

## Algoritma Analizi

BLM3021 Mine Elif KARSLIGIL & M. Arıcan GÜVENSAN

### Kaynaklar :

- 1) The Design & Analysis of Algorithms , A. Levitin
- 2) Algorithms : 4th Edition , R. Sedgewick , K. Wayne  
<http://www.algs4.princeton.edu>

3 ödev : %15

2 Vize : %45

Proje : %10

Final : %30

\* Programlama dili  $\Rightarrow$  C dili (ödevde göre değişebilir)

### Algoritma Yer Karmasıklığı

- $\alpha$  değişkenler
- $\alpha$  sabitler
- $\alpha$  program komutları
- $\alpha$  fonk. çağırma vb. sebeplerle yığın işlemleri
- $\alpha$  reküratif fonksiyonları yığını oluşturuyor.

### Algoritmaların Karşılaştırması

Dogruluğu?

Hızlı mı?

Az yer kaplıyor mu?

2

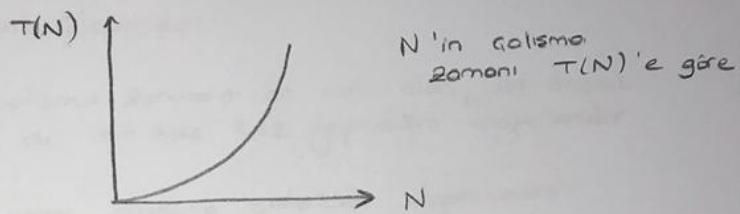
### Algoritma Çalışma Zamanının Deneysel Görümü

3-sum problemi : Verilen N farklı sayı içinden toplamı  $\phi$  eden üçlüerin bulunması

$a[i]$	$a[j]$	$a[k]$	sum
-30	-10	40	$\emptyset$
-30	-10	50	10
-10	40	50	80
-10	40	-30	$\emptyset$

```

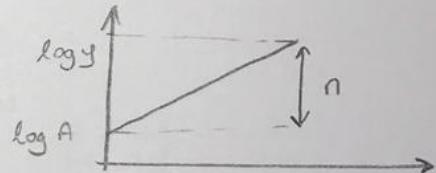
for(i=0 ; i < N ; i++)
  for(j=i+1 ; j < N ; j++)
    for(k=j+1 ; k < N ; k++)
      if( a[i] + a[j] + a[k] == 0 )
        Count++;
  
```



Log-log plot:  $N$ 'nin  $T(N)$ 'e göre logaritmik değişimini çizilir.

$y = A * x^n$  ise iki tarafın logaritmosu alınırsa

$$\begin{aligned}
 \log y &= \log(A * x^n) \\
 &= \log(A) + \log(x^n) \\
 &= \log(A) + n \cdot \log x
 \end{aligned}$$



### Deneyel Görümlerde Çalışma Zamanının Etkileyenler:

Donanım: CPU, bellek, cache

Yazılım: Derleyici

Sistem: OS, diğer uygulamalar vb.

) Sisteme bağlı

Bağımsız etkenler

✗ Algoritma

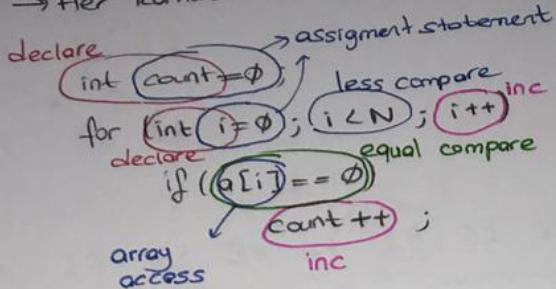
✗ Data (input)

! Deneyel Görümlerde her türlü donanım ve yazılım olasılığının aynı olması gereklidir.

### Calisma Zamaninin Matematik Modeli

\* Bütün komutların çalışma zamanlarının toplamı.

→ Her komutun çalışma zamanı aynı varsayılırsa



Her kodu tek tek toplamak gerekiyor.

### Calisma Zamaninin Teorik Ölçülmesi

Ana İstem: Algoritmanın çalışma zamanını en çok alan, en önemli işi belirle ve bu işin kaç kez yapıldığını değerlendir.

Asimptotik Analiz: Giriş verisi sonsuz giderken algoritmanın performans analizi nasıl yapılır?

Büyüme Derecesi: N büyürükçe algoritmanın çalışma hızı nasıl değişiyor?

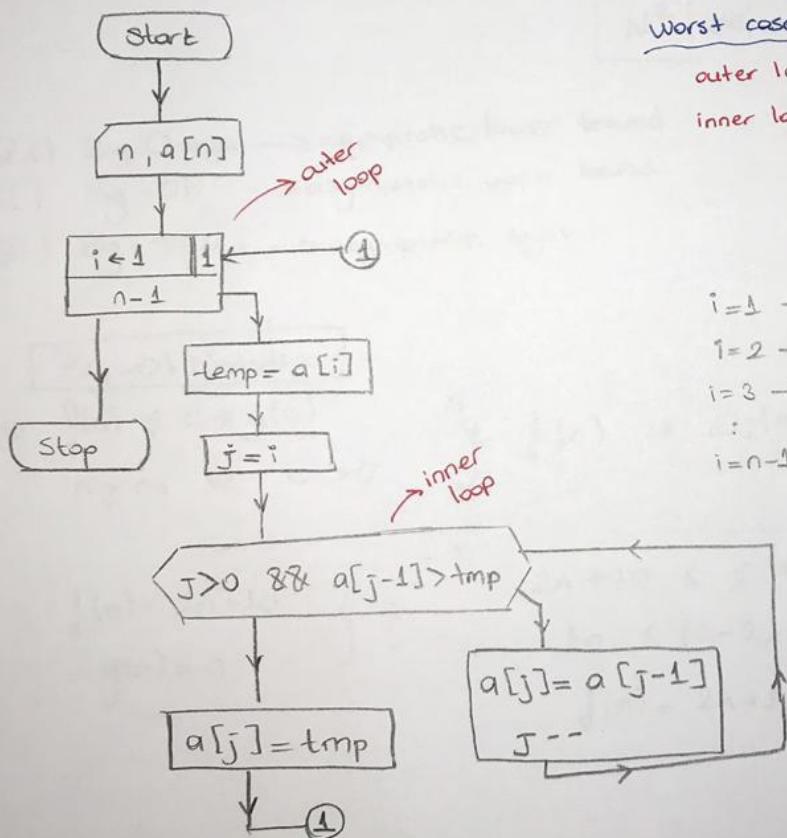
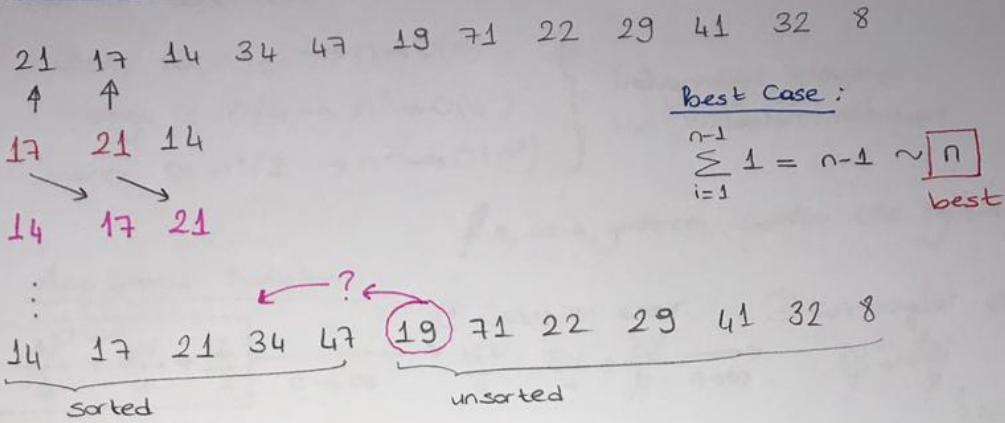
#### Hızı etkileyenler :

1) Girişteki eleman sayısı

2) Bu elemanların hangi düzende olduğu

! Bu etkenleri ölçmek için best, average ve worst senaryolarını hesaplamak gereklidir.

### Insertion Sort



### Worst case:

outer loop:  $i = 1 \dots (n-1)$

inner loop:  $\left. \begin{array}{l} j=1 \\ j=2 \\ j=3 \\ \vdots \\ j=n-1 \end{array} \right\} n-1$

$$i=1 \rightarrow j=1$$

$$i=2 \rightarrow j=0, 1$$

$$i=3 \rightarrow j=0, 1, 2$$

$$\vdots \quad \vdots$$

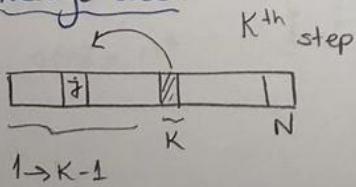
$$i=n-1 \rightarrow j=0, 1, 2, \dots, (n-2)$$

$$\sum_{i=1}^{n-1} \sum_{j=1}^{i-1} 1 = \sum_{i=1}^{n-1} i$$

$$= \frac{n(n-1)}{2} \sim \boxed{\frac{n^2}{2}}$$

worst

### Average Case:



K-j+1

$$\frac{1}{K} \cdot \sum_{j=1}^K (K-j+1) = \frac{1}{K} \left( K^2 - \frac{K(K-1)}{2} + K \right)$$

$$= \frac{2K^2 - K^2 - K + 2K}{2K} = \frac{K+1}{2}$$

$$\sum_{k=2}^N \frac{k+1}{2} = \frac{1}{2} \cdot \frac{(N+1)(N+2)}{2} = \boxed{\frac{1}{4} \cdot [N^2 + 3N + 2]}$$

Average

Insertion sort için

$$\left. \begin{array}{l} \text{Best} \approx n-1 \rightarrow n \rightarrow O(n) \\ \text{Average} \approx n^2/4 \rightarrow n^2 \rightarrow O(n^2) \\ \text{Worst} \approx n^2/2 \rightarrow n^2 \rightarrow O(n^2) \end{array} \right\} \begin{array}{l} \text{İndirgendigi göstermek} \\ \text{için kullanılan notasyon} \end{array}$$

$\forall n, \infty$  'a giderken sabitler bittiğinden dikkat.

### Asymptotic Notation

$$\boxed{\frac{N^2}{4} + \frac{3N}{4} + \frac{2}{4}} \xrightarrow{n \rightarrow \infty} \frac{N^2}{4} + \frac{3N}{4} + \frac{2}{4} \xrightarrow{n \rightarrow \infty} \frac{N^2}{4} + \frac{3N}{4}$$

$\downarrow n \rightarrow \infty$

$N^2$  ←

① Sabitler elenir.  
② Kat sayılar elenir.  
③ En yüksek dereceli terim alınır.  
 $N^2 + N$

$\Omega()$  Big-Omega → asymptotic lower bound

$O()$  Big-Oh → asymptotic upper bound

$\Theta()$  Big-Theta → asymptotic tight

### Big-Oh Notation

$\text{f}(n) \leq c * g(n)$   
\*  $n \geq n_0$  ve  $c > 0$

$f(n)$  is  $O(g(n))$

$$\left. \begin{array}{l} f(n) = 2n+10 \\ g(n) = n \end{array} \right\} ?$$

$$2n+10 \leq c * n$$

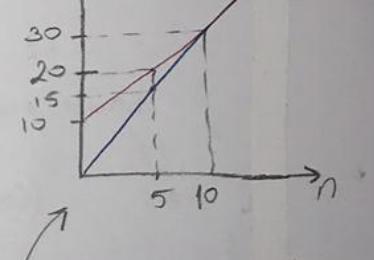
$$10 \leq (c-2) * n$$

$$f(n) = 2n+10 \rightarrow O(n)$$

$\uparrow$   
 $g(n)$

$$n \geq n_0$$

$$c=3 \quad n=10 \quad \checkmark$$



$$\left. \begin{array}{l} f(n) = 2n+10 \\ g(n) = n^2 \end{array} \right\} ?$$

$$2n+10 \leq c * n^2$$

$$cn^2 - 2n \geq 10$$

$$n \geq n_0 \quad c > 0$$

$$n=1 \quad c=100 \quad \checkmark$$

$$f(n) = 2n+10 \rightarrow O(n^2)$$

$\uparrow$   
 $g(n)$

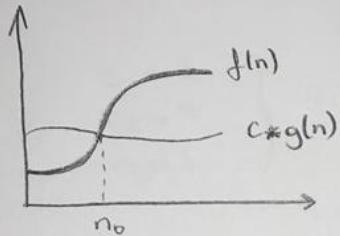
### Big-Omega Notation

$$\textcircled{2} \quad f(n) \in \Omega(g(n)) \quad \left\{ \begin{array}{l} \text{asymptotic lower bound} \\ \text{if } c > 0, n_0 > 0, n \geq n_0 \end{array} \right. \quad \left\{ \begin{array}{l} f(n) \geq c * g(n) \end{array} \right.$$

insertion sort  $\rightarrow$  best case:  $\Omega(n)$

sequential search  $\rightarrow$  best case:  $\Omega(1)$

$$f(n) = 5n^2 \rightarrow \Omega(n^2) \quad 5n^2 \geq c * n^2$$



$$c > 0 \quad n > n_0 \quad n_0 > 0$$

$$c = 1 \quad n_0 = 1 \quad \Rightarrow (5 - c)n^2 \geq 0 \quad \checkmark$$

$$f(n) = 5n^2 \rightarrow \Omega(n)$$

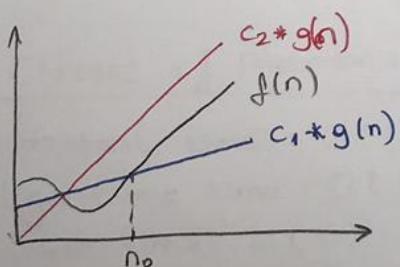
$$5n^2 \geq c * n \quad \Rightarrow n(5 - cn) \geq 0$$

$$c > 0 \quad n > n_0 \quad n_0 > 0$$

### Big-Theta Notation

$$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n) \quad \left\{ \begin{array}{l} f(n) = \Theta(g(n)) \\ c_1 \leq c_2, n_0 > 0, n > n_0 \end{array} \right.$$

tight bound



$$\left\{ \begin{array}{l} f(n) \in \Omega(g(n)) \\ \in O(g(n)) \end{array} \right\} \quad \Omega(g(n)) = O(g(n))$$

$f(n) = 2n + 5 \rightarrow \Theta(n^2)$  diyemem çünkü  $\Omega(n^2)$  degildir.

