

①

13:15  
başlangıç

# ALGORITMA ANALİZİ

- Mine Elif KARSLIGİL -

## Kaynaklar

The Design & Analysis of Algorithm, A. Levitin

bas ucu kitabı gibi  
(hemen her temel  
konu var.)

Algorithms : 4th Edition, R. Sedgewick, K. Wayne

Princeton Uni.

The Algorithm Design Manual, S. Skiena

hocaları yazmış.  
(Hoca en çok bun-

dan yararlanıyor)

Hikaye kitabı gibi (Biraz Advanced)

## Değerlendirme

-3 Ödev → %15

-Proje → %10

-2 Midterm → %45

-Final → %30

② Ödevlerde kullanılacak programlama dili C (ödevde göre değişebilir.)

eleman sayısı  
elemanların veriliş sırası

- Best case

- Average case : en mantıklısı (ama hesabi zor)

- Worst case

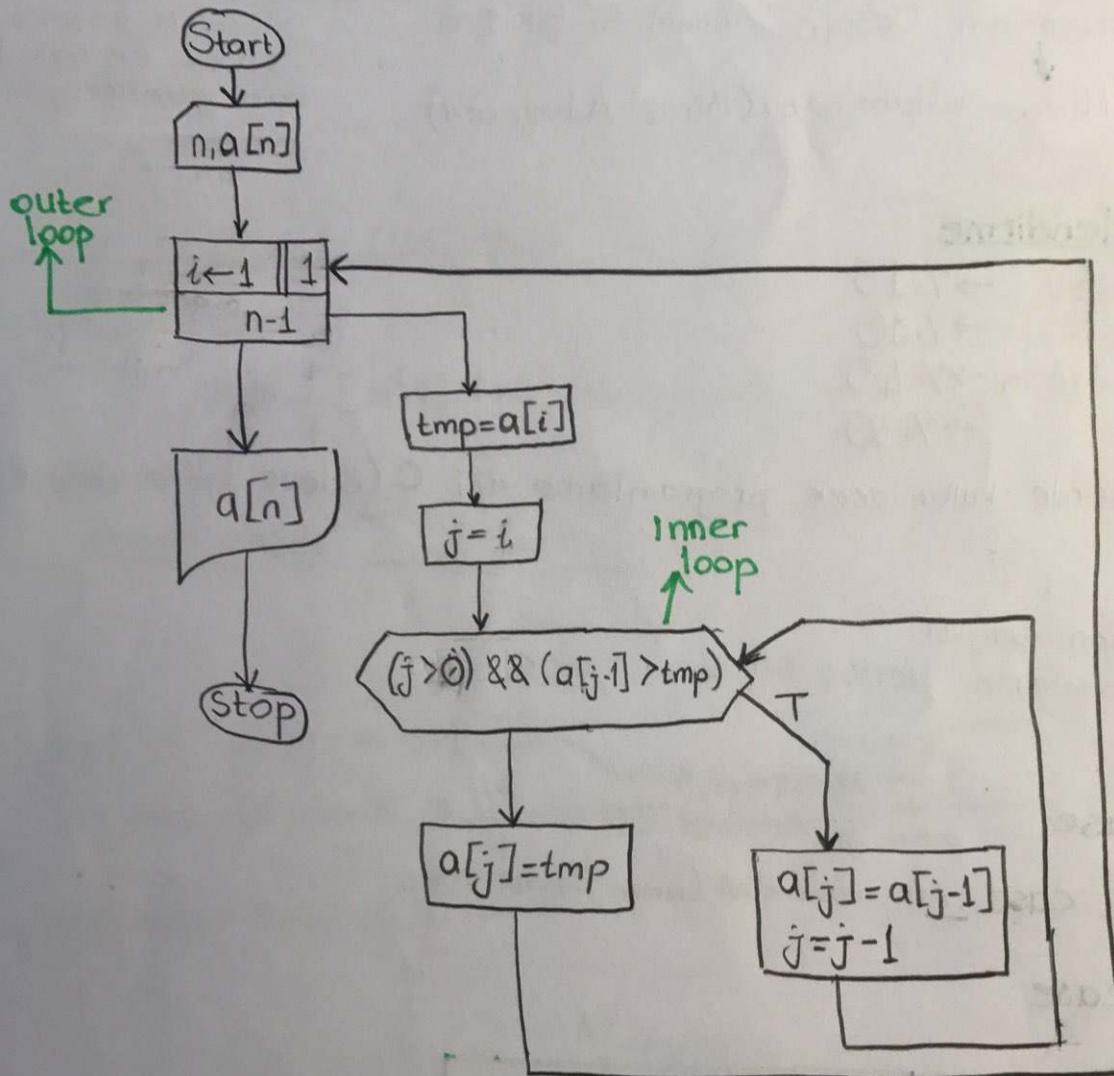
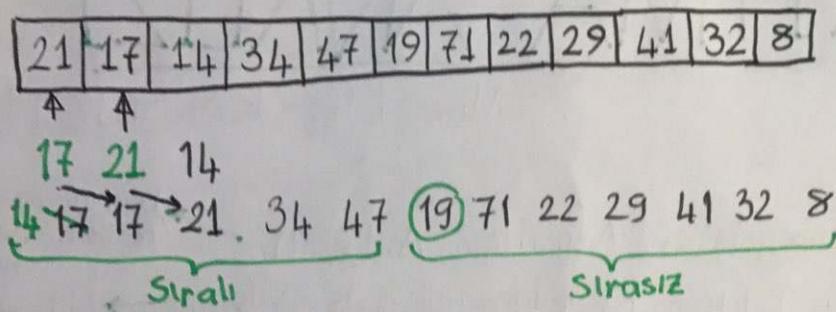
$$\sum_{i=1}^n i = 1+2+3+\dots+n = \frac{n.(n+1)}{2}$$

$$\sum_{i=1}^n i^2 = 1^2+2^2+3^2+\dots+n^2 = \frac{n.(n+1).(2n+1)}{6} = \frac{1}{3} n^3$$

$$\sum_{i=1}^n \lg i = n * \lg i$$



(2)

insertion sort

worst case: Outer loop :  $1 \dots n-1$   
inner loop :  $j=1$   
 $j=2$   
 $j=3$   
 $\vdots$   
 $j=n-1$

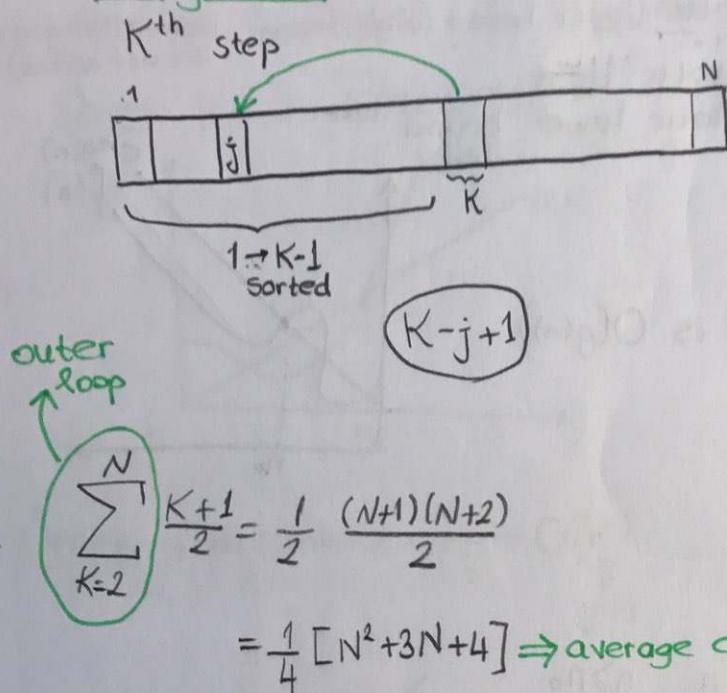
$\left. \begin{array}{c} \\ \\ \\ \\ \end{array} \right\} n-1$

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

$$= \frac{n * (n-1)}{2}$$



(3)

Average case:

Best case:  $\sum_{i=1}^{n-1} 1 = \frac{n-1}{n} \approx n$

- Best case  $\approx n-1$
- Average case  $\approx \frac{n^2}{4}$  Asymptotic Notation
- Worst case  $\approx \frac{n^2}{2}$

inner loop

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

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

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

$$\sim \frac{K+1}{2}$$

Ortalama olarak K.gözdəki elemən tain iç döngüye giri me miktonı

Big-OH

$$\begin{aligned} n &\rightarrow O(n) \\ n^2 &\rightarrow O(n^2) \\ n^2 &\rightarrow O(n^2) \end{aligned}$$

Asymptotic Notation

$$\boxed{\frac{N^2}{4} + \frac{3N}{4} + \frac{1}{2}} \xrightarrow[n \rightarrow \infty]{\text{sabitler elenir}} \frac{N^2}{4} + 3N + \cancel{X}$$

Katsayılar elenir.

$$N^2 \leftarrow \text{Sonuç} \quad N^2 + \cancel{X} \leftarrow \text{En yüksek dereceli terim alınır} \quad \frac{N^2}{4} + 3N$$

Algoritmanın karmaşıklık mertebesi

(işletim sistemi, programlama dili, yazılım ve donanımdan bağımsız)



4

### Asymptotic Notation

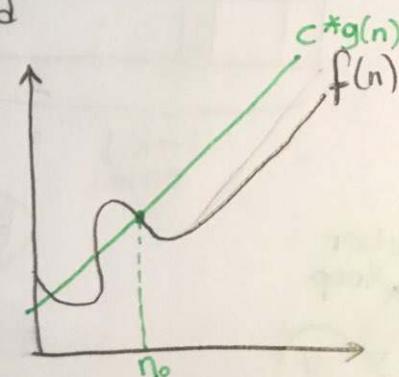
Big-Oh:  $O \rightarrow$  asymptotic upper bound (algoritmanın karmaşıklığında en üst nokta)

Big-theta:  $\Theta \rightarrow$  asymptotic tight

Big-omega:  $\Omega \rightarrow$  asymptotic lower bound

$$f(n) = 4n^2 + 2n$$

$$\left. \begin{array}{l} f(n) \leq c * g(n) \\ n > n_0 \text{ ve } c > 0 \end{array} \right\} f(n) \text{ is } O(g(n))$$



Örnek:  $f(n) = 2n + 10$   
 $g(n) = n$  ?

$$2n + 10 \leq c * n \quad n > n_0$$
$$10 \leq (c-2) * n \quad c=3 \quad n=10$$

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

$\uparrow$   
 $g(n)$

Örnek:  $f(n) = 2n + 10$   
 $g(n) = n^2$

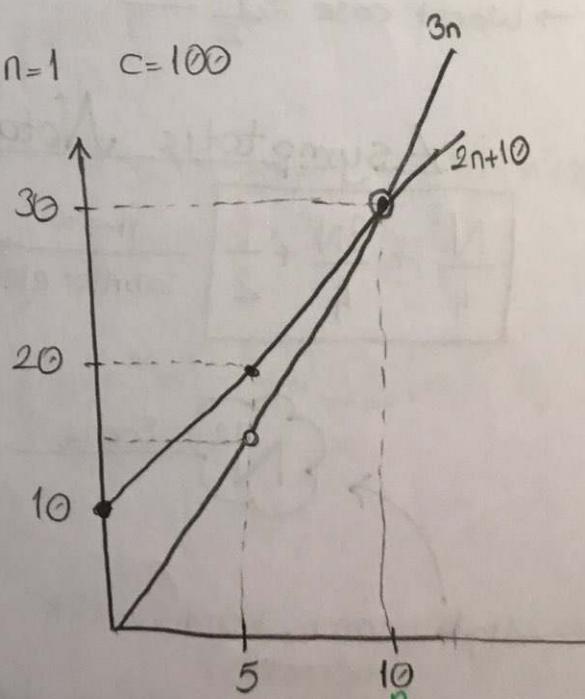
$$2n + 10 \leq c * n^2 \quad n > n_0, c > 0$$

$$cn^2 - 2n \geq 10 \quad n=1 \quad c=100$$

$$f(n) = O(g(n))$$

$$2n + 10 = O(n^2)$$

$\uparrow$   
 $\uparrow$

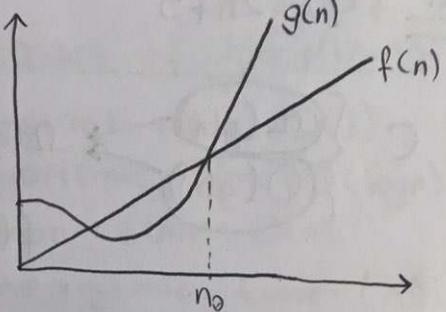


(5)

