1. The sequence A[1] is sorted
2. Suppose A[1] ≤ A[2] … ≤ A[j-1] are already sorted
3. Pick up the next element and place it with its appropriate place

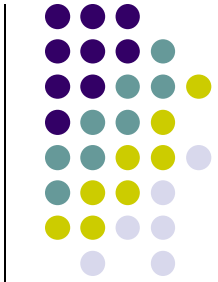
Insertion Sort



- **Pseudo code**

```
1. for j=2 to n
2.     key = A[j]
3.     i = j-1
4.     while i>0 and A[i]>key
5.         A[i+1] = A[i]
6.         i = i-1
7.     A[i+1] = key
```

Insertion Sort

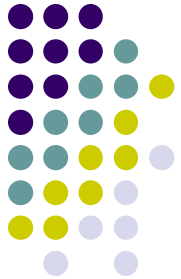


- **Analysis**

Step	Cost	Time
1	c_1	n
2	c_2	n-1
3	c_3	n-1
4	c_4	$\sum_{j=2}^{n} t_j + 1$
5	c_5	$\sum_{j=2}^{n} t_j$
6	c_6	
7	c_7	n-1

t_j = number of times the while loop executes for j

Insertion Sort



t_j problemin örneğine göre değişmektedir (girdi)

Best-case

A sıralı olduğunda

$t_j=0$ for all j

T(n) = linear function in n

= O(n)

Insertion Sort

Worst-case

A tersten sıralı olduğunda

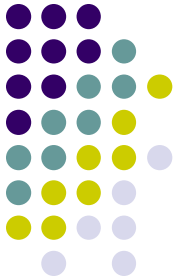
$t_j = j$ *for* $j=2,\ldots,n$

$$\sum_{j=1}^{n} t_j = \sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

T(n) = quadratic function

= $O(n^2)$

Insertion Sort



Ortalama Durum zaman karmaşıklığı (Average-case)

Dizi rastgele bir şekilde oluşturulmuş olsun.

Where in A[1…j-1] will A[j] be inserted ?

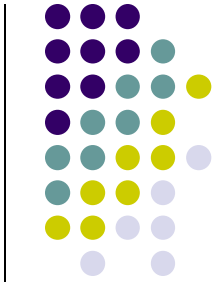
$$t_j \approx \frac{j}{2}$$

Roughly half way i.e.

$$\sum_{j=1}^{n} t_j = \sum_{j=1}^{n} \frac{j}{2} = \frac{1}{4}(n^2 + n)$$

T(n) → quadratic

Polynomial Değerlendirme (Evaluation)



Verilen : $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0 x^0$

Amaç: evaluate $f(y)$

Incremental Approach

1. Sum = a_0
2. for i = 1 to n

   add $a_i y^i$ to the sum

Polynomial Evaluation (Polinom sonucu bulma)

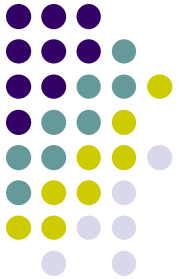
- **Analiz**
  - i^{th} adımda, çarpma ve 1 toplama işlemi

$$T = \sum_{i=1}^{n} (i+1) + 1 = O(n^2)$$

- **İyileştirme**
  - Önceki adımlardan $\mathbf{y^{i-1}}$ i biliyoruz.
  - 2 çarpım& 1 toplama

$$T(n) = 3n$$

Polynomial Evaluation



- **Daha da iyileştirme**

$$P_n(x) = x(a_n x^{n-1} + \ldots + a_1) + a_0$$

1 çarpma& 1 toplama

$$T(n) = 2n$$