$\Omega(1)$  diyemem çünkü  $\Theta(1)$  degildir.

7

$$f(n) = 2n+5 \in ? \Theta(n)$$

$$\textcircled{1} \quad 2n+5 \in ? \sim_2(n) \rightarrow 2n+5 \geq c*n$$

$$5 \geq (c-2)n$$

$$n_0=1 \quad c=3 \quad \checkmark \checkmark$$

$$\textcircled{2} \quad 2n+5 \in ? O(n) \rightarrow 2n+5 \leq c*n$$

$$5 \leq \left(\frac{c}{2}-2\right)n$$

$$n_0=1 \quad c=\frac{7}{2} \quad \checkmark \checkmark$$

$$\left. \begin{array}{l} c_1=3 \\ c_2=7 \\ n_0=1 \end{array} \right\} c_1*n \leq 2n+5 \leq c_2*n \rightarrow f(n) \in \Theta(n)$$

Insertion Sort :

$$\begin{aligned} \text{worst case } &\sim n^2 \rightarrow \Theta(n^2) \\ \text{best case } &\sim n \rightarrow \Theta(n) \end{aligned}$$

insertion sort  
 $\Theta(n^2) \rightarrow \max n^2 \text{ olabilir}$   
 $\sim_2(n) \rightarrow \min n \text{ olabilir.}$

~~OK~~

Merge Sort :

$$\left. \begin{array}{l} \text{Best case } \sim n \log n \\ \text{Worst case } \sim n \log n \end{array} \right\} \Theta(n \log n)$$

Classes of Algorithms :

constant time :  $\Theta(1) \longrightarrow a=b+c$

logarithmic time :  $\Theta(\log n) \longrightarrow \text{binary search}$

linear time :  $\Theta(n) \longrightarrow \downarrow \text{Loop}$

linearthmic time :  $\Theta(n \log n) \rightarrow \text{merge sort}$

quadratic time :  $\Theta(n^2) \longrightarrow \text{double loop}$

cubic :  $\Theta(n^3) \longrightarrow \text{triple loop}$

exponentiel :  $\Theta(2^n) \longrightarrow \text{exhaustive search}$

## Useful Formulas for the Analysis of Algorithms

### Properties of Logarithms

1.  $\log x^y = y \cdot \log x$
2.  $\log xy = \log x + \log y$
3.  $\log \frac{x}{y} = \log x - \log y$
4.  $\log_a x = \log_b x \cdot \log_b a$
5.  $a^{\log_b x} = x^{\log_b a}$

### Combinatorics

1. Number of permutations of an  $n$ -element set:  
 $P(n) = n!$
2. Number of  $k$ -combinations of  $n$ -elements set:  
 $C(n, k) = \frac{n!}{k!(n-k)!}$
3. Number of subsets of an  $n$ -elements set:  $2^n$

### Important Summation Formulas

1.  $\sum_{i=l}^u 1 = \underbrace{1+1+\dots+1}_{u-l+1} = u-l+1 \quad (l \leq u)$
  2.  $\sum_{i=1}^n i = 1+2+\dots+n = \frac{n(n+1)}{2} \approx \frac{1}{2} n^2$
  3.  $\sum_{i=1}^n i^2 = 1^2+2^2+\dots+n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3} n^3$
  4.  $\sum_{i=1}^n i^k = 1^k+2^k+\dots+n^k \approx \frac{1}{k+1} n^{k+1}$
  5.  $\sum_{i=0}^n a^i = 1+a^1+a^2+\dots+a^n = \frac{a^{n+1}-1}{a-1} \quad (a \neq 1)$
  6.  $\sum_{i=1}^n i \cdot 2^i = 1*2 + 2*2^2 + \dots + n \cdot 2^n = (n-1)2^{n+1} + 2$
  7.  $\sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \approx \ln n + \gamma \quad (\gamma = 0.5772)$
  8.  $\sum_{i=1}^n \lg i \approx n \log n$
- $$\sum_{i=0}^n 2^i = 2^{n+1} - 1$$

### Sum Manipulation Rules

1.  $\sum_{i=l}^u c \cdot a_i = c \cdot \sum_{i=l}^u a_i$
2.  $\sum_{i=l}^u (a_i + b_i) = \sum_{i=l}^u a_i + \sum_{i=l}^u b_i$
3.  $\sum_{i=l}^u a_i = \sum_{i=l}^m a_i + \sum_{i=m+1}^u a_i$  where ( $l \leq m < u$ )

### Floor and Ceiling Formulas

$$\lfloor 3.8 \rfloor = 3$$

$$\lfloor -3.8 \rfloor = -4$$

$$\lceil 3.8 \rceil = 4$$

$$\lceil -3.8 \rceil = -3$$

2.  $\lfloor x+n \rfloor = \lfloor x \rfloor + n$  and  $\lceil x+n \rceil = \lceil x \rceil + n$   $\rightarrow$  for real  $x$  and integer  $n$

$$3. \lfloor n/2 \rfloor + \lceil n/2 \rceil = n$$

$$4. \lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$$

### Mathematical Analysis of Non-Recursive Algorithms

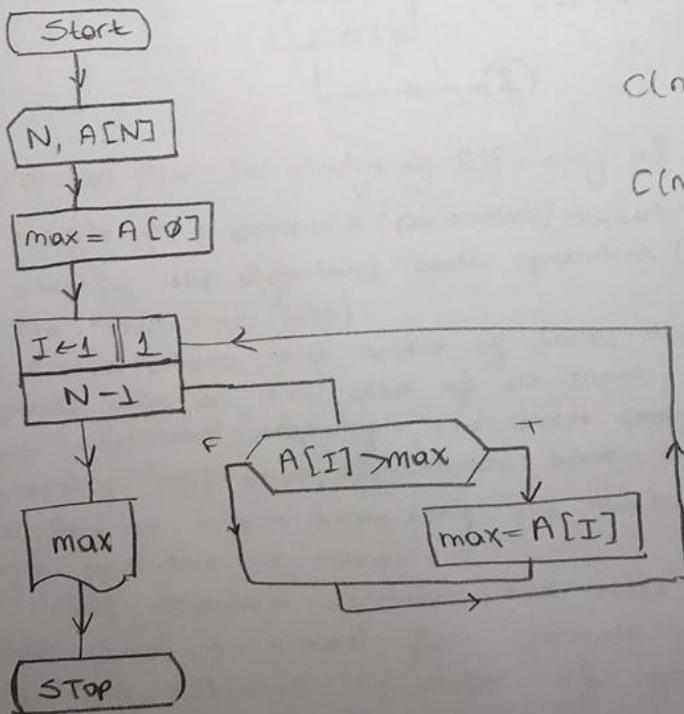
Two operations:

$A[I] > max$   
Basic operation  
 $max = A[I]$

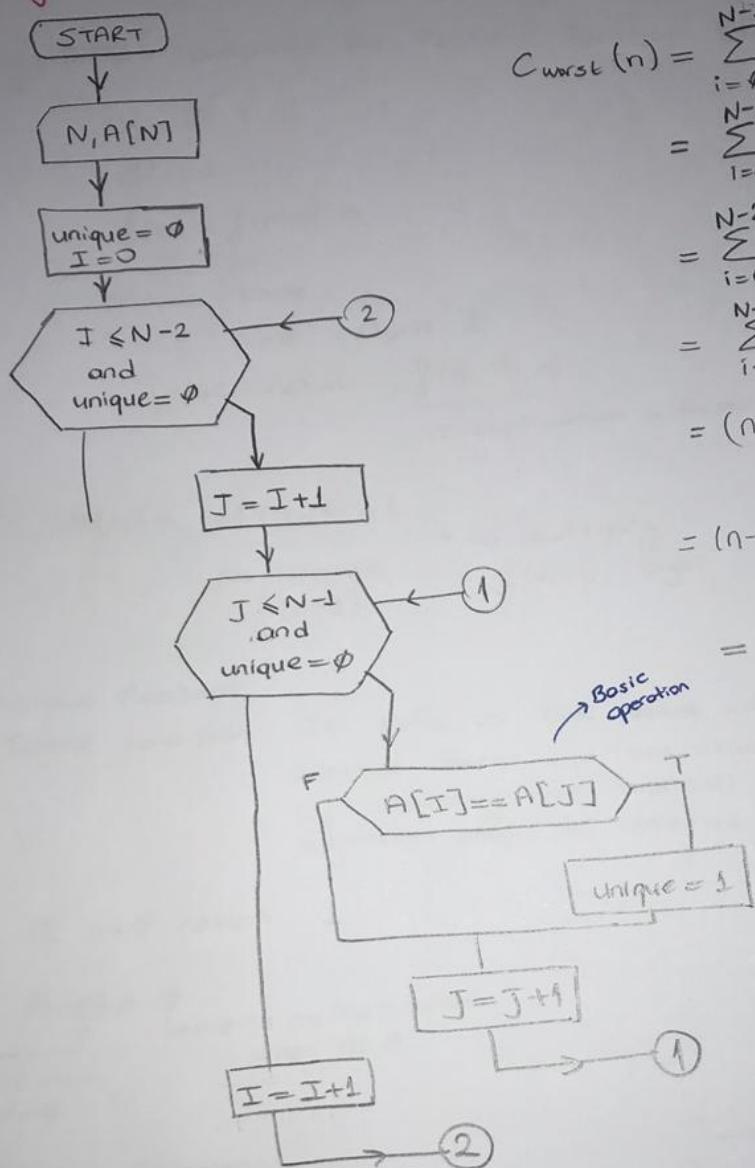
$C(n)$ : the number of times this comparison is executed

$$C(n) = \sum_{i=1}^{n-1} 1 = n-1 \approx n$$

$\Theta(n)$



## Algorithms Element Unique Problem



$$\begin{aligned}
 C_{\text{worst}}(n) &= \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} 1 \\
 &= \sum_{i=0}^{N-2} ((n-1) - (i+1) + 1) \\
 &= \sum_{i=0}^{N-2} (n-i-1) \\
 &= \sum_{i=0}^{N-2} (n-1) - \sum_{i=0}^{N-2} i \\
 &= (n-1) \cdot \sum_{i=0}^{N-2} i - \frac{(N-2)(N-1)}{2} \\
 &= (n-1)^2 - \frac{(n-2)(n-1)}{2} \\
 &= \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2)
 \end{aligned}$$

## General Plan for Analyzing Efficiency of Non Recursive Algorithms

1. Decide on a parameter (parameters) indicating input's size
2. Identify the algorithm's basic operation: (As a rule, it is located in its inner-most loop)
3. Check whether the number of times the basic operation is executed depends only on the size of an input. If it also depends on some additional property, the worst case, average-case and if necessary best-case, efficiencies have to be investigated separately.
4. Set up a sum expressing the number of times the algorithm's basic operation is executed.
5. Using standard formulas and rules of sum manipulation, either find a closed-form formula for the count or, at the very least, establish its order of growth.

## Mathematical Analysis of Recursive Algorithms

Algorithms: computing the Factorial Function  $F(n) = n!$

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

$$\emptyset! = 1$$

$$f(n) = f(n-1) \cdot n$$

pseudo-code:

if  $n = \emptyset$  return 1

else return  $f(n-1)$

$\underbrace{f(n-1) \cdot n}_{\text{multiplication}} = \text{Basic operation!}$

$$M(n) = \underbrace{M(n-1) + 1}_{\text{to compute } f(n-1)} \xrightarrow{\text{to multiply } f(n-1) \text{ by } n}$$

### Recursive Relations

Initial condition: It tells us the value with which the sequence starts. We can obtain this value by inspecting the condition that makes the algorithm stop its recursive calls.

if  $n = \emptyset$  return 1

$$M(\emptyset) = \emptyset \xrightarrow{\text{no multiplication when } n = \emptyset}$$

the calls stop when  $n = \emptyset$

### Recurrence Relations

$$M(n) = M(n-1) + 1 \quad \text{for } n > \emptyset$$

$$M(\emptyset) = \emptyset$$

### Method of Backward Substitution

$$\begin{aligned}
 M(n) &= M(n-1) + 1 \quad \text{substitute} \quad M(n-1) = M(n-2) + 1 \\
 &= [M(n-2) + 1] + 1 \quad \text{substitute} \quad M(n-2) = M(n-3) + 1 \\
 &= [M(n-3) + 1] + 2 = M(n-3) + 3 \\
 &\vdots \\
 M(n) &= M(n-i) + i
 \end{aligned}$$

To take advantage of initial condition,  
 $M(n) = M(n-1) + 1 = M(n-i) + i = \dots = M(n-n) + n = n$

$$\boxed{n=\emptyset, i=n \\ M(\emptyset) + \emptyset = \emptyset}$$

### A General Plan for Analyzing Efficiency of Recursive Algorithms

İlk 3 modde recursive fonksiyonlar ile aynı

4. Set up a recurrence relation with an appropriate initial condition, for the number of times the basic operation is executed.
5. Solve the recurrence or at least ascertain the order of growth of its solution

Ex/

Input: A positive decimal integer  
 The number of binary digits in binary's representation

if  $n=1$  return 1  
 else return BinaryDigit  $\lfloor n/2 \rfloor + 1$

A(n): The number of Additions

$$\boxed{A(n) = A(\lfloor n/2 \rfloor) + 1}$$

Initial Condition:

$$n=1 \quad A(1) = \emptyset$$

$$\overline{\overline{\overline{\overline{n=2^k \rightarrow A(2^k) = A(2^{k-1}) + 1}}}} \quad A(2^0) = \emptyset$$

Backward Substitution  
 $\Rightarrow$  öztümü

Backward substitution

$$A(2^k) = A(2^{k-1}) + 1 \quad \text{substitute } A(2^{k-1}) = A(2^{k-2}) + 1$$

$$A(2^k) = [A(2^{k-2}) + 1] + 1 \quad \begin{matrix} \text{substitute } A(2^{k-2}) \\ \rightarrow A(2^{k-2}) + 2 \end{matrix} = A(2^{k-3}) + 1$$

$$A(2^k) = [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3$$

⋮

⋮

$$= A(2^{k-i}) + i$$

⋮

$$= A(2^{k-k}) + k$$

$$= A(2^0) + k = k$$

$$\underbrace{A(1)}_{\phi} = \phi$$

$$n = 2^k \Rightarrow k = \log_2 n$$

$$A(n) = \log_2 n \in \Theta(\log n)$$

### Binary Search Algorithm

Input : A [0 ..... n-1] sorted array

Output : Index or -1

Pseudo Code :

$l \leftarrow \phi ; r \leftarrow n-1$

while ( $l \leq r$ ) do

$m \leftarrow \lfloor (l+r)/2 \rfloor$

if  $K = A[m]$  return  $m$

else if  $K < A[m]$   $r \leftarrow m-1$

else  $l \leftarrow m+1$

return -1

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value	3	14	27	31	39	42	55	74	76	81	85	93	98
Iteration 1	l					m						r	
Iteration 2							l			m			r
Iteration 3								l	r				✓

C = compare  
w = worst case

$$C_w(n) = C_w(\lfloor n/2 \rfloor) + 1, \text{ for } n > 1$$

Initial Condition:

$$C_w(1) = 1$$

$n = 2^k \rightarrow$  problem 2'ye bölünderek devam ettiği için.