### Big-Oh Notation

$f(n) \in O(g(n))$  if  $0 \leq f(n) \leq c * g(n)$ ,  $n > n_0$ ,  $c > 0$

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



$$2n + 10 \leq c * n \quad n > n_0$$

$$10 \leq (c-2)n \quad n_0 = 1$$

$$10 \leq c-2 \rightarrow 12 \leq c$$

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

$$\downarrow g(n)$$

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

$$16 \leq c * 9$$

$$n > n_0 > 0 \quad n=3$$

$$c=2, n=3$$

Örnek:  ~~$f(n^2) \in O(n)$~~

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

$$n \leq c$$

### Big-Omega Notation

$f(n) \in \Omega(g(n))$  if  $c > 0$ ,  $n_0 > 0$ ,  $n > n_0 \rightarrow f(n) \geq c * g(n)$

asymptotic lower bound

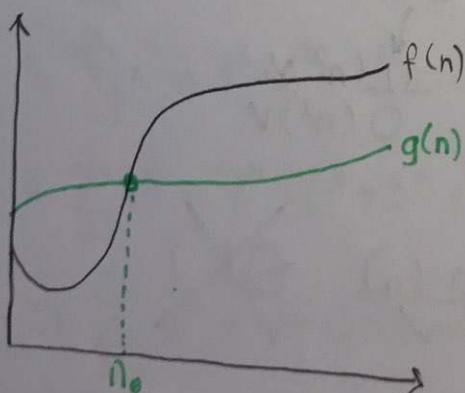
sequential search  $\rightarrow$  best case:  $\Omega(1)$

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

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

$$5n^2 \geq c * n^2, c > 0, n > n_0, n_0 > 0$$

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



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

?

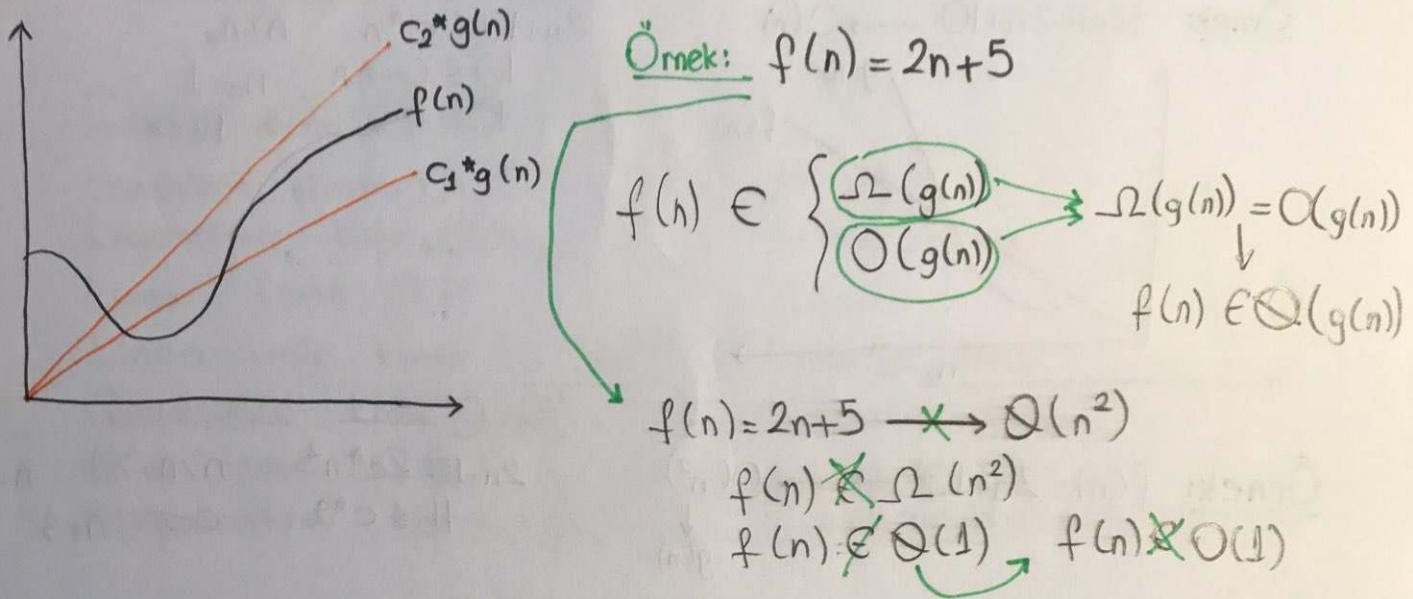
✓



⑥

Big-theta ( $\Theta$ ) Notation: tight bound  $\rightarrow f(x)$  is asymptotically equal to  $g(x)$

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad c_1 < c_2, n_0 > 0, n > n_0 \rightarrow f(n) = \Theta(g(n))$$



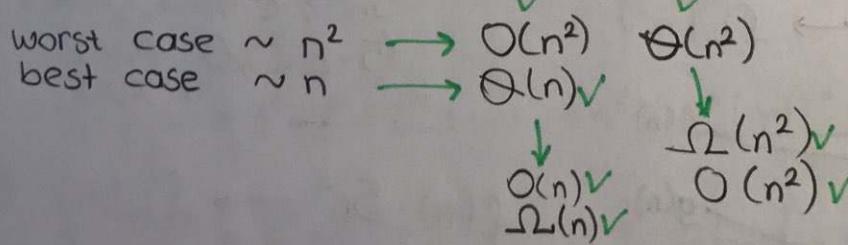
Örnek:  $f(n) = 2n + 5 \in O(n)$

$$\textcircled{1} \quad 2n+5 \in \Omega(n) \rightarrow 2n+5 \geq c \cdot n \rightarrow \begin{array}{l} 5 \geq (c-2) \cdot n \\ n_0 = 1, c = 3 \end{array} \checkmark$$

$$\textcircled{2} \quad 2n+5 \in O(n) \rightarrow 2n+5 \leq c \cdot n \rightarrow \begin{array}{l} 5 \leq (c-2) \cdot n \\ n_0 = 1, c = 7 \end{array} \checkmark$$

$$c_1 = 3, c_2 = 7, n_0 = 1 \rightarrow [c_1 \cdot n \leq 2n+5 \leq c_2 \cdot n] \rightarrow 2n+5 \in \Theta(n)$$

Örnek: Insertion Sort



Insertion Sort  $\rightarrow O(n^2)$   $\Omega(n)$   $\Theta()$

(7)

Örnek: Merge SortBest case  $\sim n^* \log n$   
Worst case  $\sim n^* \log n$  $\rightarrow \Theta(n^* \log n)$ Classes of AlgorithmsConstant time:  $O(1)$  $a = b + c$ Logarithmic time:  $O(\log n)$ 

binary search

Linear time:  $O(n)$ 

loop

Linearithmic time:  $(n^* \log n)$ 

merge sort

Quadratic time:  $O(n^2)$ 

double loop

Cubic time:  $O(n^3)$ 

triple loop

Exponential time:  $O(2^n)$ 

exhaustive search

Useful Formulas for the Analysis of Algorithms- Property of Logarithms

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

- Combinatorics

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

- Important Summation Formulas

$$1. \sum_{i=l}^u 1 = \underbrace{1+1+1+\dots+1}_{u-l+1} = u-l+1 \quad (l \leq u)$$

$$2. \sum_{i=1}^n i = 1+2+3+\dots+n = \frac{n*(n+1)}{2} \approx \frac{1}{2} n^2$$

$$3. \sum_{i=1}^n i^2 = 1^2+2^2+3^2+\dots+n^2 = \frac{n*(n+1)*(2n+1)}{6} \approx \frac{1}{3} n^3$$

$$4. \sum_{i=1}^n i^k = 1^k+2^k+3^k+\dots+n^k \approx \frac{1}{k+1} * n^{k+1}$$

$$5. \sum_{i=0}^n a^i = 1+a+a^2+\dots+a^n = \frac{a^{n+1}-1}{a-1} \quad (a \neq 1)$$

$$\boxed{\sum_{i=0}^n 2^i = 2^{n+1}-1}$$



(8)

$$6. \sum_{i=1}^n i \cdot 2^i = 1 \cdot 2 + 2 \cdot 2^2 + \dots + n \cdot 2^n = (n-1) \cdot 2^{n+1} + 2$$

$$7. \sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \approx \ln n + \gamma \quad (\gamma \approx 0.5772)$$

$$8. \sum_{i=1}^n \lg i \approx n \cdot \lg n$$

### - Sum Manipulation Rules

$$1. \sum_{i=l}^u c \cdot a_i = c \cdot \sum_{i=l}^u a_i$$

$$2. \sum_{i=l}^u (a_i + b_i) = \sum_{i=l}^u a_i + \sum_{i=l}^u b_i$$

$$3. \sum_{i=l}^u a_i = \sum_{i=l}^m a_i + \sum_{i=(m+1)}^u a_i \text{ where } (l \leq m \leq u)$$

### - Floor and Ceiling Formulas

$$\lfloor 3.8 \rfloor = 3$$

$$1. x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$$

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

$$2. \lfloor x+n \rfloor = \lfloor x \rfloor + n \text{ and } \lceil x+n \rceil = \lceil x \rceil + n \quad (\text{for real } x \text{ and integer } n)$$

$$\lceil 3.8 \rceil = 4$$

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

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

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

### Mathematical Analysis of Nonrecursive Algorithms

Örnek:

START

N, A[N]

max = A[0]

I ← 1 // 1

N-1

max

STOP

Two operations :

A[I] &gt; max

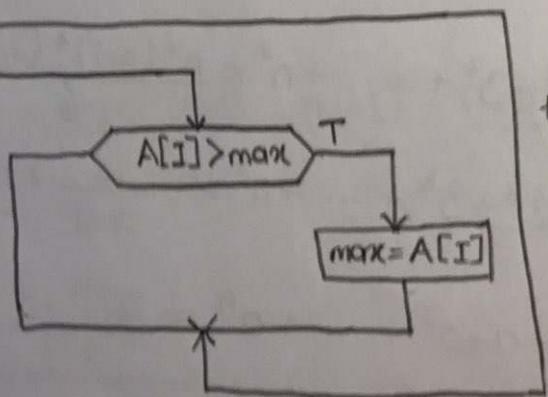
max = A[I]

→ Basic operation  
 Gündül if kontrolü her zaman yapılmıyor, atama bozun oluyor.

0	1	2	3	4	5	6	7
7	1	3	2	8	4	6	5

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

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



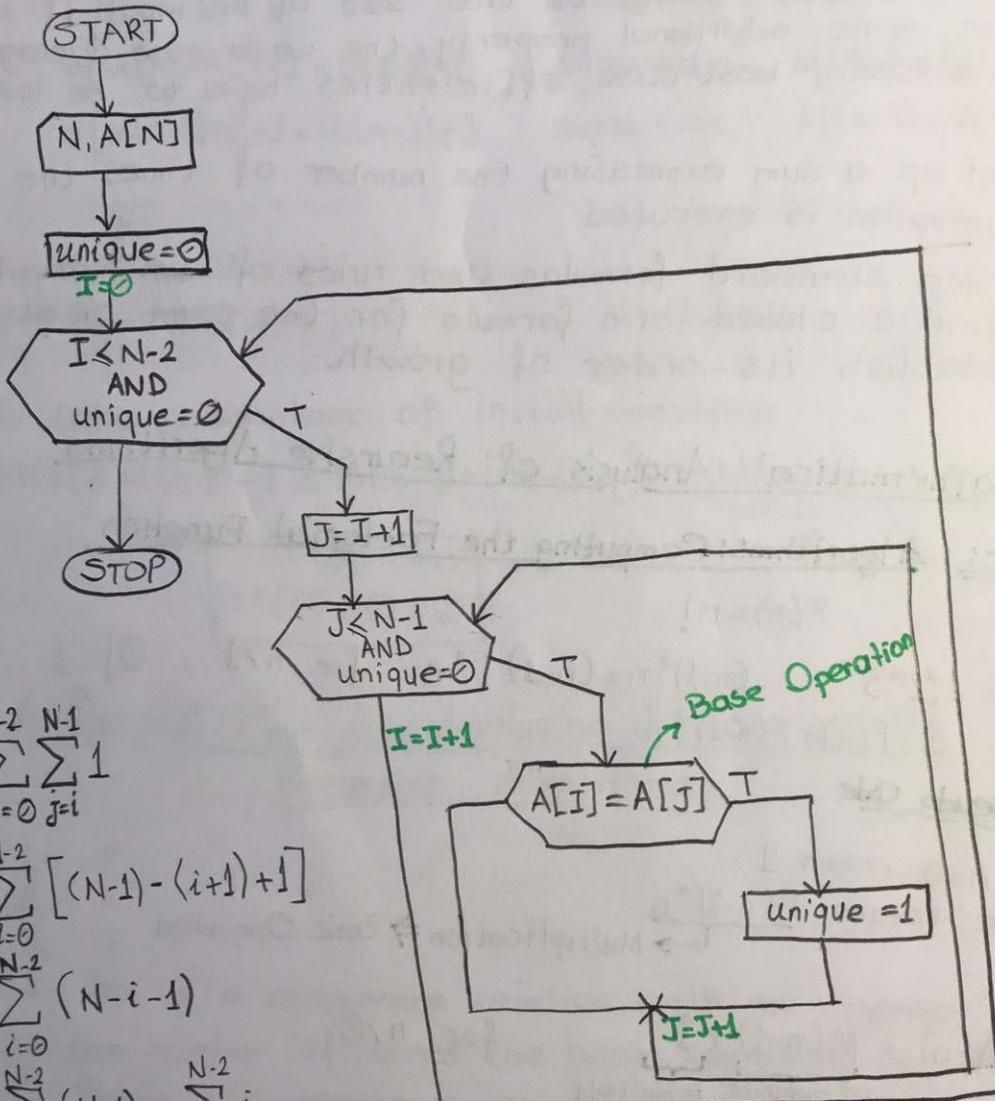
9

## Örnek: Algorithms: Element Uniqueness Problem

3	5	7	3	1	2	4
---	---	---	---	---	---	---

2	7	1	3	4	6	5
---	---	---	---	---	---	---

unique



$$C_{\text{worst}} = \sum_{i=0}^{N-2} \sum_{j=i}^{N-1} 1$$

$$= \sum_{i=0}^{N-2} [(N-1) - (i+1) + 1]$$

$$= \sum_{i=0}^{N-2} (N-i-1)$$

$$= \sum_{i=0}^{N-2} (N-1) - \sum_{i=0}^{N-2} i$$

$$= (N-1) * \sum_{i=0}^{N-2} 1 - \frac{(N-2)*(N-1)}{2} = (N-1)^2 - \frac{(N-2)*(N-1)}{2} = \frac{N*(N-1)}{2}$$

$$\approx \frac{1}{2} N^2 \in O(N^2)$$

10

## General Plan for Analyzing of Non-recursive Algorithms

1. Decide on a parameter (or parameters) indicating on input's size.
2. Identify the algorithm's basic operation (As a rule, it is located in its innermost 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

### Example: Algorithms: Computing the Factorial Function

$$F(n) = n!$$

$$n! = 1 * 2 * 3 * \dots * (n-1) * n = (n-1)! * n \quad \text{for } n > 1, \quad 0! = 1$$

$$F(n) = F(n-1) * n!$$

#### Pseudo Code

```

if n=0 return 1
else return F(n-1) * n
            ↗ Multiplication ⇒ Basic Operation
    
```

$$M(n) = M(n-1) + 1 \quad \text{for } n > 0$$

M(n-1)      +1  
 to compute      to multiply  
 F(n-1)      F(n-1) by n

#### → Recurrences Relations ←

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

if  $n=0$  return 1

$M(0) = 0$       ↗ no multiplication  
 ↑                      when  $n=0$   
 the calls stop when  $n=0$



Recurrence Relation:

$$M(n) = M(n-1) + 1 \quad \text{for } n > 0$$

$$M(0) = \emptyset$$

Method of Backward Substitution

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

$$\begin{array}{ll} \text{substitute } M(n-1) = M(n-2) + 1 & \\ \text{substitute } M(n-2) = M(n-3) + 1 & \\ \text{substitute } M(n-3) = M(n-4) + 1 & \end{array}$$

$$M(n) = M(n-i) + i$$

To take advantage of initial condition

$$M(n) = M(n-1) + 1 = M(n-i) + i = \dots = M(n-n) + n = n$$

$$\boxed{n=0, i=n \\ M(0)+0=0}$$

A General Plan for Analyzing Efficiency of Recursive Algorithms

1. "
2. "
3. "
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.

(12)

Example:

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

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

Initial condition:

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

$$n=2^k$$

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

$$A(2^0) = \emptyset$$

Backward Substitution:

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

$$\begin{aligned} &\qquad\qquad\qquad = A(2^{k-2}) + 2 \\ &\qquad\qquad\qquad = A(2^{k-3}) + 3 \\ &\qquad\qquad\qquad = A(2^{k-i}) + i \end{aligned}$$

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

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

$$\begin{aligned} &\qquad\qquad\qquad = A(2^{k-k}) + k \\ &\qquad\qquad\qquad = A(2^0) + k \\ &\qquad\qquad\qquad = \underbrace{A(2^0)}_{0} + k \\ &\qquad\qquad\qquad = k \end{aligned}$$