Backward Substitution:

$$C_w(2^k) = k+1 = \log_2 n + 1 \quad (\text{Bir önceki soyfoda aynı var})$$

Verify by Substitution that function  $C_w(n) = \lfloor \log_2 n \rfloor + 1$  indeed satisfies equation ---- for any positive even number even numbers

$$n = 2i, i > 0$$

$$\begin{aligned} C_w(n) &= \lfloor \log_2 n \rfloor + 1 = \lfloor \log_2 2i \rfloor + 1 = \lfloor \log_2 2 + \log_2 i \rfloor + 1 \\ &= \lfloor 1 + \log_2 i \rfloor + 1 = \log_2 i + 2 \end{aligned}$$

$$C_w(\lfloor n/2 \rfloor) + 1 = C_w(\lfloor 2i/2 \rfloor) + 1 = C_w(i) + 1$$

$$= (\lfloor \log_2 i \rfloor + 1) + 1 = \lfloor \log_2 i \rfloor + 2$$

The worst-case efficiency of Binary Search:  $\Theta(\log n)$

$$C_{\text{avg}}(n) \approx \log_2 n \rightarrow C_{\text{avg}}^{\text{yes}} \approx \log_2 n - 1$$

$$C_{\text{avg}}^{\text{no}} \approx \log_2(n+1)$$

$$\lfloor \log_2 10^3 \rfloor + 1 = 10$$

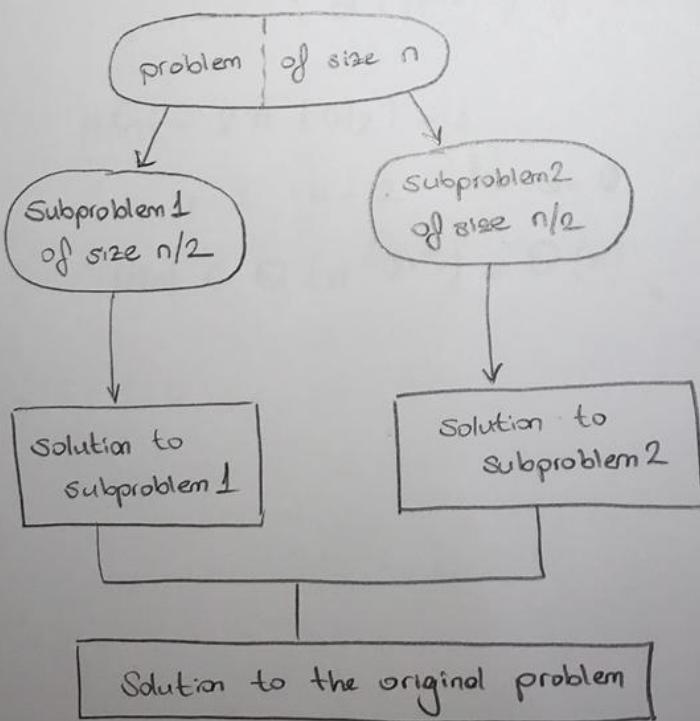
$$\lfloor \log_2 10^6 \rfloor + 1 = 20$$

Binary search is  
recursive Gözemeyle  
geliş

### Divide and Conquer Algorithms

They work according to the following general plan.

1. A problem's instance is divided into several smaller instances of the same problem, ideally about the same size.
2. The smaller instances are solved (typically recursively though sometimes a different algorithm is employed when instances become small enough).
3. If necessary, the solutions obtained for the smaller instances are combined to get a solution to the original problem.



İsimde yaramayacak  
ise olt problemlere  
ayırmak maliyeti  
ortaktır.

Computing the sum of  $n$

$$a_0 + a_1 + a_2 + \dots + a_{n-1} = (a_0 + a_1 + \dots + a_{\lfloor n/2 \rfloor - 1}) + (a_{\lfloor n/2 \rfloor} + \dots + a_{n-1})$$

instance size :  $n$

subproblems :  $b$

the subproblems that need to be solved :  $a$

$$T(n) = a T(n/a) + f(n) \Rightarrow \text{Running Time} \quad \begin{bmatrix} \text{General Divide and} \\ \text{Conquer Recurrence} \end{bmatrix}$$

$\hookrightarrow f(n)$  is a function that accounts for the time spent on dividing the problem into smaller ones and combining their solutions.

### Master Theorem

if  $f(n) \in \Theta(n^d)$  where  $d \geq 0$

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

$$A(n) = 2A(n/2) + 1$$

$$a=2 \quad b=2 \quad \hookrightarrow d=0$$

$$A(n) \in \Theta(n^{\log_2 2}) = \Theta(n)$$

## Merge Sort Algorithm

Pseudo Code:  $A[\emptyset \dots n-1]$

if ( $n > 1$ )

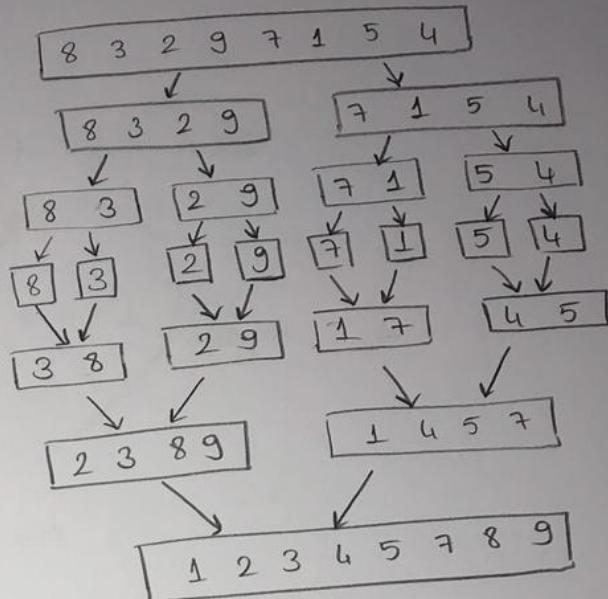
copy  $A[\emptyset \dots \lfloor n/2 \rfloor - 1]$  to  $B[\emptyset \dots \lfloor n/2 \rfloor - 1]$

copy  $A[\lfloor n/2 \rfloor \dots n-1]$  to  $C[\emptyset \dots \lfloor n/2 \rfloor - 1]$

mergeSort ( $B[\emptyset \dots \lfloor n/2 \rfloor - 1]$ )

mergeSort ( $C[\emptyset \dots \lfloor n/2 \rfloor - 1]$ )

merge ( $B, C, A$ )



$$C(n) = 2C(n/2) + C_{\text{merge}}(n) \quad n > 1$$

Initial Condition :  $C(1) = \emptyset$

## Partition Algorithm:

Merge ( $B[\emptyset \dots p-1], C[\emptyset \dots q-1], A[\emptyset \dots p+q-1]$ )

$i \leftarrow \emptyset; j \leftarrow \emptyset; k \leftarrow \emptyset$

while  $i < p$  and  $j < q$  do

| if  $B[i] \leq C[j]$

| |  $A[k] \leftarrow B[i]; i \leftarrow i+1$

| else

| |  $A[k] \leftarrow C[j]; j \leftarrow j+1$

| |  $k \leftarrow k+1$

| if  $i = p$

| | copy  $C[j \dots q-1]$  to  $A[k \dots p+q-1]$

else

| | copy  $B[i \dots p-1]$  to  $A[k \dots p+q-1]$

Recurrence Relation

$$C(n) = 2 C(n/2) + C_{\text{merge}}(n) \quad \text{for } n > 1$$

Initial Condition :  $C(1) = \emptyset$

$$C_{\text{worst}}(n) = 2 C_{\text{worst}}(n/2) + n - 1$$

Backward Substitutions

$$C(n/2) = 2 C(n/4) + n/2$$

$$C_{\text{worst}}(n) = 2 [2 C(n/4) + n/2] + n$$

$$= 4 C(n/4) + 2n$$

$$= 8 C(n/8) + 3n \quad n = 2^i \rightarrow i = \log_2 n$$

$$= 2^i C(n/2^i) + i * n$$

$\downarrow$  initial condition's boğlu şartı

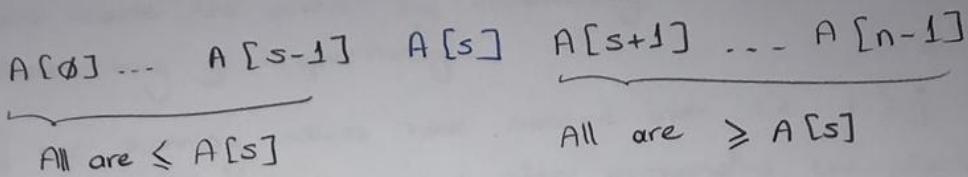
$$= n * \emptyset + \log_2 n \approx n$$

$$C_{\text{worst}}(n) \in \Theta(n \log_2 n)$$

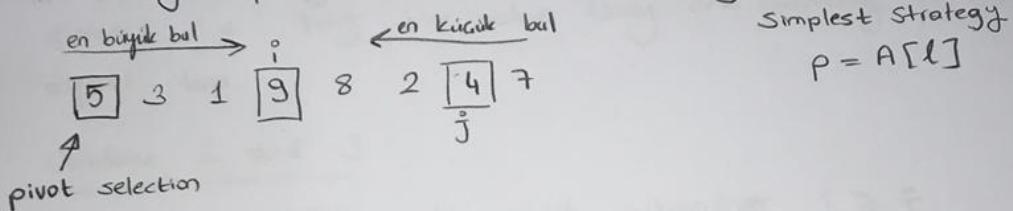
! Principal Shortcoming : Linear amount of extra storage

### Quick Sort Algorithm

Quick Sort divides the input's elements according to their value.



Obviously, after a partition has been achieved,  $A[s]$  will be in its final position in the sorted array.



#### pseudo Code :

```

if  $l < r$ 
   $s \leftarrow \text{Partition}(A[l \dots r])$ 
  QuickSort( $A[l \dots s-1]$ )
  QuickSort( $A[s+1 \dots r]$ )
  
```

① Stops on encountering the first element greater than or equal to the pivot : left-to-right scan

② Stops on encountering the first element smaller than or equal to the pivot : right-to-left scan

- ① If scanning indices  $i$  and  $j$  have not crossed i.e.  $i < j$  we simply exchange  $A[i]$  and  $A[j]$  and resume the scans by incrementing  $i$  and decrementing  $j$  respectively.
- ② If the scanning indices have crossed over i.e.  $i > j$  we have partitioned the array after exchanging the pivot with  $A[j]$
- ③ If the scanning indices stop while pointing to the same element i.e.  $i = j$  the value they are pointing to must be equal to  $p$ .

Combine 2 and 3

Exchange  $A[j]$  with pivot whenever  $i \geq j$

pivot	5	3	1	$\boxed{9}$	8	2	$\boxed{4}$	7
	5	3	1	4	$\boxed{8}$	$\boxed{2}$	9	7
	5	$\frac{3}{\boxed{2}}$	$\frac{1}{\boxed{3}}$	4	$\frac{8}{2}$	$\frac{2}{8}$	9	7
yeni pivot	2	$\boxed{3}$	$\boxed{1}$	4	5	8	9	7
	2	$\frac{1}{\boxed{2}}$	$\frac{3}{\boxed{1}}$	4	5	8	9	7
$j < i$ oldugunda pivot $j$ ile yer degistirir.	{ 1	2	3	4	5	8	9	7

Partition Algorithm:

$p \leftarrow A[l]$

$i \leftarrow l ; j \leftarrow r+1$

repeat

repeat  $i \leftarrow i+1$  until  $A[i] \geq p$

repeat  $j \leftarrow j+1$  until  $A[j] \leq p$

swap ( $A[i], A[j]$ ) ;

until  $i \geq j$

swap ( $A[i], A[j]$ ) // undo last swap when  $i \geq j$

swap ( $A[l], A[j]$ )

return  $j$

Best Case

If all the splits happen in the middle of corresponding subarrays, we will have the best case.

## Recurrence Relation

$$C_{\text{Best}}(n) = 2C_{\text{Best}}(n/2) + n, n > 1$$

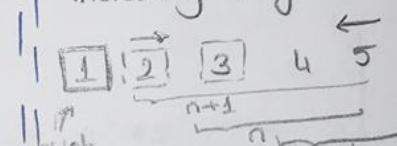
$$C_{\text{Best}}(1) = \emptyset$$

$$C_{\text{Best}}(n) \in \Theta(n \log n)$$

Analiz yapmadık. Merge sort'a yorumistik Backward substitution yapılarak çözüller.