## Binary Search Algorithm

Input :  $A[0, \dots, n-1]$

Output : Index or -1

### Pseudo Code:

```

 $l \leftarrow 0, r \leftarrow n-1$ 
while ( $L \leq r$ ) do
     $m \leftarrow L(l+r)/2$ 
    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	70	74	81	85	93	98
iteration1	<u><math>l</math></u>						<u><math>m</math></u>						<u><math>r</math></u>
iteration2								<u><math>l</math></u>	<u><math>m</math></u>				<u><math>r</math></u>
iteration3								<u><math>l</math></u>	<u><math>r</math></u>				

Search value = 70

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

### Initial Condition

$$\underline{\underline{C_w(1) = 1}} \quad \underline{\underline{\dots}}$$

$n = 2^k$

### Backward Substitution

$$C_w(2^k) = k+1 = \log_2 n + 1$$

(14)

Verify by Substitution that function  $C_w(n) = \lfloor \log_2 n \rfloor + 1$

indeed satisfies equation .... for any positive even number

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

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

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

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

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

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

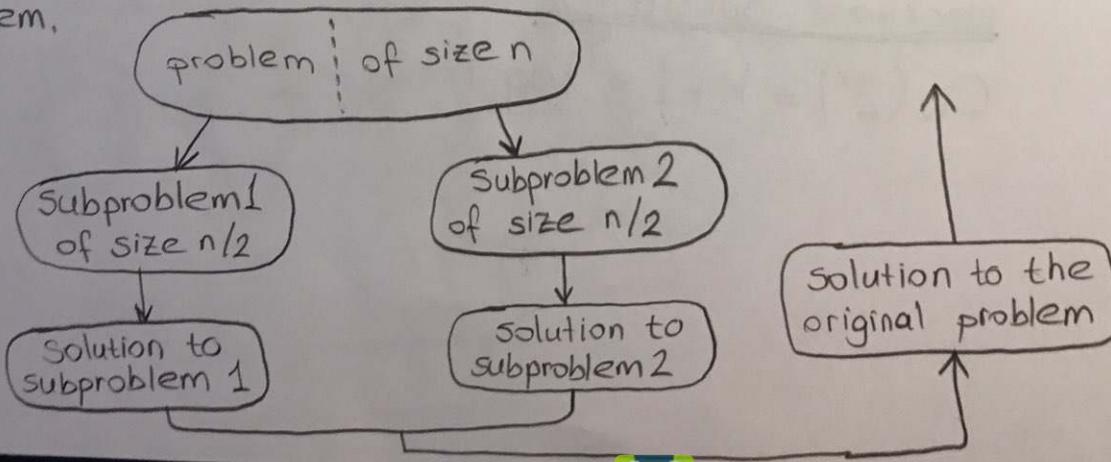
$$\lfloor \log_2 10^3 \rfloor + 1 = 10$$

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

### 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 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.



Computing the sum of  $n$

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

Instance size:  $n$

Subproblems:  $b$

the subproblems that need to be solved:  $a$

[General Divide-and-Conquer Recurrence]

$$T(n) = a T(n/b) + f(n) \rightarrow \text{Running Time}$$

$f(n)$  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  $d \geq 0$

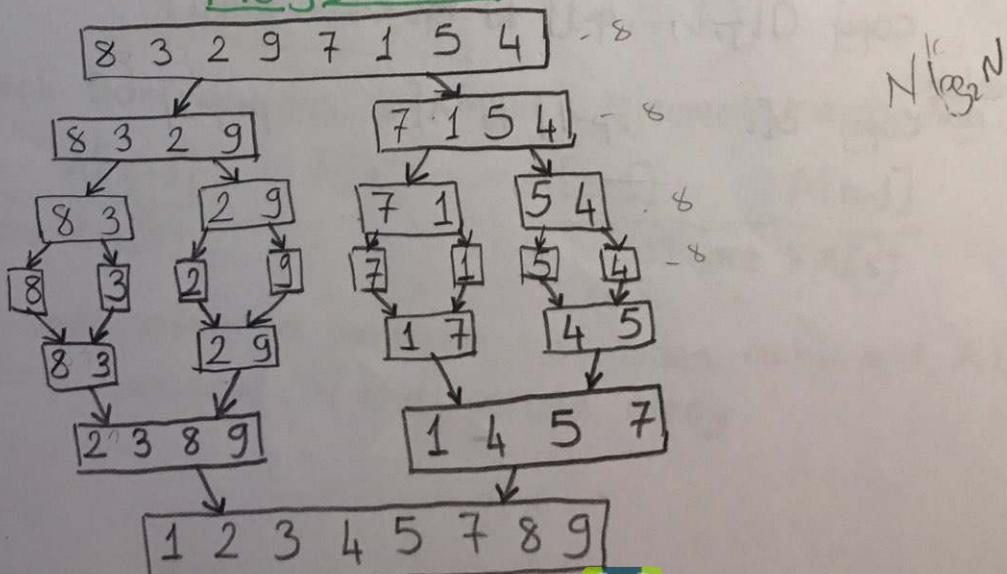
$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Example:  $A(n) = 2A(n/2) + 1$  if  $n > 1$

$a=2$   
 $b=2$   
 $d=0$

n'e bağlı değil  $\rightarrow A(n) \in \Theta(n^{\log_2 2}) = \Theta(n)$

## Merge Sort



(16)

### Pseudo Code

Mergesort( $A[0, \dots, n-1]$ )

if ( $n > 1$ )

copy  $A[0, \dots, \lfloor n/2 \rfloor + 1]$  to  $B[\emptyset, \dots, \lfloor n/2 \rfloor + 1]$

copy  $A[\lfloor n/2 \rfloor + 1, \dots, n-1]$  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( $B, C, A$ )

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

- 4. hafta / 18.10.2018

### Initial Condition

$C(1) = \emptyset$

Merge( $B[0, \dots, p-1], C[0, \dots, q-1], A[0, \dots, p+q+1]$ )

$i \leftarrow 0, j \leftarrow 0, k \leftarrow 0$

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 -

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

$$C_{\text{merge}}(n) = n-1$$

$$\text{Initial Condition: } C(1) = 0$$

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

- Backward Substitution -

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

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

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

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

$$= 8C(n/8) + 3n$$

$$= 2^i C(n/2^i) + i * n \quad \begin{matrix} \rightarrow n = 2^i \\ i = \log_2 n \end{matrix}$$

$$= nC(1) + \cancel{\log_2 n} * n$$

$$= n * \cancel{\log_2 n}$$

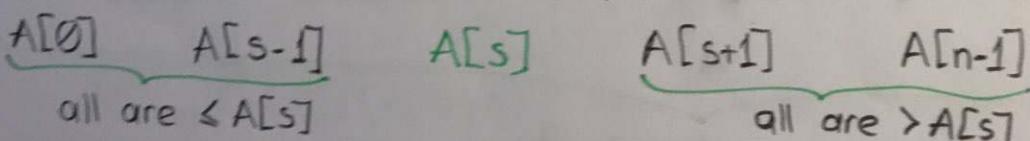
space complexity si  
digerlerine göre çok  
yüksek.

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

Principal Shortcoming: Linear amount of extra storage.

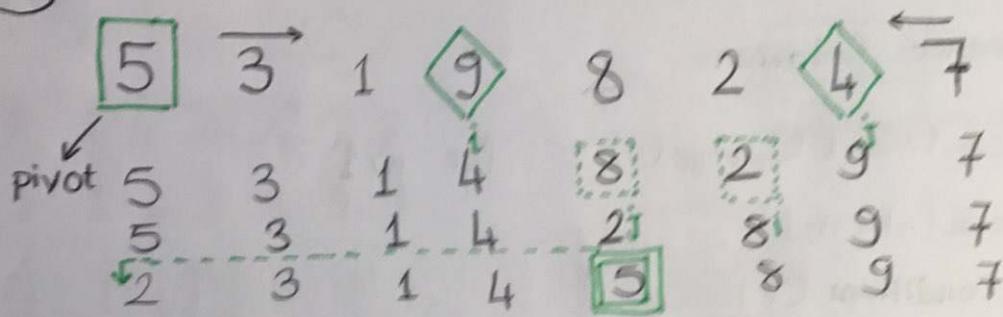
## Quick Sort

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

(18)



(a short example  
(whole example on p.19))

### Algorithms:

if  $l < r$

$S \leftarrow \text{Partition}(A[l, \dots, r])$

$\text{Quicksort}(A[l, \dots, s-1])$

$\text{Quicksort}(A[s+1, \dots, r])$

- ① Steps on encountering the first element greater than OR equal to the pivot : left-to-right scan
- ② Steps on encountering the first element smaller than OR equal to the pivot : right-to-left scan

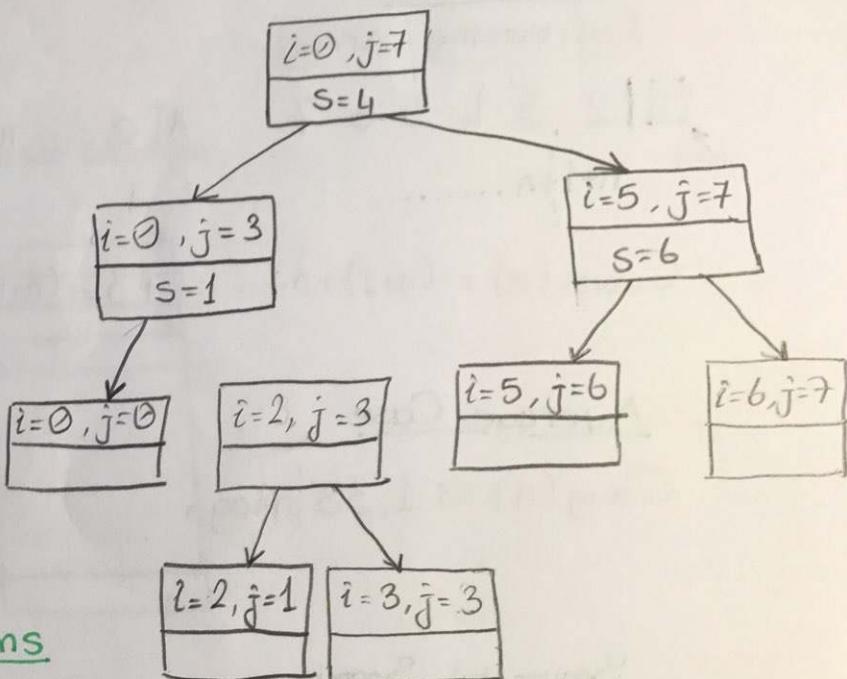
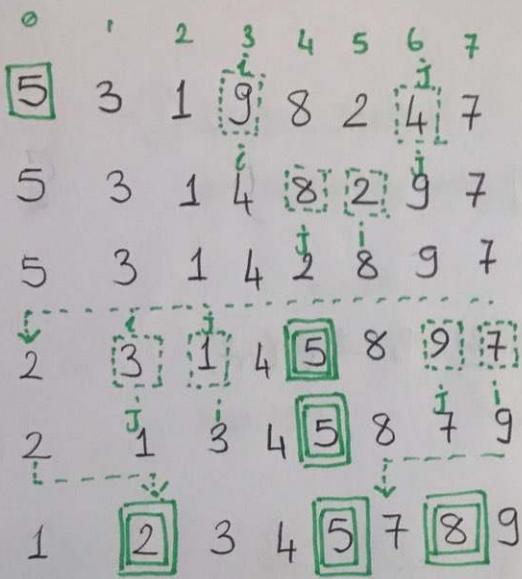
### Situations:

- 1 If scanning indices  $i$  and  $j$  have not crossed e.g.  $i < j$  we simply exchange  $A[i]$  and  $A[j]$  and resume the scans by incrementing  $i$  and decrementing  $j$  repeatedly.
- 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 searching indices stop while pointing to the same element i.e.  $i = j$  the value they are pointing to must be equal

Combine 2 and 3

Exchange  $A[j]$  with pivot whenever  $i > j$

(19)



## Partition Algorithms

```

 $p \leftarrow A[l]$ 
 $i \leftarrow l, j \leftarrow r+1$ 
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 least swap when  $i > j$ 
Swap ( $A[l], A[j]$ )
  
```

## Best Case

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

### Recurrence Relation

$$C_{\text{best}}(n) = 2C_{\text{best}}(n/2) + n$$

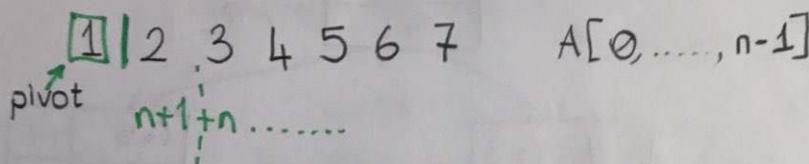
$$C_{\text{best}}(1) = 0$$

$$C_{\text{best}}(n) = \Theta(n \log n)$$

(20)

### Worst-Case

#### Increasing Array

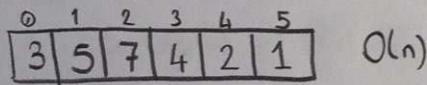


$$C_{\text{worst}}(n) = (n+1) + n + \dots + 3 = \frac{(n+1)(n+2)}{2} - 3 \Rightarrow \Theta(n^2)$$

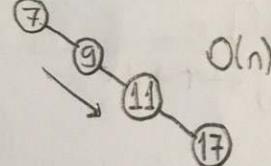
### Average Case

$$C_{\text{avg}}(n) \approx 1.38 n \log n$$

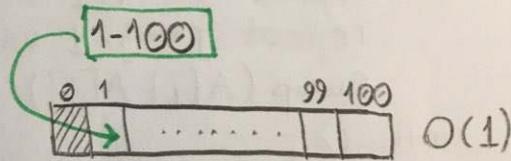
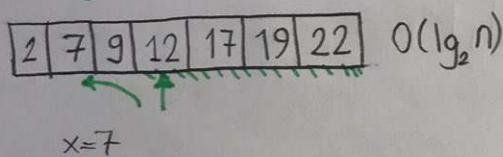
### Sequential Search



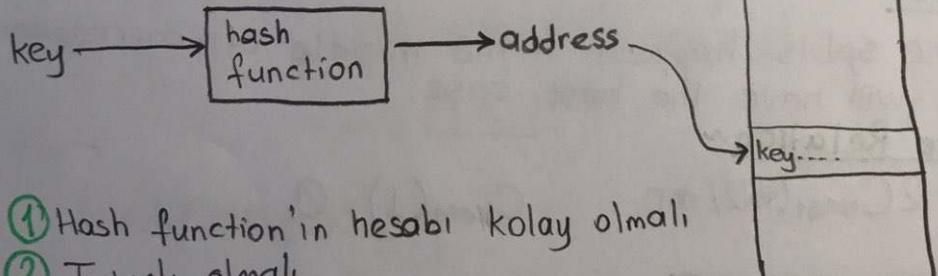
### BST



### Binary Search



### Hashing



- ① Hash function'in hesabı kolay olmalı
- ② Tutarlı olmalı
- ③ Adresin olabildiğince uniform dağılması sağlanmalı

key mod  $m$   
↑ tablo uzunluğu

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

$$\text{key} \rightarrow 1000, 10, 40, 50$$

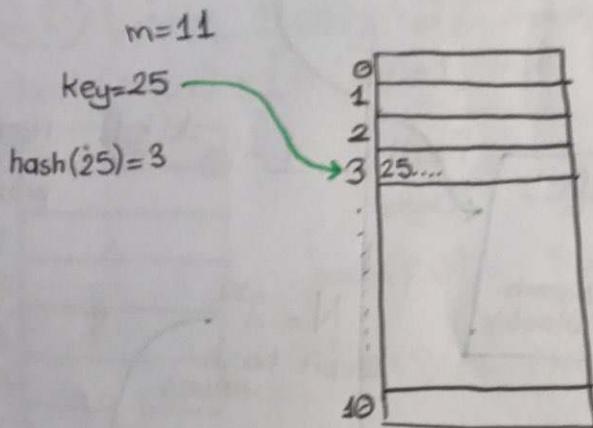
★  $m$ 'in asal sayı olması performansı artırır.

(21)

### ① Division Method

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

tablo uzunluğu



### ② Multiplication Method

$$A: \text{constant } 0 < A < 1 \quad A = \frac{\sqrt{5}-1}{2} = 0,618033$$

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

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

tablo  
uzunluğu

$$m=10.000 \quad k=123456$$

$$\begin{aligned} \text{hash}(123456) &= \lfloor 10.000 * (\underbrace{123456 * 0,618033}_{763.000,004151} \bmod 1) \rfloor \\ &= \lfloor 10.000 * 0,004151 \rfloor \\ &= \lfloor 41,51 \rfloor \end{aligned}$$

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

address



(22)

## Birthday Paradox

$$365 \rightarrow \frac{365-1}{365} * \frac{365-2}{365} * \dots * \frac{365-(k-1)}{365}$$

3 kişinin doğum  
günü farklı

table size: m

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

m tablo uzunlığında  
çakışmama olasılığı

$$1 - e^{-\frac{k(k-1)}{m}} : \text{Hash collision probability}$$

$$N = 2^{32}$$

32-bit hash values

### Key

- ① Integer
- ② Floating number
- ③ String

ikisi farklı  
değer  
döndürmeli  
"ELMA"  
"MALE"

→ Horner's Method:

$$\text{key} = R^{L-1} * \text{str}[0] + \dots + R^0 * \text{str}[L-1]$$

31  
(olarak)  
length of string  
Seçilir

Büyük harf ise:  
 $\text{str}[0] - 'A' + 1$

- ④ Compound key

31.08.1997

$$\hookrightarrow 31 * R + 8 * R^2 + 1997 * R^3$$

31



## Cözüm Problemine Çözüm Önerileri

### 1) Open Addressing

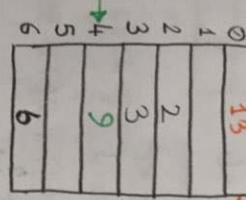
table size  $m \sim 2 * N$  olduğunda rahat delete işlemi sıkınlıktan gelenen sorunları解决 etmek için kullanılır.

#### 1.1 Linear Probing

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

$i = 0, \dots, m-1$

0	13
1	
2	2
3	3
4	9
5	
6	6



$m = 7$

$\text{key} = 9$

$$\text{hash}(13, 0) = 13 \bmod 7 = 6$$

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

$$= \frac{9}{7}$$

$$\text{hash}(9, 2) = (9+2) \bmod 7 = 4$$



$\text{key}$

$i = 0$   
 $h = \text{key} \bmod m$

$((i < m) \&\& (\text{hash}[h] == \text{key}))$

$\&\&$

$\text{hash}[h] \neq \text{NULL}$

$i++$

$$h = (h+1) \% m$$

(23)



Scanned by Photo Scanner

(24)

## 1.2 Quadratc Probing

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

Linear Probing'de her yerlesect eleman icin önegin 1er pozisyon durumu olabilin' Bunu önlemek icin arada bosluk bırakarak bir çözüm sunar.

0	
1	
2	
3	
4	
5	
6	9

$$\begin{aligned} h(9, 0) &= 9 \bmod 7 = 2 \\ h(9, 1) &= (2+1) \bmod 7 = 3 \\ h(9, 2) &= (2+2^2) \bmod 7 = 6 \\ h(k, i) &= (h'(k) + C_1 * i + C_2 * i^2) \bmod m \\ C_1 &= 2 \quad C_2 = 1 \end{aligned}$$

$$\begin{aligned} h(9, 1) &= (2 + 2 * 1 + 1 * 1) \bmod 7 = 5 \\ C_1 &= 2 \quad C_2 = 1 \\ i &= 1 \quad i = 1 \end{aligned}$$

## 1.3 Double Hashing

$$h_1(\text{key}, i) = [h_2(\text{key}) + i * h_2(\text{key})] \bmod m$$

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

table size

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

genelde asal

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

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

$$\text{Ex: } h_1(14) = ? \quad \rightarrow \quad h_1(14, 0) = h_1 \quad i=0 \quad h_1(14) = 14 \bmod 13 = 1$$

$i=1$

$$h_1(14, 1) = 1 + [1 + 14 \bmod 11] * 1 = 1 + 4 = 5$$

$i=2$

$$h_1(14, 2) = [1 + 4 * 2] \bmod m = 9$$

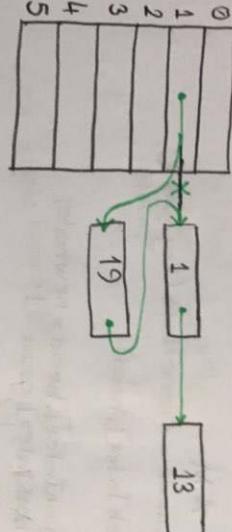
$$\begin{matrix} h_1(\text{key}) \\ h_2(\text{key}) \end{matrix}$$



Scanned by Photo Scanner

## 2 Separate Chaining

(25)



$M < N$   
Maximum  $\rightarrow M = N$

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

\* Statistical optimum value  $\rightarrow M \approx N/5$

key = 13 % 5  $\rightarrow$  1  
key = 1 % 6  $\rightarrow$  1  
key = 19 % 6  $\rightarrow$  1

```
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];  
};
```

```
hashTable = (struct hash*) calloc (M, sizeof (struct hash));  
if (hashTable == NULL)  
{  
    free (hashTable);  
    return 0;  
}
```

```
insertInHash (key, name);  
searchInHash (key);  
deleteFromHash (key);  
return 0;
```



(26)

```
Void inserttoHash (int key, char name [ ]) {
    int hashIndex;
    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++;
    return;
}

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

Void searchInHash (int key)
{
    int hashIndex = key % M, flag = 0;
    struct node *myNode;
    myNode = hashTable [hashIndex].head;
    if (!myNode)
        printf ("Search element unavailable in hash table\n");
    while (myNode != NULL && flag == 0)
        if (myNode->key == key)
            {
                printf ("%d\n", myNode->key);
                printf ("Name : %s\n", myNode->name);
                flag = 1;
            }
}
```



Scanned by Photo Scanner

```

myNode = myNode->next;
}
if (!flog)
printf("yok!");
return;
}

void deleteFromHash (int key)
{
    int hashIndex = key % M;
    struct node *temp, *myNode;
    myNode = hashTable[hashIndex].head;
    if (myNode)
    {
        printf("Given data is not present in hash Table!\n");
        return;
    }
    temp = myNode;
    while (myNode != NULL && flog != 1)
    {
        if (myNode->key == key)
        {
            flog = 1;
            if (myNode == hashTable[hashIndex].head)
                hashTable[hashIndex].head = myNode->next;
            else
                temp->next = myNode->next;
            hashTable[hashIndex].count--;
            free(myNode);
        }
        temp = myNode;
        myNode = myNode->next;
    }
    if (flog)
        printf("Data deleted successfully from Hash Table!\n");
    else
        printf("Given data is not presented in hash Table\n");
}

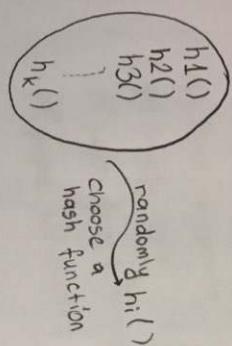
```



(28)

### [3] Universal Hashing

15, 22, 29 → hash → 1



- Kullanılan Hash Fonksiyonları -

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

a, b, p → chosen randomly

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

Table size is not random

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

open addressing : M > N  
separate chaining : M < N

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

open addressing linear probe

$$\alpha = \frac{\# \text{ of probe}}{M} \rightarrow \frac{1}{10} \rightarrow \frac{N}{M}$$

$$\begin{array}{l} \alpha \rightarrow \# \text{ of probe} \\ 0,1 \rightarrow 1,11 \\ 0,2 \rightarrow 1,28 \\ 0,3 \rightarrow 1,52 \\ 0,4 \rightarrow 1,89 \\ 0,5 \rightarrow 2,5 \\ 0,6 \rightarrow 3,62 \\ 0,8 \rightarrow 13,00 \\ 0,9 \rightarrow 50,50 \end{array}$$

( $\frac{1}{2}$ ) ✓



Scanned by Photo Scanner

### Unsuccessful Search

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

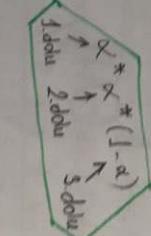
$P(\text{empty cell}) = (1-\alpha)$

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

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

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

Average # of steps:



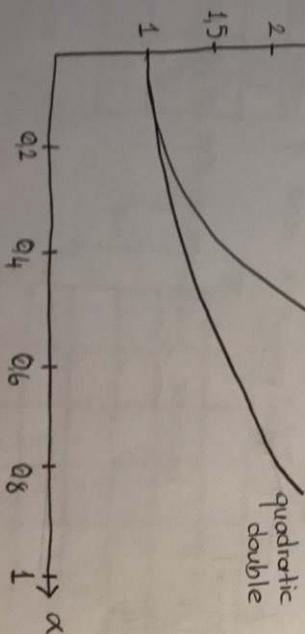
$$\sum_{k=1}^m k * \alpha^{k-1} * (1-\alpha) \Rightarrow \text{geometric series} = \frac{1}{1-\alpha}$$

Expected number of probes  
in unsuccessful search.

$$\text{insertion expected # of probes} \rightarrow \frac{1 + \frac{1}{(1-\alpha)^2}}{2}$$

Average # of probes  
successful search  
linear probe

quadratic  
double

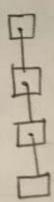


30

### Separate Chaining

#### ① Unsuccessful search

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



Worst case: Bitün elemalar aynı adreste.  
Average case:  $\alpha = \frac{N}{M}$

Expected length of a probe  $\Rightarrow \alpha = \frac{N}{M}$

$$\Theta(1 + \alpha) \xrightarrow{\text{search}} \Theta(1 + \alpha) = \Theta(1)$$

↓  
hash function

$$\alpha = \frac{N}{M} \Rightarrow \text{sabit}$$

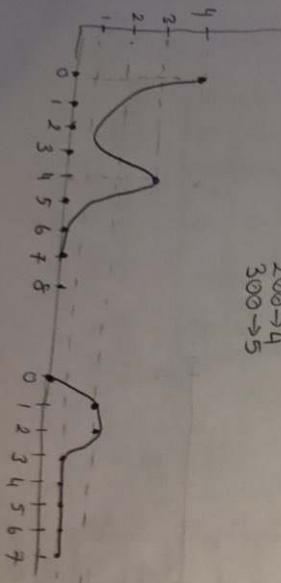
keys : 64, 100, 128, 200, 300, 400, 500

table size = 8  $\rightarrow$   $\text{hash(key)} = x \bmod 8$

$$\begin{array}{ll} 64, 128, 200, 400 \rightarrow 0 & \text{because } \gcd(64, 100, 128, 200, 300, \\ 100, 300, 500) = 4 & 400, 500, 8) = 4 \end{array}$$

table size = 7  $\rightarrow \text{hash(key)} = x \bmod 7$

$$\begin{array}{ll} 64, 100 \rightarrow 1 & \text{because } \gcd(64, 100, 128, 200, 300, 400, 500, 7) = \frac{1}{7} \\ 100, 128 \rightarrow 2 & \\ 300 \rightarrow 3 & \\ 200 \rightarrow 4 & \\ 300 \rightarrow 5 & \end{array}$$



## Dynamic Programming

Planning Richard Bellman (1950's)

Example Fibonacci Numbers

0 1 1 2 3 5

int

fibonacci(int n)