Worst Case

## Increasing Array



$A[0 \dots n-1]$   
Already sorted

$$\begin{aligned} C_{\text{Worst}}(n) &= (n+1) + n + \dots + 3 \\ &= \frac{(n+1)(n+2)}{2} - 3 \end{aligned}$$

$$C_{\text{Worst}}(n) \in \Theta(n^2)$$

Average Case

$$C_{\text{Avg}}(n) \approx 2n \ln n \approx 1.38 n \log n$$

Sequential Search:

0	1	2	3	4
3	5	1	4	2

$O(n)$

Binary Search:

0	1	2	3	4	5	6
2	7	9	12	17	19	22

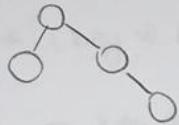
$\leftarrow \uparrow$

$x=7$

$O(\log_2 N)$

Binary Search Tree:

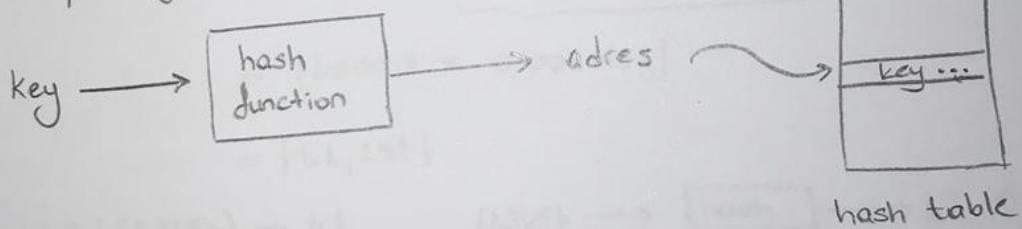
$O(n)$



denge bozulmuş olabilir.

## HASHING

Unique key kullanılır.



① Hesabı kolay.

② Tutarlı olmalı.

③ Adresin olabildiğince uniform dağılması sağlanmalı

key mod m  
table uzunluğu

$$m=10 \rightarrow 2^d$$

$$\text{key} \rightarrow 100, 10, 40, 50 \rightarrow \emptyset$$

### En çok kullanılan Hash Fonksiyonları

#### ① Division Method

$$\text{hash(key)} = \text{key} \bmod m$$

$\uparrow$   
tablo uzunluğu

$$m=11$$

$$\text{key}=25$$

$$\text{hash}(25)=3$$

0
1
2
3
25...
!
10

#### ② Multiplication Method

$$A: \text{constant} \quad 0 < A < 1 \quad A = \frac{\sqrt{5} - 1}{2} = 0.618033$$

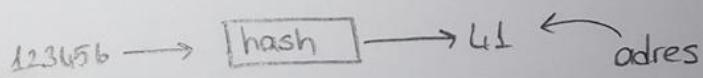
$$\text{hash(key)} = \lfloor m * [(\text{key} * A) \bmod 1] \rfloor$$

$$(\text{key} * A) \bmod 1 = \text{key} * A - \lfloor \text{key} * A \rfloor$$

$$m = 10000 \quad k = 123456$$

$$\begin{aligned} \text{hash}(123456) &= \lfloor 10000 * (\underbrace{123456 * 0,618033}_{76300,00461151}) \bmod 1 \rfloor \\ &= \lfloor 10000 * 0,00461151 \rfloor \\ &= \lfloor 41,151 \rfloor \end{aligned}$$

$$\text{hash}(123456) = 41$$



#### Birthday Paradox

$$365 \rightarrow \frac{365-1}{365} \Rightarrow \text{iki kişinin doğum günlerin aynı olma olasılığı}$$

$$365 \rightarrow \frac{365-1}{365} * \frac{365-2}{365} \Rightarrow \text{üç kişinin doğum günlerin aynı olma olasılığı}$$

$$365 \rightarrow \frac{365-1}{365} * \frac{365-2}{365} * \dots * \frac{365-k}{365} \Rightarrow k \text{ kişinin}$$

Hash map' te

table size :  $m$

$$\frac{m-1}{m} * \frac{m-2}{m} * \dots * \frac{m-(k-1)}{m} \sim e^{-\frac{k(k-1)}{m}}$$

$k$  sayının  
çakışma  
olasılığı

$$1 - e^{-\frac{k(k-1)}{m}} \rightarrow k \text{ sayının çakışma olasılığı}$$

Hash collision probability

Ex : 32 bit hash values

$$M = 2^{32} \rightarrow \text{after } 250\,000 \text{ insert \% 100}$$

$$77163 \text{ insert \% 50}$$

### Key

- ① Integer
- ② Floating number
- ③ String

"ELMA"      "MALE"  
 Aynı sayıyi içermemeli  
 Bu uygulanacak iken  
 Horner methodu kullanılır.

Horner's Method :

$$\text{hash(key)} = R * \underbrace{\text{str}[0] + \dots + R^0 * \text{str}[L-1]}_{\substack{\text{String'in} \\ \text{sayı karşılığı} \\ \text{bulunması}}} \quad L \Rightarrow \text{length of string}$$

$\text{str}[0] - 'A' + 1$

④ Compound Keys: 31.08.1997

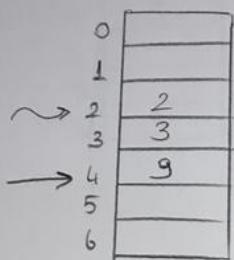
$$\downarrow \quad 31 * R + 8 * R^2 + 1997 * R^3$$

### Collision Problemı Çözümü

① Open Addressing:  $\rightarrow$  Table size  $m \sim 2 * N$

#### 1.1 Linear Probing:

$$\text{hash}(\text{key}, i) = [\text{h}'(\text{key}) + i] \bmod m$$

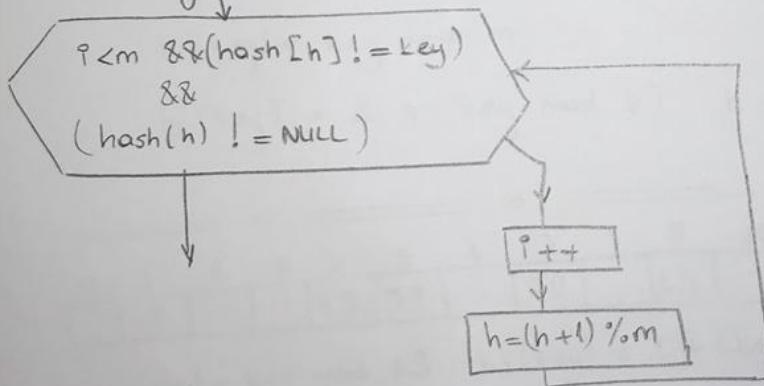


$$\begin{aligned} M &= 7 & i &= \emptyset, \dots, (m-1) \\ \text{key} &= 9 & i &= 0 \\ \text{hash}(9, \emptyset) &= 9 \bmod 7 = 2 \\ \text{hash}(9, 1) &= (9+1) \bmod 7 = 3 \\ \text{hash}(9, 2) &= (9+2) \bmod 7 = 4 \end{aligned}$$

Yerlestirilmek istenen yer doluyaşa boş yer arama sistemidir.

### Search

$$h = \text{key} \bmod m$$



### Delete

- ① Silinen eleman yerine bir flag konulmalıdır. Baska bir eleman orarken  $\text{hash}(h) \neq \text{NULL}$  koşulunu sağlamanın için bu yapılmalıdır.
- ② Silinen elemanın yerini doldurmak için modu aynı olan satırlar kaydedilebilir.

### (1.2) Quadratic Probing

$$h(k, i) = (h'(k) + i^2) \bmod m$$

0
1
2
3
4
5
6
9

$$\text{hash}(9, \emptyset) = 9 \bmod 7 = 2$$

$$\text{hash}(9, 1) = (2+1) \bmod 7 = 3$$

$$\text{hash}(9, 2) = (2+2^2) \bmod 7 = 6 \bmod 7 = 6$$

$$h(k, i) = (h'(k) + c_1 * i + c_2 * i^2)$$

$$c_1 = 2 \quad c_2 = 1$$

$$i=1$$

$$\text{hash}(9, 1) = (2 + \underset{c_1=2}{\cancel{2*1}} + \underset{i=1}{\cancel{1*1}}) \bmod m = 5$$

$$c_1=2 \quad i=1 \quad c_2=1$$

### (1.3) Double Hashing

$$\text{hash}(\text{key}, i) = [h_1(\text{key}) + \underbrace{i * h_2(\text{key})}_{\text{adın büyüklüğü}}] \bmod m$$

$$h_1(\text{key}) = \text{key} \bmod m$$

$$h_2(\text{key}) = 1 + (\text{key} \bmod k) \quad k < m$$

0	1	2	3	4	5	6	7	8	9	10	11	12
+9			69	58		72		14				

$$h_1(\text{key}) = \text{key} \bmod 13 \quad h_2(\text{key}) = 1 + (\text{key} \bmod 11)$$

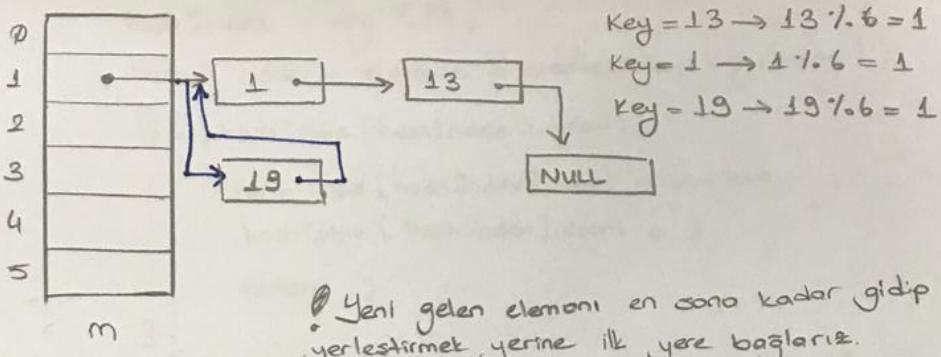
$$\text{hash}(14) = ?$$

$$\text{hash}(14, \emptyset) = h_1(14) = 14 \bmod 13 = 1 \quad \text{dolu}$$

$$i=1 \Rightarrow \text{hash}(14, 1) = 1 + [1 + 14 \bmod 11] * 1 = 1 + 4 = 5 \quad \text{dolu}$$

$$i=2 \Rightarrow \text{hash}(14, 2) = 1 + \underbrace{[1 + 14 \bmod 11]}_4 * 2 = 9 \quad \text{bos} \quad \checkmark$$

## (2) Separate Chaining



$m < N$

maximum  $M = N$

$m$  too small  $\rightarrow$  chains too long

$m \sim N/5$  olursa chain'in çok uzadığı hesaplanır.

```
struct node {
    int key;
    char name[40];
    struct node *next;
};

struct hash {
    struct node *head;
    int count;
};
```

```
struct hash *hashTable = NULL;
==

int main() {
    int key;
    char name[40];
    // okuma islemleri
    hashTable = (struct hash *) malloc(M, sizeof(struct hash));
    if (hashTable == NULL) {
        printf("Yer acilamadi"); exit(0);
    }
    insertToHash(key, name);
    searchInHash(key);
    deleteFromHash(key);
}
```

```

void insertToHash (int key , char name []) {
    int hashIndex ;
    hashIndex = key % M ;
    struct node * newNode = createNode (key , name) ;
    if (!hashTable [hashIndex].head) {
        hashTable [hashIndex].head = newNode ;
        hashTable [hashIndex].count = 1 ;
        return ;
    }
    newNode -> next = (hashTable [hashIndex].head) ;
    hashTable [hashIndex].head = newNode ;
    hashTable [hashIndex].count += 1 ;
    return ;
}

```

```

struct node *createNode (int key , char * name) {
    struct node * newNode ;
    newNode = (struct node *) malloc (sizeof (struct node)) ;
    newNode -> key = key ;
    // newNode -> age = age ;
    strcpy (newNode -> name , name) ;
    newNode -> next = NULL ;
    return newNode ;
}

```

```

void searchInHash (int key) {
    int hashIndex = key % m ;
    int flag = 0 ;
    struct node * myNode ;
    myNode = hashTable [hashIndex].head ;
    if (!myNode) {
        printf ("search element unavailable\n") ;
        return ;
    }
    while (myNode != NULL && !flag) {
        if (myNode -> key == key) {
            printf ("Element found\n") ;
            flag = 1 ;
        }
        myNode = myNode -> next ;
    }
}

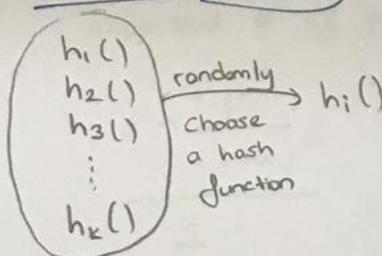
```

→ if (!flag) {  
    printf ("Element yok");  
}  
    return ;  
}

```
Void deleteFromHash (int key) {
    int hashIndex = key % M;
    int flag = 0; struct node *tmp, *myNode;
    myNode = (hashTable[hashIndex].head);
    if (!myNode) {
        printf ("Element yet");
        return;
    }
    tmp = myNode;
    while (myNode != NULL) {
        if (myNode->key == key) {
            flag = 1;
            if (myNode == hashTable[hashIndex].head) {
                hashTable[hashIndex].head = myNode->next;
            } else {
                tmp->next = myNode->next;
            }
            hashTable[hashIndex].count--;
            free (myNode);
        }
        tmp = myNode;
        myNode = myNode->next;
    }
    if (flag) {
        printf ("Deleted");
    } else {
        printf ("Not deleted");
    }
    return;
}
```

30

### ③ Universal Hashing



$$h_{a,b}(\text{key}) = [(a*k+b) \bmod p] \bmod m$$

$a, b, p \rightarrow \text{chosen randomly}$

$$p=17, m=6, a=3, b=4$$

$\hookrightarrow$  table size is not random

$$h(8) = ((3*8+4) \bmod 17) \bmod 6 = 5$$

Open addressing :  $M > N$

Separate chaining :  $M < N$

$$\alpha = \frac{N}{M} \quad \text{load factor}$$

$\alpha \Rightarrow$  genellikle 0.5 olur.  
Yer ve zaman dengesi  
sağlanır.