```

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

```

Recurrence Relation:

$$f_{\text{fibonacci}}(n) = f_{\text{fibonacci}}(n-1) + f_{\text{fibonacci}}(n-2) \quad (1, 6)^n$$

Summation  
cok fazla tekrar  
lenen hepsi var!

0	1	1	2	3	5	6	7	8
fibonacci								

(\*) Hesaplamaları tek tek yeniden değil, bu direkti gözde daha önceki  
toplamları değerleri kullanıyor.

abandon  
select

### Example 0-1 Knapsack Problem

$W$  → weight capacity of bag

Brute force:  $2^n$

Ex:

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

item	1	2	3	4	5
0					
1					
2					
3					
4					
5					

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

maximize  $\sum v_j$   
subject to  $w$   
maximum  
kazanç

(32)

case 1: thief takes item  $i$

$$\begin{aligned} P(i, w) &= P(i-1, w-w_i) + \sqrt{w} \\ i=3, w_i=2 & \quad \xrightarrow{2+10} \\ P(3, 4) &= P(2, 2) + \sqrt{3} \\ &\quad \text{4 kilo gelindiğinde} \\ &\quad \text{maximum kazanç} \end{aligned}$$

esya  
3 kg'a kadar

case 2: not takes item  $i$

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

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

Initialization → $P[0, w] = 0$ , $P[i, 0] = 0$				
0	1	2	3	4
0	0	0	0	0
1	0	0	12	12
2	0	0	22	22
3	0	12	30	32
4	0	10	25	30

$\xrightarrow{\text{maximum kazanç}}$

$w_4 = 2$   
 $P[4, 1] = P[3, 1] + 15, P[3, 2] = 15$   
 $P[4, 2] = \max \left\{ P[3, 2] + 15, \frac{12}{2} \right\} = 25$   
 $P[4, 3] = \max \left\{ P[3, 3] + 15, \frac{10}{2} \right\} = 30$

$w_3 = 3$   
 $P[3, 1] = P[2, 1] + 10, P[2, 2] = 22$   
 $P[3, 2] = \max \left\{ P[2, 2] + 10, \frac{22}{2} \right\} = 30$   
 $P[3, 3] = \max \left\{ P[2, 3] + 10, \frac{12}{2} \right\} = 30$   
 $P[3, 4] = \max \left\{ P[2, 4] + 10, \frac{10}{2} \right\} = 22$   
 $P[3, 5] = \max \left\{ P[2, 5] + 10, \frac{10}{2} \right\} = 22$

Ins

Sort Link



Scanned by Photo Scanner

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

Reconstructing Optimal Solution  
 $\rho[i,j] = \rho[i-1,j]$

$$\rho[i,j] \rightarrow \rho[i-1,j-w_i]$$

not taken ↑  
 $w_1=2$

taken ←  
 $w_2=1$

$w_3=3$

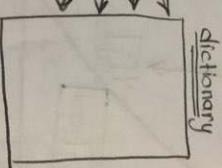
$w_4=2$

Hangi itemler alınmış?

### Edit Distance

Sunny

(yanlış  
yazılmış  
kelime)



### The Levenshtein Algorithm (Edit Distance)

Sunny → sunny

Sunny → sun-y

Sunny → Sunnny

Yapılabilecek olası hatalar

dist(MATHS, ARTS) → MATHS

ARTS

MATHS  
ARTS

(33)



Scanned by Photo Scanner

(34)

$X[1, \dots, m] \leftarrow y[1, \dots, n]$

① copy

② change

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

$y[j]$

insert

change

delete

MATHS

ARTS

:Ex

③ insert

④ delete

MATHS

ARTS

change  
insert  
change

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

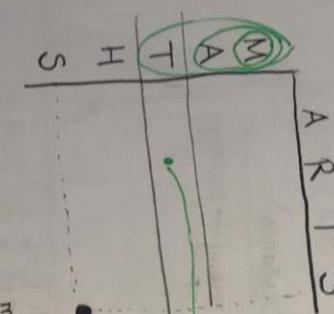
$y[j]$

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

$x[i]$

$y[j]$

$ED[i,j]: y[j]$  gelene kadarki  
minimum mesafe



minimum  
distance

### Recurrence Relation

$$ED[i,j] = \begin{cases} ED[i-1,j-1] + 0 & : \text{copy} \\ ED[i-1,j-1] + cost & : \text{change} \\ \min \{ ED[i-1,j] + cost, \text{insert} \} & : \text{otherwise} \\ ED[i,j-1] + cost & : \text{delete} \end{cases}$$



Scanned by Photo Scanner

(35)

Initialize:  $ED[i, 0] = i$ ,  $ED[0, j] = j$

$\begin{array}{c} M \\ A \\ T \\ H \\ S \end{array}$

$\begin{array}{cccc} & 1 & 2 & 3 & 4 \\ R & \downarrow & \downarrow & \downarrow & \downarrow \\ & 1 & 2 & 3 & 4 \\ T & \downarrow & \downarrow & \downarrow & \downarrow \\ & 1 & 2 & 3 & 4 \\ S & \downarrow & \downarrow & \downarrow & \downarrow \end{array}$

$\rightarrow$  sonlu türün her<sup>1</sup>'ler  
sorusunu cevabi bulunulabilir

MATHS ne oldu da ARTS oldu  
MATHS ne oldu da ARTS oldu

1. deneme: MAT HS  
→ ARTS  
→ ARTS

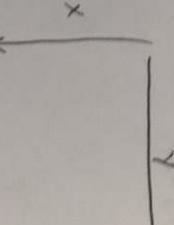
2. deneme: MAT HS  
→ ARTS  
→ ARTS

Sonki tüm harfler  
ler eklenmiş gibi  
düşün

minimum  
distance

### Longest Common Subsequence

X: BACDB } LCS: BCB  
Y: BD CAB }



$$LCS[i, j] = \begin{cases} LCS[i-1, j-1] + 1, & \text{if } X[i] = Y[j] \\ \max \left\{ \begin{array}{l} LCS[i-1, j], \\ LCS[i, j-1] \end{array} \right\}, & \text{otherwise} \end{cases}$$

8. Hafta

$\langle A, C, G, T \rangle$

X: ATCTGAT } Longest Common Subsequence of X and Y  
Y: TGCATA }

$$\begin{aligned} LCS(X, Y) &= TCTA \\ LCS(X, Y) &= TGAT \end{aligned}$$

Subsequence  
K O Z A L A K

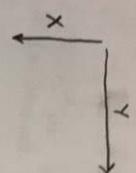
K A L  
K O L  
K O L A  
K A L K



(36)

$$\begin{aligned} X &= \langle x_1, x_2, \dots, x_m \rangle \\ Y &= \langle y_1, y_2, \dots, y_n \rangle \end{aligned} \quad \text{LCS}$$

$$\begin{aligned} S &= \langle s_1, s_2, \dots, s_k \rangle \\ S_{i_1}, S_{i_2}, \dots, S_{i_k} \\ i_1 < i_2 < \dots < i_k \end{aligned}$$



Önek:  $X: \langle BACDB \rangle$

$X_i \quad Y_j \rightarrow C[i,j]$  → 0 noktaya kadar  
ontok ekleme n  
uqunluğunu

		B		C		D		A		B	
		B		C		D		A		B	
		0	1	0	1	0	1	0	1	0	1
X	↓	0	1	0	1	0	1	0	1	0	1
A		0	1	0	1	0	1	0	1	0	1
C		0	1	1	2	1	2	1	2	1	2
D		0	1	2	2	1	2	1	2	1	2
B		0	1	2	2	3					

Hangi harfleri alındığında dair  
[M,N] gelenen ikerlemeye başladığında  
harf ortak olarak alındı m'ı alınmadı m'  
belli olmugor karneşa olyor.

ÇÖZÜM

Yeni bir Matris

ADD:0  
SKIPX:1  
SKIPPY:2

$x_i = y_j \rightarrow B[i,j] = \text{ADD}$   
 $x_i \neq y_j \rightarrow C[i-1,j] > C[i,j-1] \text{ then}$   
     $x[i] \text{ not in LCS}$   
     $B[i,j] = \text{SKIPX}$

else

$B[i,j] = \text{SKIPPY}$   
 $y[j] \text{ is not in LCS}$

BCB



Scanned by Photo Scanner

### Longest Common Subsequence