$\alpha$	# of probe (average)
0.1	1.11
0.2	1.23
0.3	1.52
0.4	1.89
0.5	2.5
0.6	3.62
0.8	13.00
0.9	50.5

### Unsuccessful Search

$$p(\text{first location is occupied}) = \alpha$$

$$p(\text{empty cell}) = (1-\alpha)$$

$$p(\text{first two is occupied}) = \alpha * \alpha$$

$$p(\text{probe terminates 2 steps}) = \alpha * (1-\alpha)$$

↑                      ↑  
1st step          2nd step

$$p(\text{probe terminates } k \text{ steps}) = \alpha^{k-1} * (1-\alpha)$$

Average # of steps :

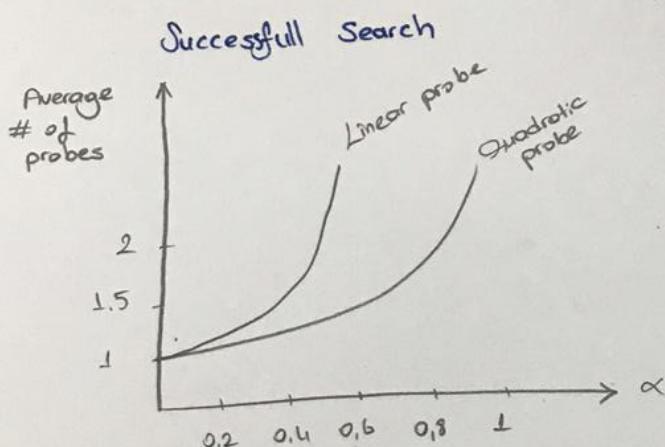
$$\sum_{k=1}^m k * \alpha^{k-1} * (1-\alpha) \Rightarrow \text{geometric series} = \frac{1}{1-\alpha}$$

31

insertion :

expected # of probes

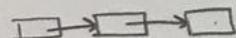
$$\frac{1 + \frac{1}{(1-\alpha)^2}}{2} \quad \alpha = 0.5 \rightarrow 2.5$$



Separate Chaining

① Unsuccessful Search

$$\alpha = \frac{N}{M}$$



keys = 64, 100, 128, 200, 300, 400, 500

tableSize = 8

hash(key) = x mod 8

64, 128, 200, 400 → 0

100, 300, 500 → 4

gcd (64, 100, 128, 200, 300, 400, 500, 8)

worst case : Bütün elementler aynı adreste

average case :  $\alpha = \frac{N}{M}$

expected length of a probe :  $\alpha = \frac{N}{M}$

$\mathcal{O}(1 + \alpha)$

- ↳ search
- ↳ hash function

$\alpha = \frac{N}{m}$

↳ slot

$\mathcal{O}(1 + \alpha) = \mathcal{O}(1)$

## Dynamic Programming

→ Planning Richard Bellman (1950's)

### Fibonacci Numbers

0 1 1 2 3 5

→  $n=0$  :  $\text{fibonacci}(n) \rightarrow 0$   
 $n=1$  :  $\text{fibonacci}(n) \rightarrow 1$   
 $n > 1$  :  $\text{fibonacci}(n-1) + \text{fibonacci}(n-2)$

```
int fibonacci(int n) {
    if (n==0)
        return 0;
    if (n==1)
        return 1;
    return fibonacci(n-1)+fibonacci(n-2);
}
```

Recurrence Relation:  $\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$

0	1	2	3	4	5	6
0	1	1	2	3	5	8

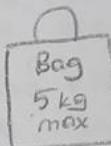
fibonacci

### 0-1 Knapsack Problem

Hırsız, yüze hafif pochada  
ağır seyleri almaya çalışıyor.

Brute Force:  $2^n$

item	weight	price
1	2	10
2	2	20
3	3	15
4	4	50



$W = 5 \text{ kg max}$

item	weight	value
1	2	12
2	1	10
3	3	20
4	2	15

		Weight					
		0	1	2	3	4	5
item	0	∅	∅	∅	∅	∅	∅
	1	∅	∅	12	12	12	12
2	∅	10	12	22	22	22	
3	∅	10	12	22	30	32	
4	∅	10	15	25	30	37	

max kozane

Amacim 2:

maximize  $\sum V_i$  subject to  $W$

$P(i, j)$ : maximum profit  
from item 1 to  $i$   
from weight 1 to  $j$

case 1 : thief takes item  $i$

$$P(i, w) = P(i-1, w-w_i) + V_i$$

$$i=3 \quad P(3, 3) = P(2, 1) + V_3$$

$$w_i=2 \quad \text{esya} \quad \downarrow \quad \downarrow \quad w_3=2$$

3kg iyo  
zador  
kozane

table  
ile okar  
yok

Case 2 : NOT takes item  $i$

$$P(i, w) = P(i-1, w)$$

$$\begin{aligned} w_1=2 \quad P(1, 1) &= \emptyset \quad 2 > 1 \\ P(1, 2) &= \max \{ 12 + 0, \emptyset \} = 12 \\ P(1, 3) &= \max \{ P(0, 1) + 12, P(0, 3) \} \\ w_2=1 \quad P(2, 1) &= \max \{ P(1, 0) + 10, P(1, 1) \} \\ P(2, 2) &= \max \{ P(1, 2), P(1, 1) + 10 \} \\ P(2, 3) &= \max \{ P(1, 2) + 10, P(1, 3) \} \\ P(3, 3) &= \max \{ P(2, 0) + 20, P(2, 3) \} \\ P(4, 3) &= \max \{ P(2, 1) + 20, P(2, 4) \} \end{aligned}$$

$$P[i, w] = \begin{cases} P[i-1, w] & \text{if } w_i > w \\ \max \left\{ \begin{array}{l} V_i + P[i-1, w-w_i] \\ P[i-1, w] \end{array} \right\} & \text{others} \end{cases}$$

Initialization

$$P[\emptyset, w] = \emptyset$$

$$P[i, \emptyset] = \emptyset$$

34

Hangi eşyaları aldığımızı nasıl buluruz?

	0	1	2	3	4	5
0	∅	∅	∅	∅	∅	∅
1	∅	∅	12	12	12	12
2	∅	10	12	22	22	22
3	∅	10	12	22	30	32
4	∅	10	15	25	30	37

Reconstructing Optimal Solution

Not taken : ↑

$$P[i,j] = P[i-1,j]$$

taken : ←

X

$$\checkmark w_1=2$$

$$\checkmark w_2=1$$

$$\checkmark w_4=2$$

### Edit Distance

Mesaj yazarken yanlış yazılan kelimeyi düzeltme önerileri:

"İki farklı durum var."

① sunny

Sunny

	A	R	T
M			
A			
T			
H			
S			

② sunny

suny

③ sunny

sunnyy

ED[i,j]:

x[i] gelene kadar ki  
y[j] min mesafe

dist(MATHS, ARTS)

MATHS  
-ARTS  
↑↑↑  
MATHS  
ART-S  
↑↑↑

x[1..m] y[1..n]

① copy  $x[i] \rightarrow y[j]$

$$ED[i,j] = ED[i-1,j-1]$$

② change  $x[i] \rightarrow y[j]$

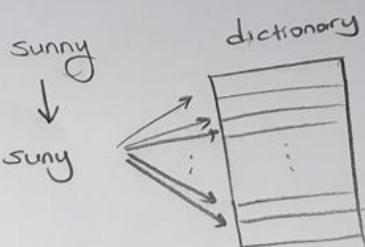
$$ED[i,j] = ED[i-1,j-1] + cost$$

③ insert  $x[i] \rightarrow -$

$$ED[i,j] = ED[i-1,j] + cost$$

④ delete  $- \rightarrow y[j]$

$$ED[i,j] = ED[i,j-1] + cost$$



Recurrence Relation

$ED[i, j] = \min$	$\begin{cases} ED[i-1, j-1] + \phi & : \text{copy} \\ ED[i-1, j-1] + \text{cost} & : \text{change if } (x[i] \neq y[j]) \\ ED[i-1, j] + \text{cost} & : \text{insert if } (x[i] \neq y[j]) \\ ED[i, j-1] + \text{cost} & : \text{delete if } (x[i] \neq y[j]) \end{cases}$																																																				
<table border="1" style="display: inline-table; vertical-align: middle;"> <thead> <tr> <th></th> <th>A</th> <th>R</th> <th>T</th> <th>S</th> </tr> <tr> <th>M</th> <td>0</td> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <th>1</th> <td>1</td> <td>2</td> <td>3</td> <td>4</td> <td></td> </tr> <tr> <th>2</th> <td>2</td> <td>(1)</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <th>3</th> <td>3</td> <td>2</td> <td>2</td> <td>(2)</td> <td>3</td> </tr> <tr> <th>4</th> <td>4</td> <td>3</td> <td>3</td> <td>3</td> <td>3</td> </tr> <tr> <th>5</th> <td>5</td> <td>4</td> <td>4</td> <td>4</td> <td>(3)</td> </tr> </thead> <tbody> <tr> <td style="text-align: right;">↓</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td style="text-align: right;">minimum distance</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>		A	R	T	S	M	0	1	2	3	4	1	1	2	3	4		2	2	(1)	2	3	4	3	3	2	2	(2)	3	4	4	3	3	3	3	5	5	4	4	4	(3)	↓						minimum distance					
	A	R	T	S																																																	
M	0	1	2	3	4																																																
1	1	2	3	4																																																	
2	2	(1)	2	3	4																																																
3	3	2	2	(2)	3																																																
4	4	3	3	3	3																																																
5	5	4	4	4	(3)																																																
↓																																																					
minimum distance																																																					

Longest Common Subsequence

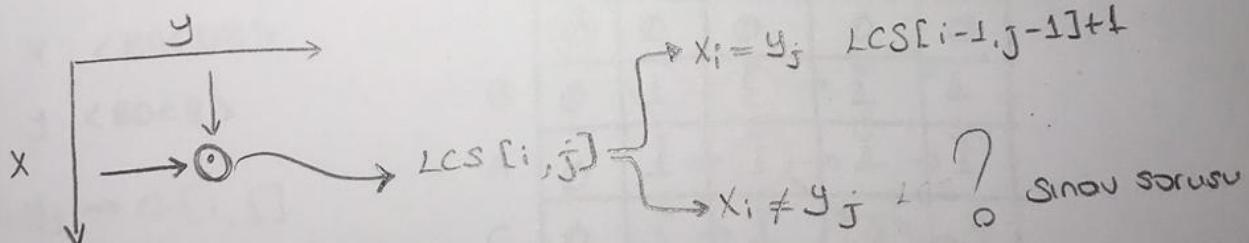
M A R K A

A R T K S A B

) içinde aynı sırada bulunurması

X: BACDB { LCS: BCB  
Y: BDCA B BDB  
                  BAB

DNA dizilerinde kullanılır.



| 10. hafta / 29 Kasım |

### Longest Common Subsequence

$\langle A, C, G, T \rangle \rightarrow \text{DNA dizimleri}$

$$X = \text{ATCTGAT}$$

$$Y = \text{TGCATA}$$

) Longest Common Subsequence  
of  $X$  and  $Y \rightarrow \text{LCS}(X, Y) = \text{TCTA}$   
 $\text{LCS}(X, Y) = \text{TGAT}$

Ortak en uzun alt dizim nedir? Bu soruya cevap arayacağınız.

KOZA LAK Kelimesinin subsequence'lerini bulun.



KALV AZ X

KOLV

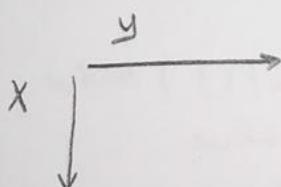
KOLA V

KALK ✓

$$X = \langle x_1, x_2, x_3, \dots, x_m \rangle$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle$$

$$\left. \begin{array}{l} \text{LCS } S = \langle s_1, s_2, \dots, s_k \rangle \\ s_{i_1}, s_{i_2}, \dots, s_{i_k} \\ i_1 < i_2 < \dots < i_k \end{array} \right\}$$



Matris yapısı kullanırsak  
karşılıştırma işlemi daha rahat  
olur.

$$X : \langle BACDB \rangle$$

$$Y : \langle BDCB \rangle$$

$$x_i, y_j \rightarrow C[i, j]$$

$x_m, y_n \rightarrow$  tamamının  
uzunluğunu  
bulcaz.

Geriye giderek  
neler oldığımıza  
bakıcaz.

	B	D	C	B
B	∅	∅	∅	∅
A	∅	1	1	1
C	∅	1	1	2
D	∅	1	2	2
B	∅	1	2	2

Initial

$$\text{if } i = \emptyset \rightarrow c[\emptyset, j] = \emptyset$$

$$\text{if } j = \emptyset \rightarrow c[i, \emptyset] = \emptyset$$

$$c[i, j] = \begin{cases} \emptyset & \text{if } i = \emptyset \text{ or } j = \emptyset \\ c[i-1, j-1] + 1 & i, j > \emptyset, x_i = y_j \\ \max \{c[i-1, j], c[i, j-1]\} & x_i \neq y_j \end{cases}$$

ADD :  $\emptyset$

$$x_i = y_j \rightarrow b[i, j] = \text{ADD}$$

SKIPX : 1

$$x_i \neq y_j \rightarrow \langle [i-1, j] \rangle = c[i, j-1]$$

SKIPY : 2

then

$x[i]$  not in LCS

$b[i, j] = \text{SKIPX}$

else

$b[i, j] = \text{SKIPY}$

$y[j]$  is not in LCS

while ( (i !=  $\emptyset$ ) && (j !=  $\emptyset$ ) ) {

    switch ( b[i][j] ) {

        case ADD:

            add  $x[i]$  to front of LCS

            i--;

            j--;

            break;

        case SKIPX :

            i--;

            break;

        case SKIPY :

            j--;

            break;

}

}

### Vize - 1 Soruları

1) Çalışma zamanı  $8N^2 + 13N^{4/3} + 12N + 3$

$N$  elemanlı dizi  $\rightarrow 1$  saat

$2N$  elemanlı dizi  $\rightarrow ?$  saat

Bellek ve çalışma hızının  $\rightarrow \times 2$  olduğu ortamda çalıştırılacak.

İlgilendirmiş kısım  $\rightarrow N^2$

$N$  yerine  $2N$  koymasak  $(2N)^2 = 4N^2$  yönü

normal şartlarda  $4$  saat sürerdi.

Çalışma hızı  $\times 2$  olduğu için  $2$  saat sürecek.

2)  $3n^3 + 20n^2 + 5 \in \Omega(n^3)$  olduğunu gösteriniz.

$$f(n) < c * g(n)$$

$$3n^3 + 20n^2 + 5 \leq c * n^3 \quad n > n_0, c > 0$$

$$\downarrow \\ 28 \quad n=1 \quad \checkmark$$

$$28 \leq 28$$

3) İki algoritmanın birbiri göre hangisinin daha verimli olduğunu gösteren yöntemler nelerdir?

Time complexity

Space Complexity

- ↳ Best Case
- ↳ Average Case
- ↳ Worst Case

- ↳ Theoretical Analysis
- ↳ Empirical Analysis

4) a) Bubble sort algoritmasının aşağıda verilen sözde kod parçasına göre karmaşıklik analizini yapın. (worst-case iain)

b) Bubble sort algoritmasının best case durumu iain algoritmasının  $O(n)$  olacak şekilde yazınız.

Best-case  $\rightarrow$  already sorted

```
void bubbleSort ( int N, int * dizi ) {
    int i=0, j=0, swap, tmp;
    swap=1;
    while ( (i < N-2) && (swap==1) ) {
        swap=0;
        j=0;
        while ( j < N-2-i ) {
            if ( dizi[j+1] < dizi[j] ) {
                swap=1;
                tmp=dizi[j+1];
                dizi[j+1]=dizi[j];
                dizi[j]=tmp;
            }
            j++;
        }
        i++;
    }
}
```

5)  $n(n+5)$  ve  $5000n^2$  Eşit

$2^{n-1}$  ve  $2^{n+1}$  Eşit

$\log_2(n)$  ve  $\ln(n)$  Eşit

$1000n^2$  ve  $0.001n^3$  Küçük

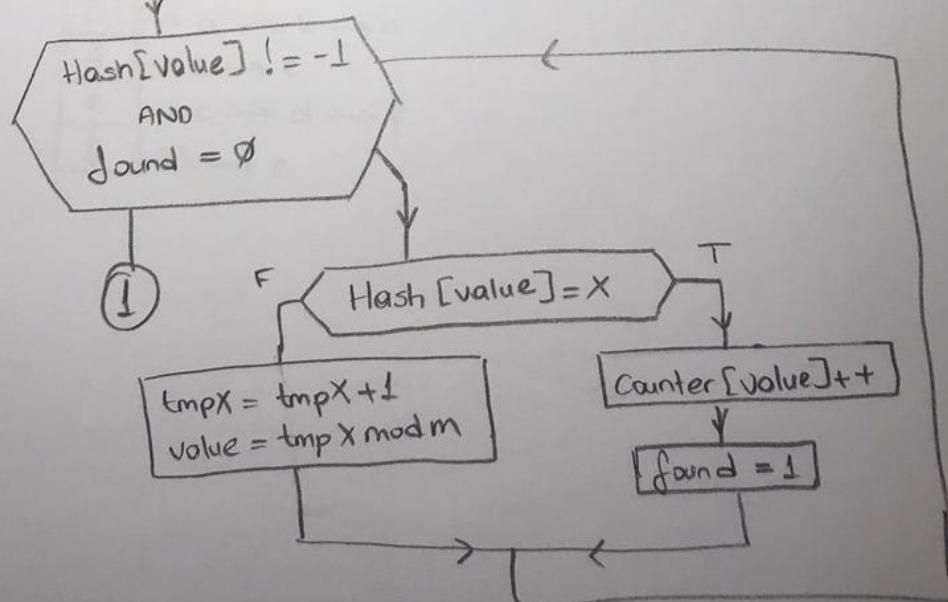
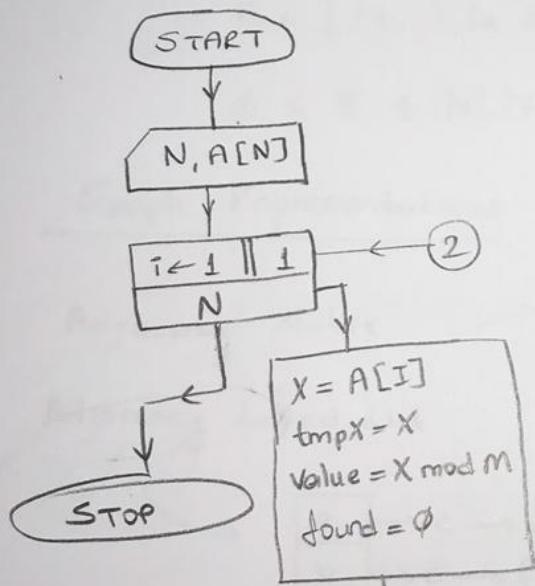
$(n-5)!$  ve  $2^n$  Büyüğ

$(10n+5) \log_2 n$  ve  $100n + (n+n) \log_2 n$  Eşit

Verilen ifadelerin büyük derecelerini karşılaştırarak  
E, B, K ifadelerini yazınız. (Sol, sağdan büyük ise B)

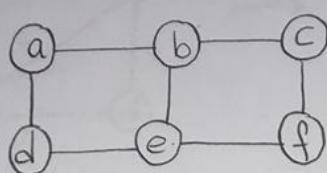
- 6) Bir A dizisindeki ( $N$  elemanlı) sayıların tekrarı kaydedilecektir.  
Linear probing yöntemi ile Hash'e yerleştirilmiştir.

$$\text{hash}(x) = x \bmod m$$

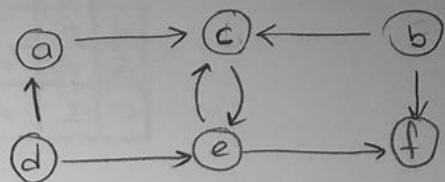


## Graph

$$\rightarrow G = \langle V, E \rangle$$



Undirected



Directed

$$V = \{a, b, c, d, e, f\}$$

$$E = \{(a, c), (a, d), (b, c), (b, f), (c, e), (d, e), (e, f)\}$$

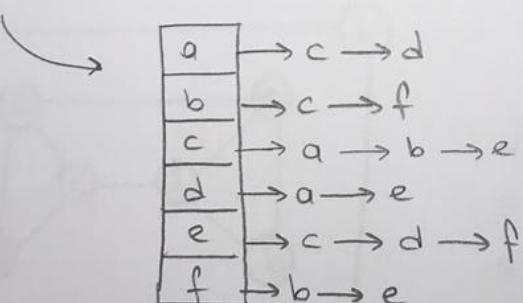
$$\emptyset \leq E \leq (|V| |V-1|)/2$$

## Graph Representations

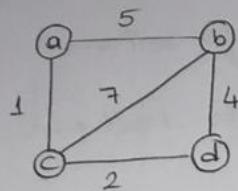
Adjacency Matrix

	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

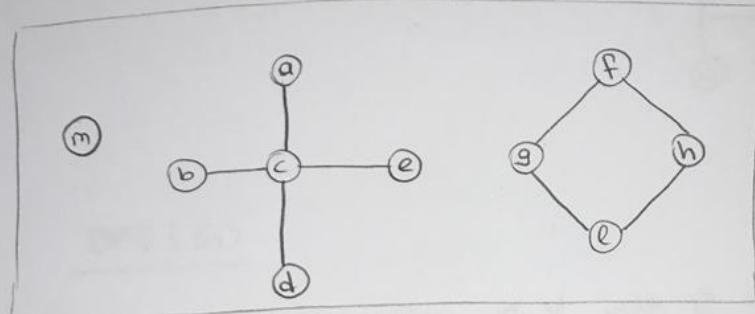
Adjacency Linked List



### Weighted Graph



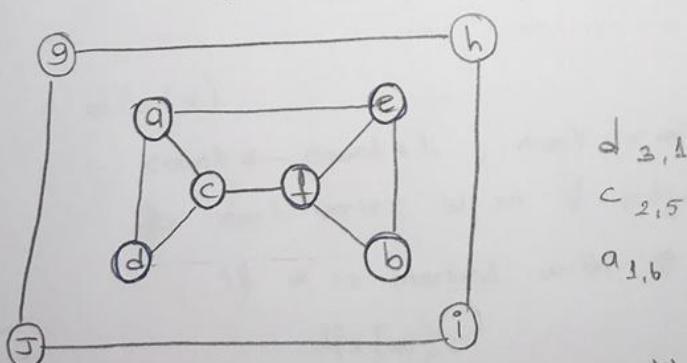
	a	b	c	d
a	0	5	1	$\infty$
b	5	0	7	4
c	1	7	0	2
d	$\infty$	4	2	0



connected  
component  
durumu

### Dept - First - Search

Decrease-by-one technique



e<sub>6,2</sub>

b<sub>5,3</sub>

f<sub>4,4</sub>

d<sub>3,1</sub>

c<sub>2,5</sub>

a<sub>1,6</sub>

j<sub>10,7</sub>

i<sub>9,8</sub>

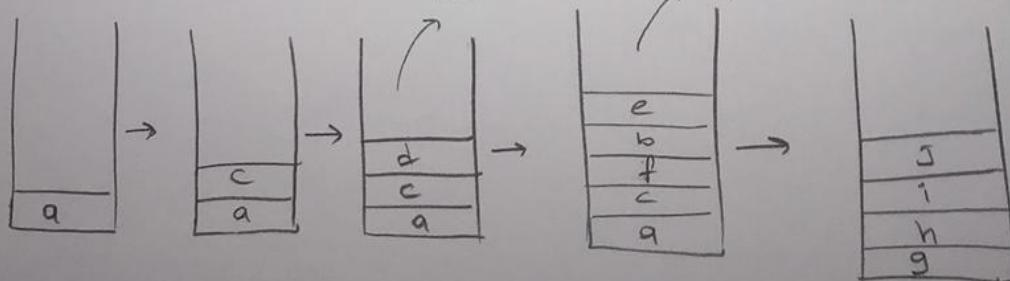
h<sub>8,9</sub>

g<sub>7,10</sub>

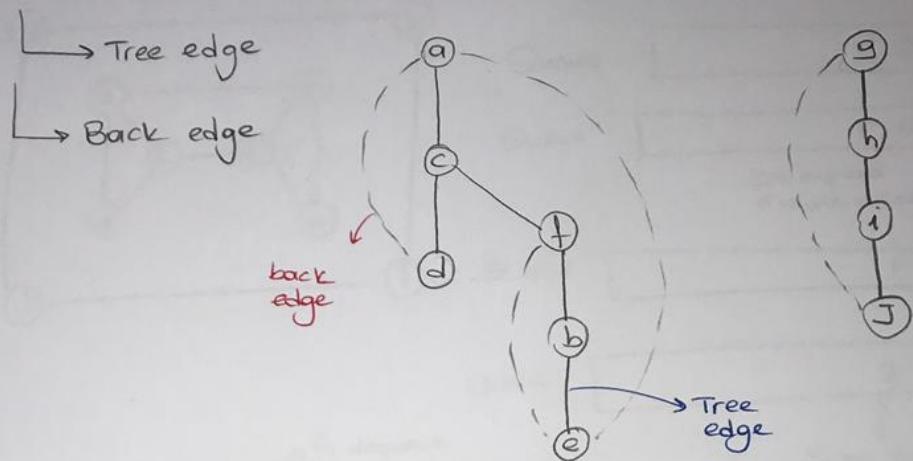
Stack  $\rightarrow$  LIFO

d pop edildi

Bağlantı  
kaldırıldı  
pop edildi



### Dept-First-Search Forest



### DFS ( $G$ )

mark each vertex in  $V$  with  $\emptyset$  as a mark of being unvisited  
 $\text{count} \leftarrow \emptyset$

for each vertex  $v$  in  $V$  do

if  $v$  is marked with  $\emptyset$

$\text{dfs}(v)$

$\text{dfs}(v)$

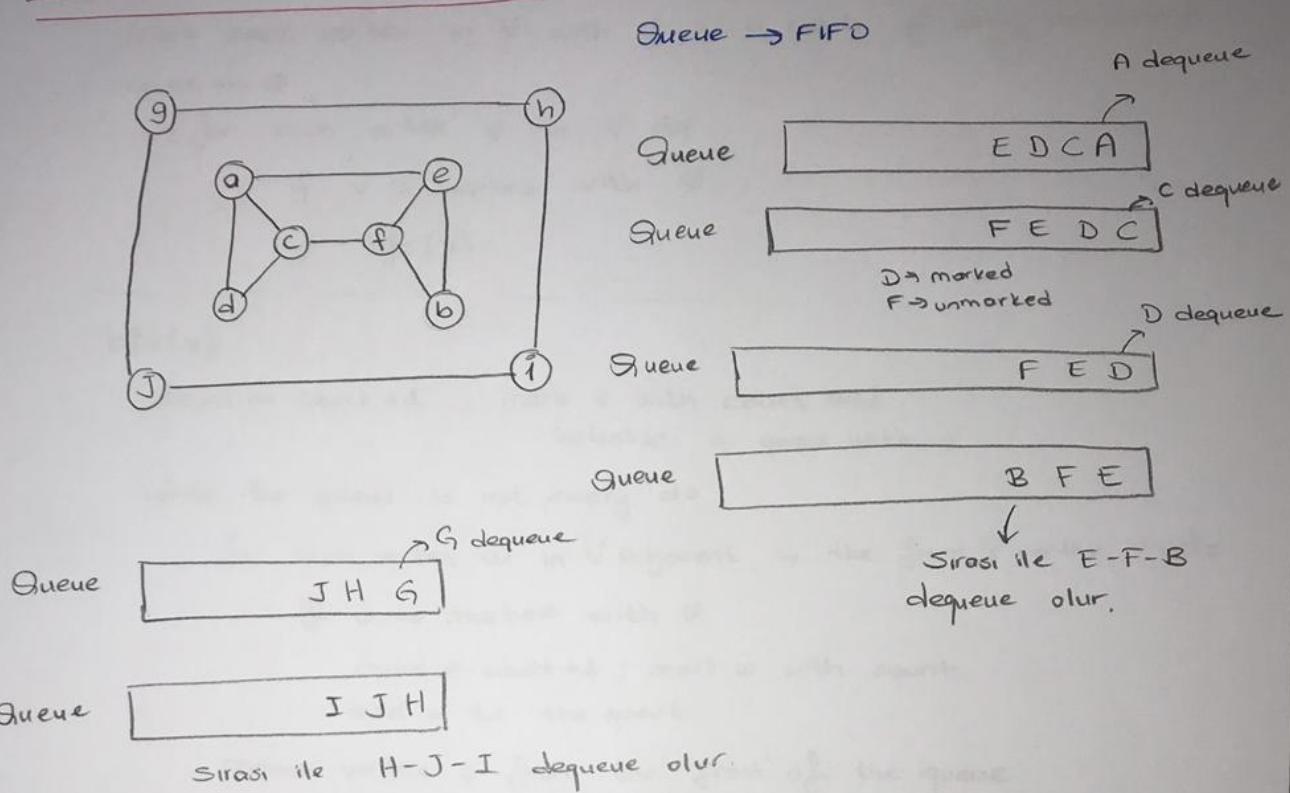
$\text{count} \leftarrow \text{count} + 1$ ; mark  $v$  with  $\text{count}$