```
while ((i!=0) && (j!=0))  
{  
    switch (b[i][j])  
    {  
        case ADD:  
            add x[i] to front of LCS  
            i--;  
            j--;  
            break;  
        case SKIP:  
            i--;  
            break;  
        case SKIPY:  
            j--;  
            break;  
    }  
}
```

(37)



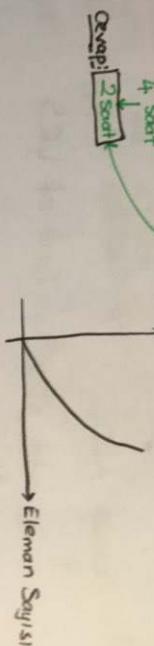
Scanned by Photo Scanner

(38)

- 4. Vize Cevapları -

$$5/1 \text{ Çalışma zamanı : } \cancel{N^2} + 13N^{4/3} + 12N + 3$$

$$\begin{aligned} N &\rightarrow 1 \text{ saat} \\ 2N &\rightarrow ? \quad (\text{Bellek ve çalışma zamanının hızı} \rightarrow \times 2) \\ \text{Avg: } & \boxed{2 \text{ saat}} \end{aligned}$$



5/2  $3n^3 + 20n^2 + 5 \rightarrow O(n^3)$  olduğunu gösteriniz.

$3n^3 + 20n^2 + 5 \rightarrow O(n^3)$  → BÖYLE SAĞMALANMIŞ

$$\begin{aligned} f(n) &\leq c g(n) \\ 3n^3 + 20n^2 + 5 &\leq cn^3 \quad n > n_0, \quad c > 0 \\ n=1 & \\ c=28 & \end{aligned}$$

$$\begin{aligned} 3+20+5 &\leq 28 \\ 28 &\leq 28 \end{aligned}$$

10/3 İki algoritmanın performans karşılaştırması kılası.

Time complexity, Space complexity

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

- ↳ Theoretical Analysis
- ↳ Empirical Analysis (Girdiği artırıp denemeler yapmak)

15/4

10/5

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



Scanned by Photo Scanner

6/⑥

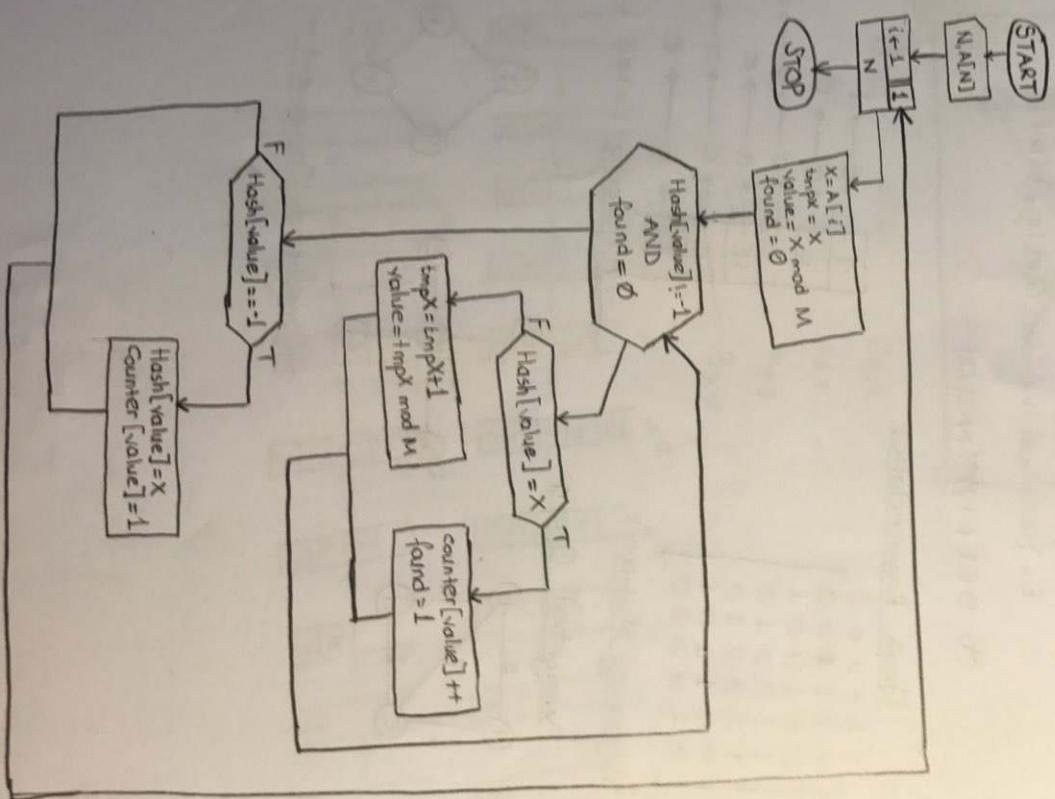
(A: ifode 2.ye göre)

$$\begin{aligned}
 n(n+5) &\leq 5000n^2 \rightarrow E \\
 2^{n+1} &\leq 2^{n+1} \rightarrow E \\
 \log(n) &\leq \ln(n) \rightarrow E \\
 1000n^2 &\leq 1000n^3 \rightarrow K \\
 (n-5)! &\leq 2^n \rightarrow B \\
 (10n+5)\log n &\leq 100n + (10n)\log n \rightarrow E
 \end{aligned}$$

③9

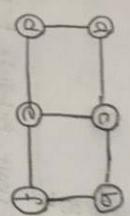
25/⑦

(E: exit, B: bulgın, K: hizla)

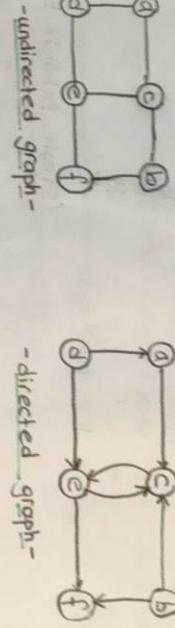


(40)

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



Can be graph  
-undirected graph-



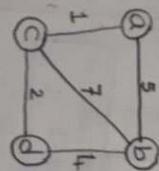
$V = \{a, b, c, d, e, f\}$   
 $E = \{(a,c), (a,d), (b,c), (d,e), (c,e), (e,f), (b,f)\}$

\*  $0 \leq E \leq |V| * (|V|-1)/2$

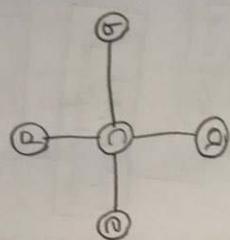
Graph Representation

$$\begin{bmatrix} a & b & c & d & e & f \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ f & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

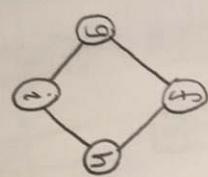
Adjacency Matrix



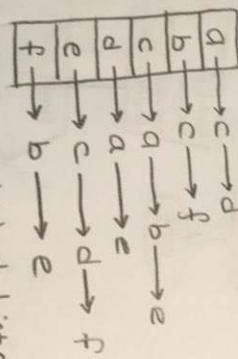
Weighted Graph



Connected Component



Adjacency Linked List



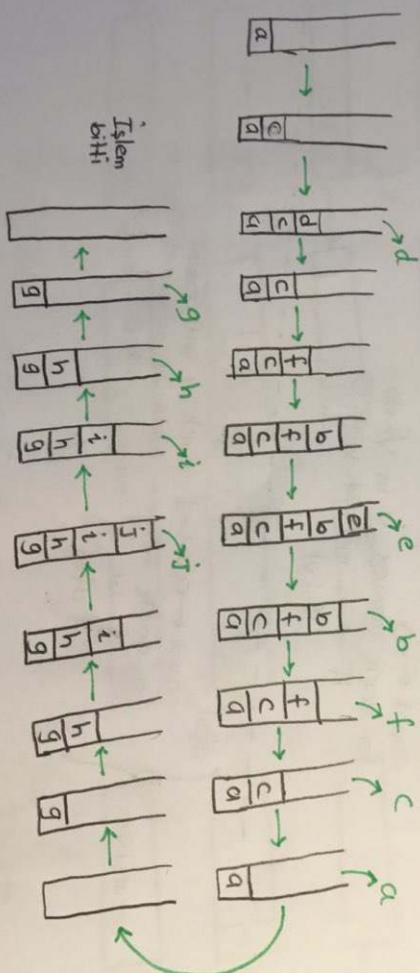
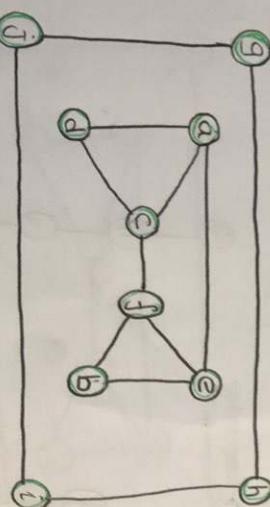
Scanned by Photo Scanner

## Depth-First-Search

Decrease-by-one technique

LIFO  
STACK kullanılır.  
(push & pop yapılacak)

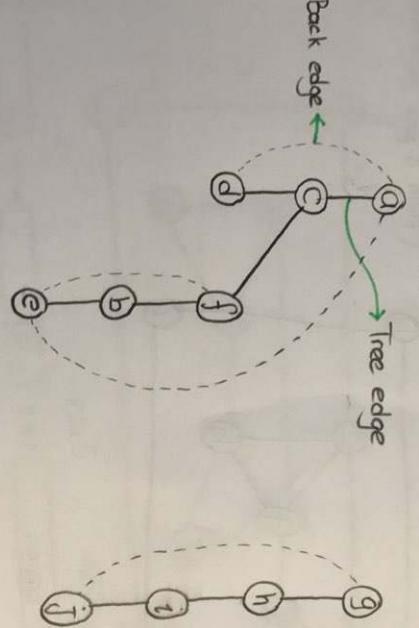
41



(12)

### Dept-First-Search Forest

Tree edge  
Back edge



### DFS( $G$ )

mark each vertex in  $V$  with  $\emptyset$  as a mark of being unvisited

count  $\leftarrow 0$

for each vertex  $v$  in  $V$  do

if  $v$  is marked with  $\emptyset$

dfs( $v$ )

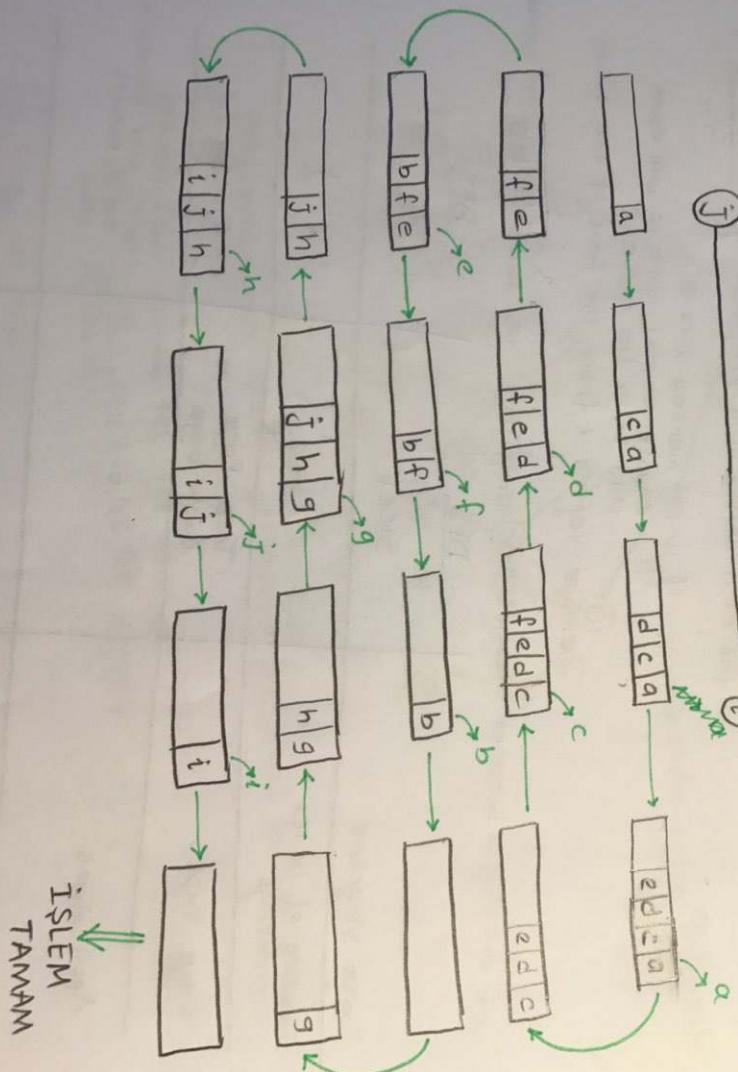
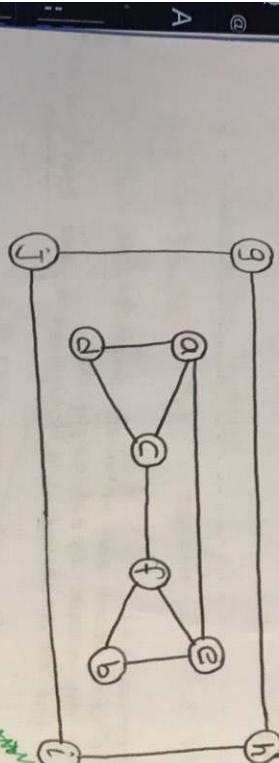
{  
dfs( $v$ )  
count  $\leftarrow$  count + 1, mark  $v$  with count  
recursive  
for each vertex  $w$  in  $V$  adjacent to  $v$   
if  $w$  is marked with  $\emptyset$   
dfs( $w$ )  
}

(43)

### Breadth-First-Search

cesur / brave → depth  
tedbirli / cautious → breadth

FIFO  
QUEUE (queue & enqueue yapılımları)  
a<sub>1</sub>, c<sub>2</sub>, d<sub>3</sub>, e<sub>4</sub>, f<sub>5</sub>, b<sub>6</sub>  
g<sub>7</sub>, h<sub>8</sub>, j<sub>9</sub>, i<sub>10</sub>



Scanned by Photo Scanner

(44)

### BFS(G)

mark each vertex in  $V$  with  $\emptyset$  as a mark of being unvisited  
 $count \leftarrow 0$   
 for each vertex  $v$  in  $V$  do  
     if  $v$  is marked with  $\emptyset$   
         bfs( $v$ )

bfs( $v$ )

$count \leftarrow count + 1$   
     mark  $v$  with  $count$  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  
             if  $w$  is marked with  $\emptyset$   
                  $count \leftarrow count + 1$ , mark  $w$  with  $count$   
                 add  $w$  to the queue

remove vertex  $v$  from the front of the queue

web crawling  $\rightarrow$  internet sitesinde link içinde link ve buradaki bilgileri okume

DFS

Stack

Queue