for each vertex  $w$  in  $V$  adjacent to  $v$  do

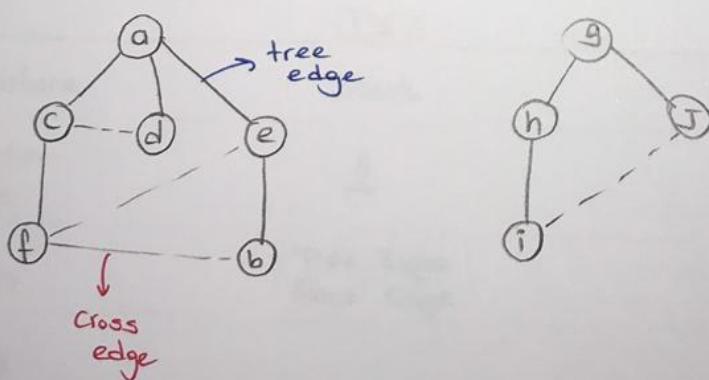
if  $w$  is marked with  $\emptyset$

$\text{dfs}(w)$

### Breadth-First-Search



### Breadth-First-Search Forest



BFS(G)

mark each vertex in  $V$  with  $\emptyset$  as a mark of being unvisited  
 $\text{count} \leftarrow \emptyset$   
for each vertex  $v$  in  $V$  do  
if  $v$  is marked with  $\emptyset$   
    **bfs(v)**

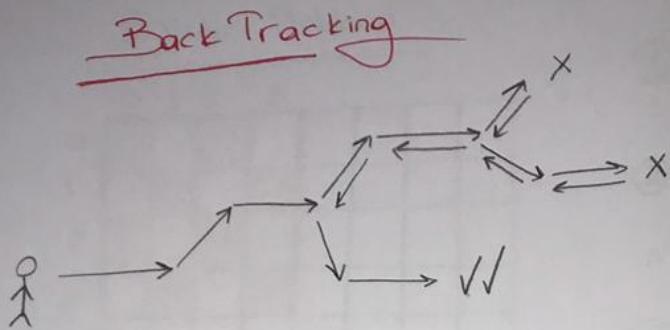
bfs(v)

$\text{count} \leftarrow \text{count} + 1$ ; mark  $v$  with  $\text{count}$  and  
initialize a queue with  $v$

while the queue is not empty do  
for each vertex  $w$  in  $V$  adjacent to the front's vertex  $v$  do  
if  $w$  is marked with  $\emptyset$   
     $\text{count} \leftarrow \text{count} + 1$ ; mark  $w$  with  $\text{count}$   
    add  $w$  to the queue  
remove vertex  $v$  from the front of the queue

	DFS	vs.	BFS
Data Structure	Stack		Queue
No. of vertex orderings	2		1
Edge Types	Tree Edge Back Edge		Tree Edge Cross Edge
Applications			
Efficiency for adjacent matrix	$\Theta V^2 $		$\Theta V^2 $
Efficiency for adjacent linked list	$\Theta V + E $		$\Theta V + E $

46



Aslında yaptığımiz  
sey DFS  
Ancak DFS'nin  
iyileştirilmiş halidir.

= DFS with pruning =

= Stack ya da recursion kullanılır =

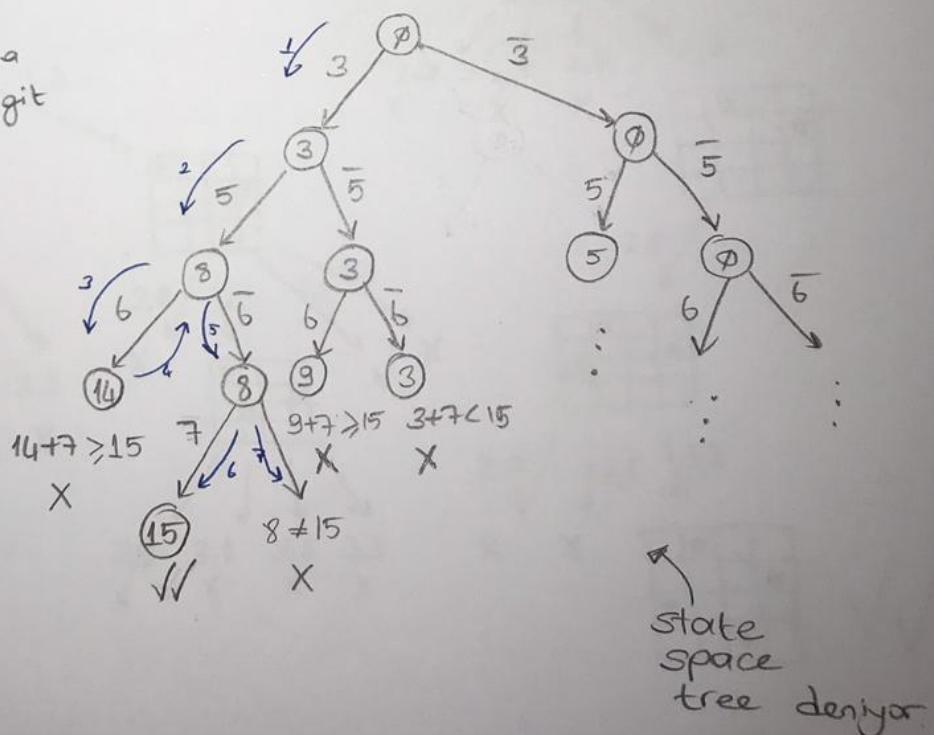
### Subset Sum Problem

$S = \{S_1, S_2, S_3, \dots, S_n\}$  → Pozitif tam sayılarından oluşan sayılar  
 $S_1 \leq S_2 \leq S_3 \dots \leq S_n$

| Toplamı d eden alt kümeleri bulma problemi

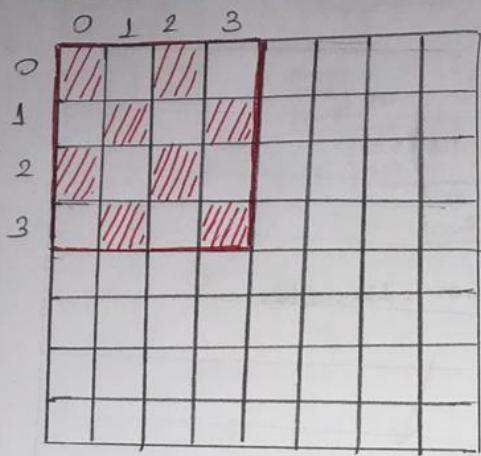
Ex/  $S = \{3, 5, 6, 7\}$   $d = 15$

o En küçük sayıdan başla  
üzerine ekleme ekleme git



47

### 8 Vezir Problemi



$$\textcircled{1} \quad \binom{64}{8} = \frac{64!}{8!(64-8)!} = 4426165368$$

\textcircled{2} One per row

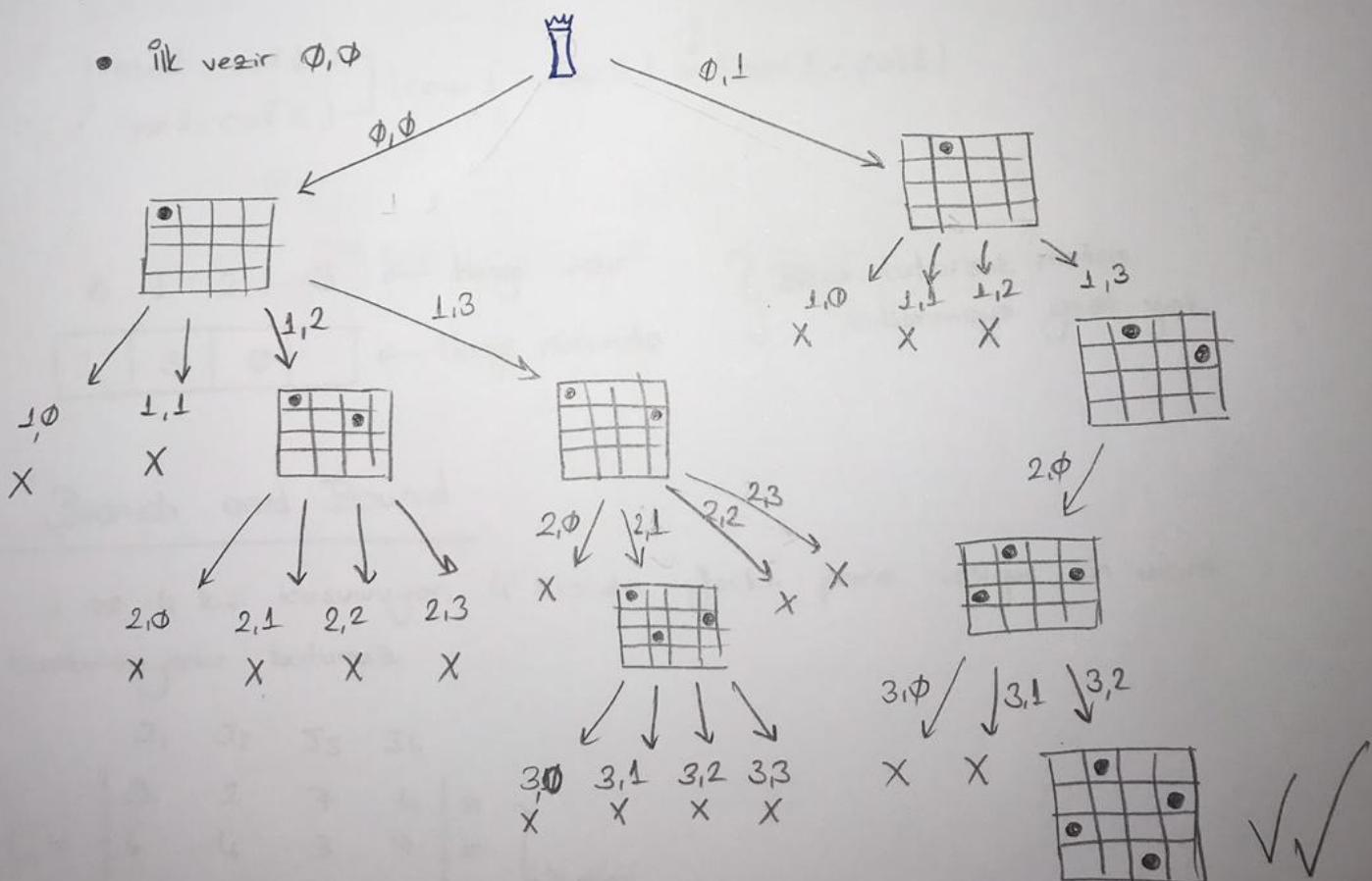
$$n^n \Rightarrow 8^8 \Rightarrow 16777216$$

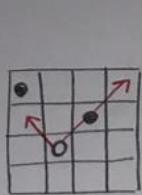
\textcircled{3} One per row and per column

$$n! = 8! = 40320$$

\textcircled{4} Backtracking  $\rightarrow 2057$  adım  
8x8 için

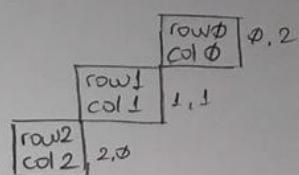
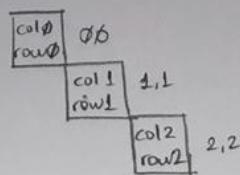
! 8x8 lik zar olasılarından 4x4 lik yapılacak derste.





• → Koyulmuş vezir  
○ → Koyulacak vezir

Diagoneldeki kontrolleri nasıl yaparız?



$$\text{col } 2 - \text{col } 1 = \text{row } 2 - \text{row } 1$$



$$|\text{col}_j - \text{col}_i| = |\text{row}_j - \text{row}_i|$$

$$\frac{\text{col } \emptyset - \text{col } 2}{2} = \frac{\text{row } \emptyset - \text{row } 2}{2}$$

$$\frac{\text{col } 1 - \text{col } 2}{1} = \frac{\text{row } 1 - \text{row } 2}{-1}$$

$$\begin{matrix} (\text{row } 1, \text{col } 1) \\ (\text{row } 2, \text{col } 2) \end{matrix} \quad |\text{row } 1 - \text{row } 2| = |\text{col } 1 - \text{col } 2|$$

$$\begin{matrix} \emptyset & 1 & 2 & 3 \end{matrix} \leftarrow \text{Hangi vezir}$$

$$\boxed{2 \mid 3 \mid \emptyset} \leftarrow \text{Hangi sütunda}$$

Boyle tutarsız matris kullanmaya gerek yok.