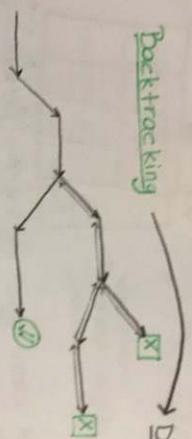
Data structure	Stack	Queue
Number of vertex orderings	2	1
Edge types	Tree Edge Back Edge	Tree Edge Cross Edge
Applications	(Belirli sınırlar içermek için) Web crawling Social network Map	
Efficiency for adjacent matrix	$\mathcal{O}( V ^2)$	$\mathcal{O}( V ^2)$
Efficiency for adjacent linked-list	$\mathcal{O}( V  +  E )$	$\mathcal{O}( V  +  E )$



(45)

### Backtracking

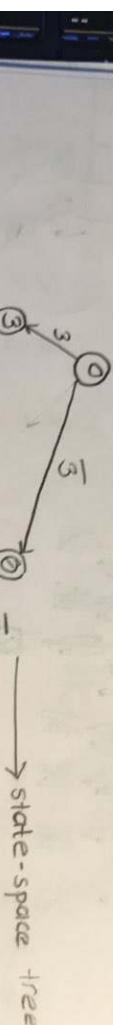
DFS with pruning



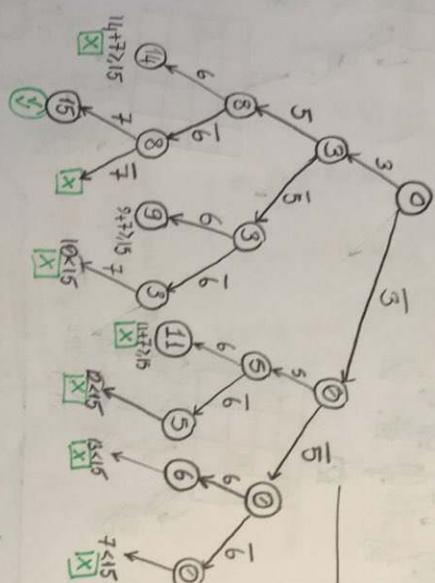
### Subset Sum Problem

$S = \{S_1, S_2, \dots, S_n\}$  toplam  $d$  olan alt kümeler? ( $S_1 \cup S_2 \cup S_3 \cup \dots \cup S_d$ )

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



Scanned by Photo Scanner



### 8-vezir problemi (n-queen problem)

Sırańt tahtasına birbirini görmeyecek(gönenyecek) şekilde 8 vezirini yerlestirme.

① Her bir durumu tek tek denemek

$$\binom{64}{8} = \frac{64!}{8! \cdot (64-8)!} = 4.426.165.368 \text{ deneme}$$

② One per row

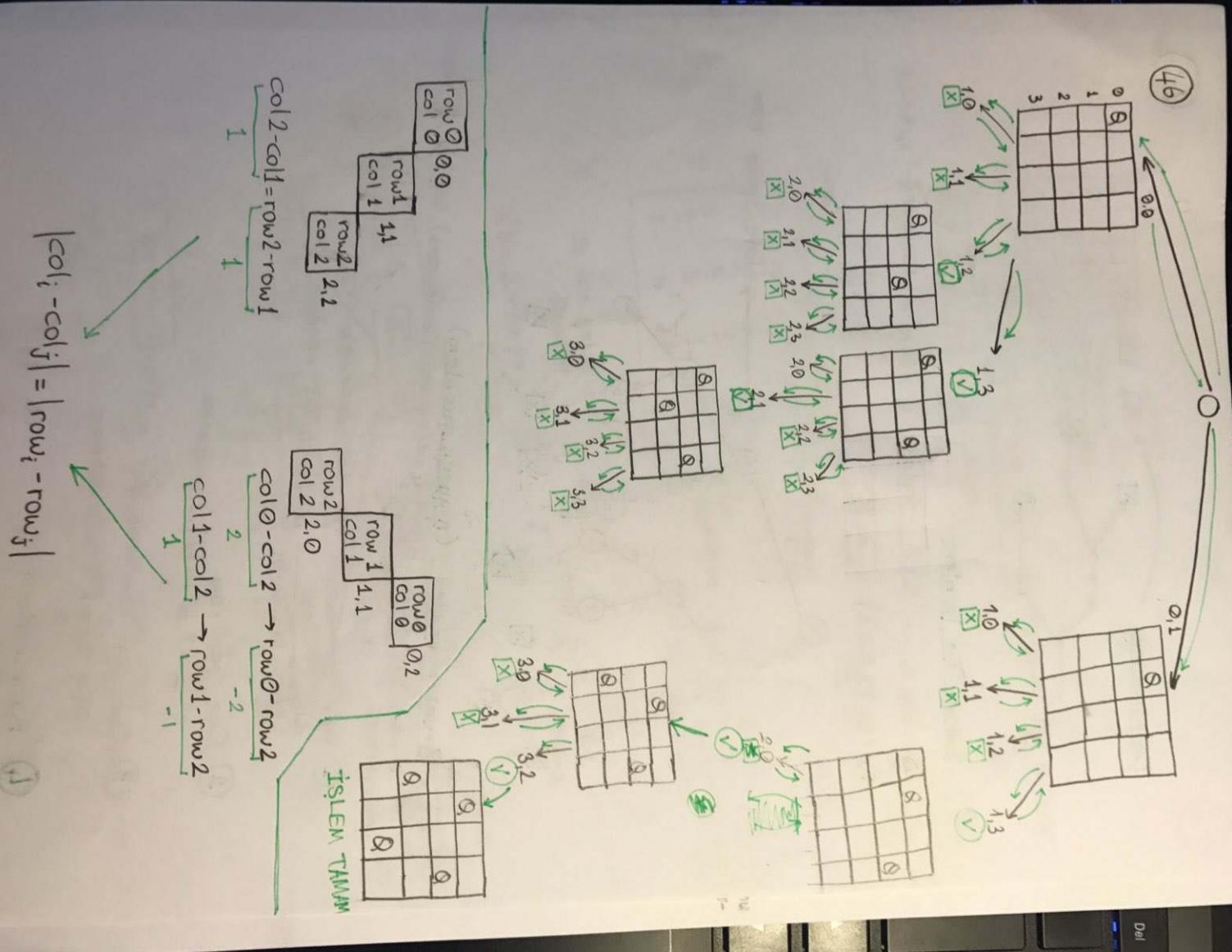
$$n^n = 8^8 = 16777.216 \text{ deneme}$$

③ One per row and per column

$$n! = 8! = 40.320 \text{ deneme}$$

④ Backtracking

2047 deneme



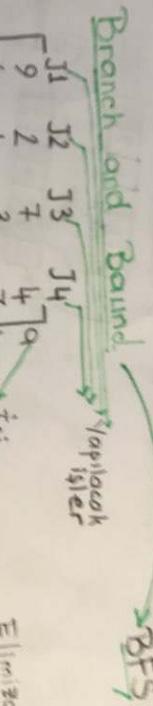
1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

→ Vezirleri  
yerlestirdigim  
matris

\* Ekstra bir dizisi hangi vezir hangi satır (o satır veziri) sütündə

0	1	2	3
2	4	1	3

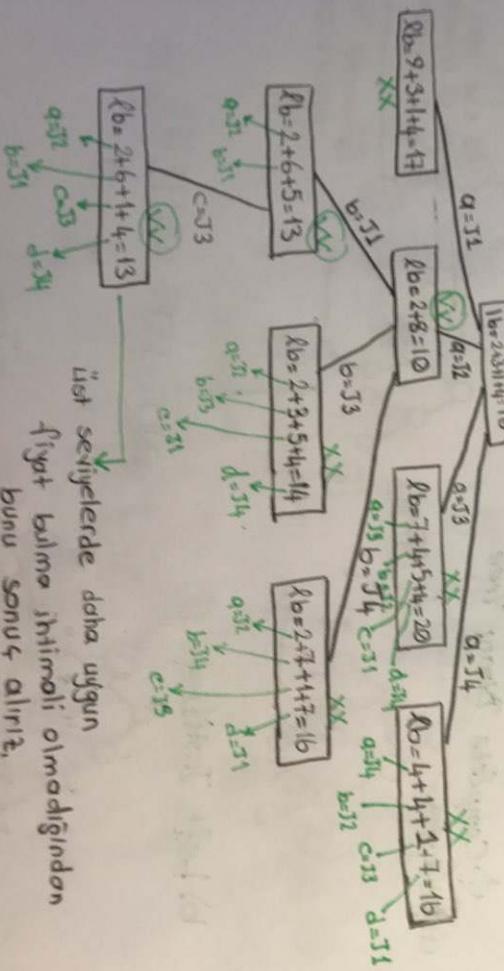
← Hangi



Eliminədə öz paro şəkərini  
yapılacak işler yan şunu  
en az mənalı ilə yopmuş  
ya çalışırıv.

İş için teklif edilen  
fiyatlar

$$Q_b = 2 + 3 + 1 + 4 = 10$$



Üst seviyelerde daha uygun  
fiyat bulma ihtiyaci olmadından  
buunu sonuç alırız.



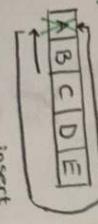
Scanned by Photo Scanner

48

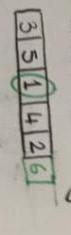
### Priority Queue

First-in-first-out

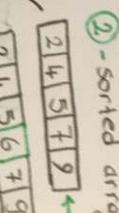
A  $\leftarrow$  B  $\leftarrow$  C  $\leftarrow$  D



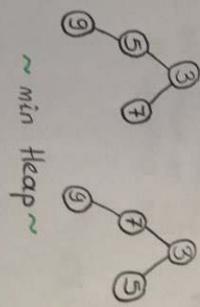
① - unsorted array -  
- insert O(1)  
- remove O(n)



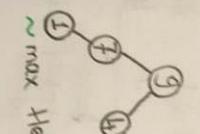
② - sorted array -  
- insert O(1)  
- remove O(n)



③ - linked circular array -  
- insert O(n)  
- remove O(n)



~ min heap ~



~ max heap ~

### - Heap Tree -

◊ Complete Binary Tree  
balanced

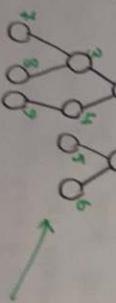
$\rightarrow$  depth 0  
 $\rightarrow$  n-2  
 $\rightarrow$  n-1  
 $\rightarrow$  n

$\rightarrow$  0  
 $\rightarrow$  n-2  
 $\rightarrow$  n-1  
 $\rightarrow$  n

All the nodes at  
depth 0 through n-2  
have 2 children!  
Balanced Tree  
condition

### b) Left Justified

NOT LEFT-JUSTIFIED



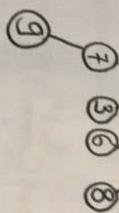
left-justified



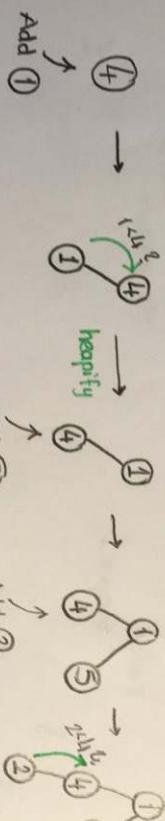
Scanned by Photo Scanner

② a parent's value  $\leq$  its children's values ; min Heap

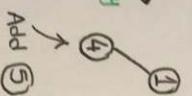
(49)



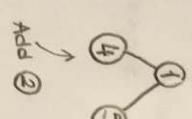
Heapify



heapify



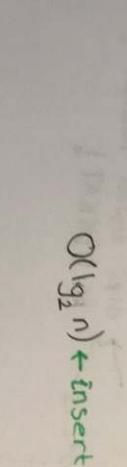
heapify



$O(\lg n) \leftarrow$  insert

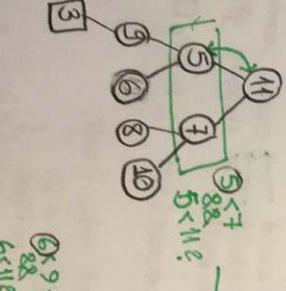
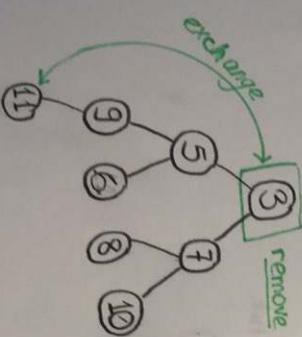
UPHEAP

Heapify



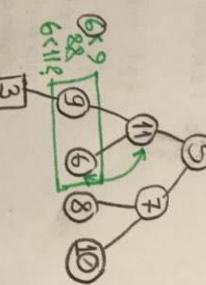
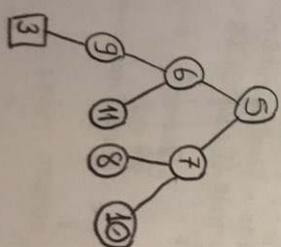
$O(\lg n) \leftarrow$  remove

remove



$O(\lg n) \leftarrow$  remove

DOWNSHIFT



$O(\lg n) \leftarrow$  insert

UPHEAP



Scanned by Photo Scanner

(50)

```
void insert(int newItem, int n, int pQueue[])
{
    int i, stop = 0, parent;
    pQueue[n] = newItem;
    i = n;
    n++;
    while ((i > 0) && (!stop))
    {
        parent = (i - 1) / 2;
        if (pQueue[i] < pQueue[parent])
        {
            pQueue[i] ↔ pQueue[parent];
            i = parent;
        }
        else
            stop = 1;
    }
}

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

① 2 child  
② 1 child  
③ no child

Olası senaryolar  
Hepsini göz önünde tut.

i < n  
i = 2\*i + 1

dizi dışına taşıabilir.  
DİKKAT!



Scanned by Photo Scanner

```
int findSmallChild(int i, int pQueue[], int n)
```

```
{
```

```
if (2*i+2 < n) → 2 child
```

```
{
```

```
if (pQueue[2*i+1] < pQueue[2*i+2])
```

```
return 2*i+1;
```

```
else return 2*i+2;
```

```
} else if (2*i+1 < n) → 1 child
```

```
return 2*i+1;
```

```
else → no child
```

```
{
```

(51)



Scanned by Photo Scanner