### Branch and Bound

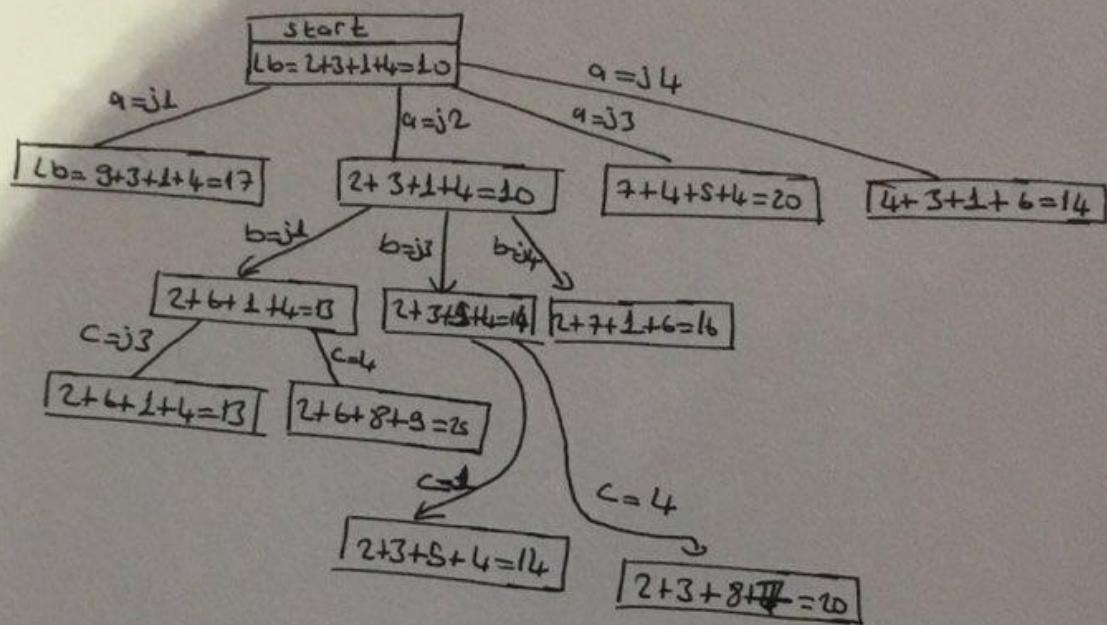
4 ise 4 kişi basıruyor. 4 kişide farklı para istiyor. En ucuz kombinasyonu bulunuz.

$$C = \left( \begin{array}{cccc|c} J_1 & J_2 & J_3 & J_4 & \\ \hline 9 & 2 & 7 & 4 & a \\ 6 & 4 & 3 & 7 & b \\ 5 & 8 & 1 & 8 & c \\ 7 & 6 & 9 & 4 & d \end{array} \right) \left. \begin{array}{l} \\ \\ \\ \end{array} \right\} \text{Kisiler}$$

$\underbrace{\hspace{10em}}$  işler

(37)

$$Lb = 2+3+1+4 = 10 //$$



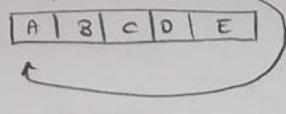
ilk olarak job farklı etmemek için en küçük sayıları topla  $2+3+1+4=10$  kişi için bütün joblar için yeni sonuçlar bUL.  $A=j3$  olduğunda demet oluyor ki diğerleri  $j3$ 'ü alamaz. Diğerlerinin çakışıp çakışmadığını kontrol etmeden en düşük 2. fiyatını alıyoruz.  $A=j2$  en düşük sonuç ordan devam edip aynı işlemleri b kişi için yapıyoruz. b j1 için 13, b j3 sese se  $\begin{matrix} a=2 \\ b=3 \\ c=5 \\ d=4 \end{matrix}$  böyle olmak zorunda. böyle böyle devam ediyoruz.

## Priority Queue

### Queue

First In First Out

A  $\leftarrow$  B  $\leftarrow$  C  $\leftarrow$  D



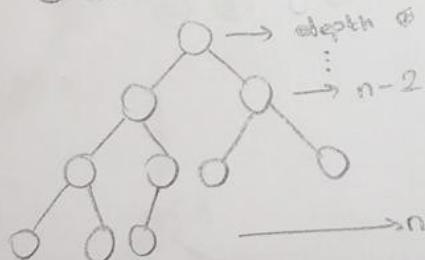
\* array

\* linked list

! Max. kalanca iisin  
nasıl bir yapı kullanabiliriz?  
- Heap Tree yapısı kullanabiliriz.

### Heap Tree

#### (a) Balanced



All the nodes at depth  $\phi$  through  $(n-2)$  have two children!

#### (1) Unsorted Array

3	5	1	4	2	6
---	---	---	---	---	---

- insert  $O(1)$

- remove  $O(n)$

#### (2) Sorted Array

2	4	8	12	25	27
---	---	---	----	----	----

- insert  $O(n)$

- remove  $O(1)$  veya  $O(n)$

↓

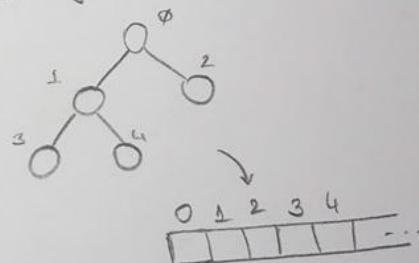
circle array

ya da

linked list

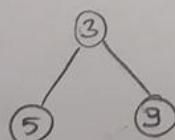
kullanılırsa

#### (b) Left Justified

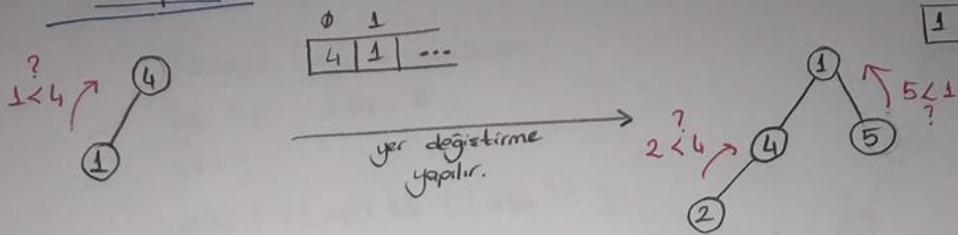


#### (3) Complete Binary Tree

② a parent's value  $\leq$  its children's values : Min Heap



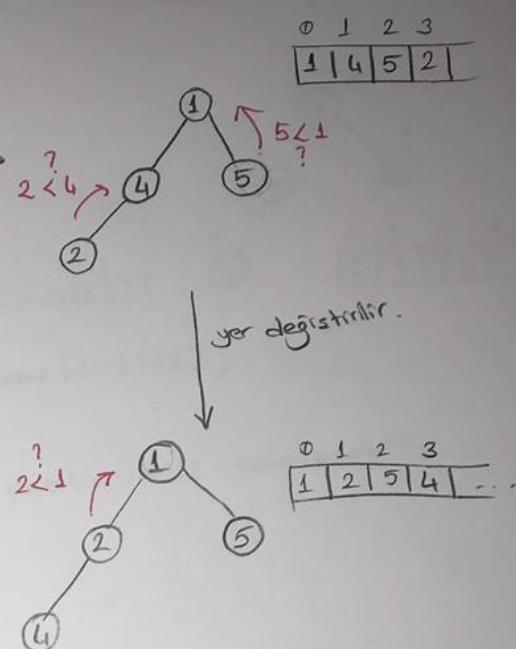
### Heapify



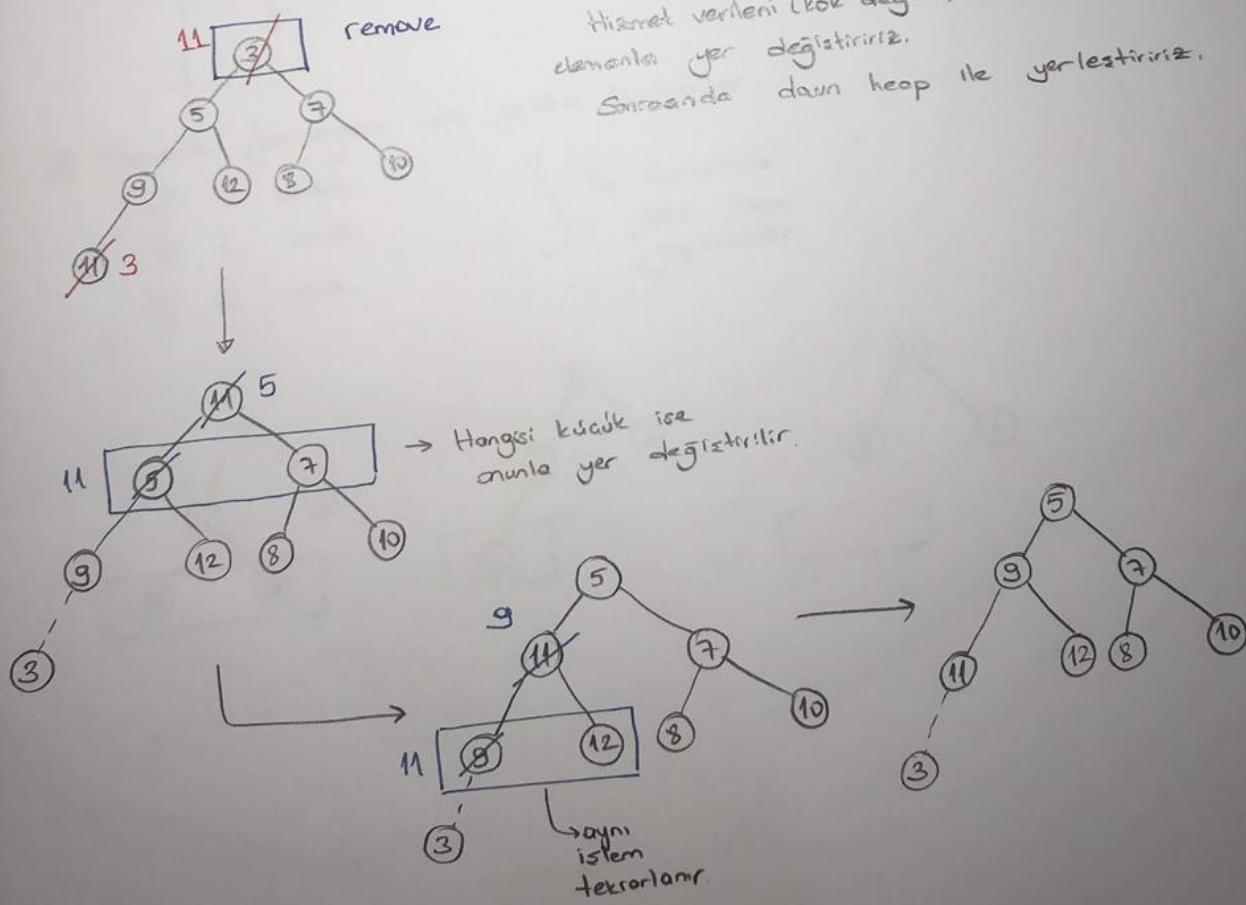
Up heap  
islemi denir.

Bu islemi  
insert icin kullanabiliriz.

$O(\log_2 n)$



Hizmet verken (kök degeri) en sondaki  
elementin yer degistirilir.  
Sonaonda down heap ile yerlestirilir.

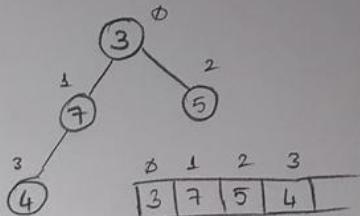


Void insert ( int newItem, int n, int pQueue[] ) {

```

    |   pQueue[n] = newItem;
    |   int i=n, stop=∅ ;
    |   n++;
    |   while ( (i>∅) && (!stop) ) {
    |       |   if ( pQueue[i] <= pQueue[(i-1)/2] ) {
    |           |       |   pQueue[i] ↔ pQueue[(i-1)/2];
    |           |       |   // swap işlemi
    |           |       |   i=(i-1)/2; // parent=(i-1)/2 tanımlanabilir.
    |           |   }
    |           |
    |           |   else {
    |           |       |   stop = 1;
    |           |   }
    |           |
    |       }
    |
    |   }

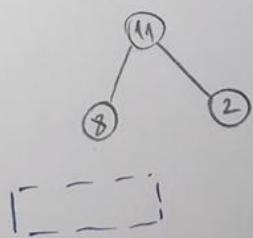
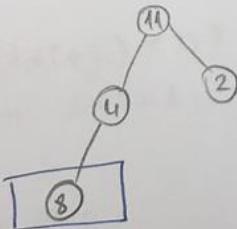
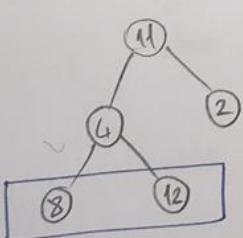
```



\* Remove fonksiyonu için gerekli kontroller

- (1) 2 cocuk
- (2) 1 cocuk
- (3) ∅ cocuk

} Bu durumlara  
göre işlem  
yapılmalıdır.



```

int remove (int n, int pQueue[ ]) {
    int min, i
    if (n == 0)
        return -1;
    min = pQueue[0];
    n--;
    i = 0;
    adr = findSmallChild(i, pQueue, n);
    while ((adr > 0) && (pQueue[adr] < pQueue[i])) {
        pQueue[adr] ↔ pQueue[i]; // swap islemi
        i = adr;
        adr = findSmallChild(i, pQueue, n)
    }
    return min;
}

int findSmallChild (int i, int pQueue[], int n) {
    if ((2 * i + 2) < n) { // 2 cocuk durumu
        if (pQueue[2 * i + 1] < pQueue[2 * i + 2])
            return 2 * i + 1;
        else
            return 2 * i + 2;
    } else if ((2 * i + 1) < n)
        return 2 * i + 1;
    else
        return 0;
}

```