ALGORİTMA ANALİZİ

13:15 başlangıç

-Mine Elif KARSLIGIL-

Kaynaklar

The Design & Analysis of Algorithm, A. Levitin → baş ucu kitabı gibi (hemen her temel konu var.)
Algorithms : 4th Edition, R. Sedgewick, K. Wayne → Princeton Uni. hocaları yazmış. (Hoca en çok bundan yararlanıyor)
The Algorithm Design Manual, S. Skiena → Hikaye kitabı gibi (Biraz Advanced)

Değerlendirme

- 3 Ödev → %15
- Proje → %10
- 2 Midterm → %45
- Final → %30

* Ödevlerde kullanılacak programlama dili C (ödeve göre değişebilir.)

eleman sayısı
elemanların veriliş sırası

- Best case
- Average case : en mantıklısı (ama hesabı 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)

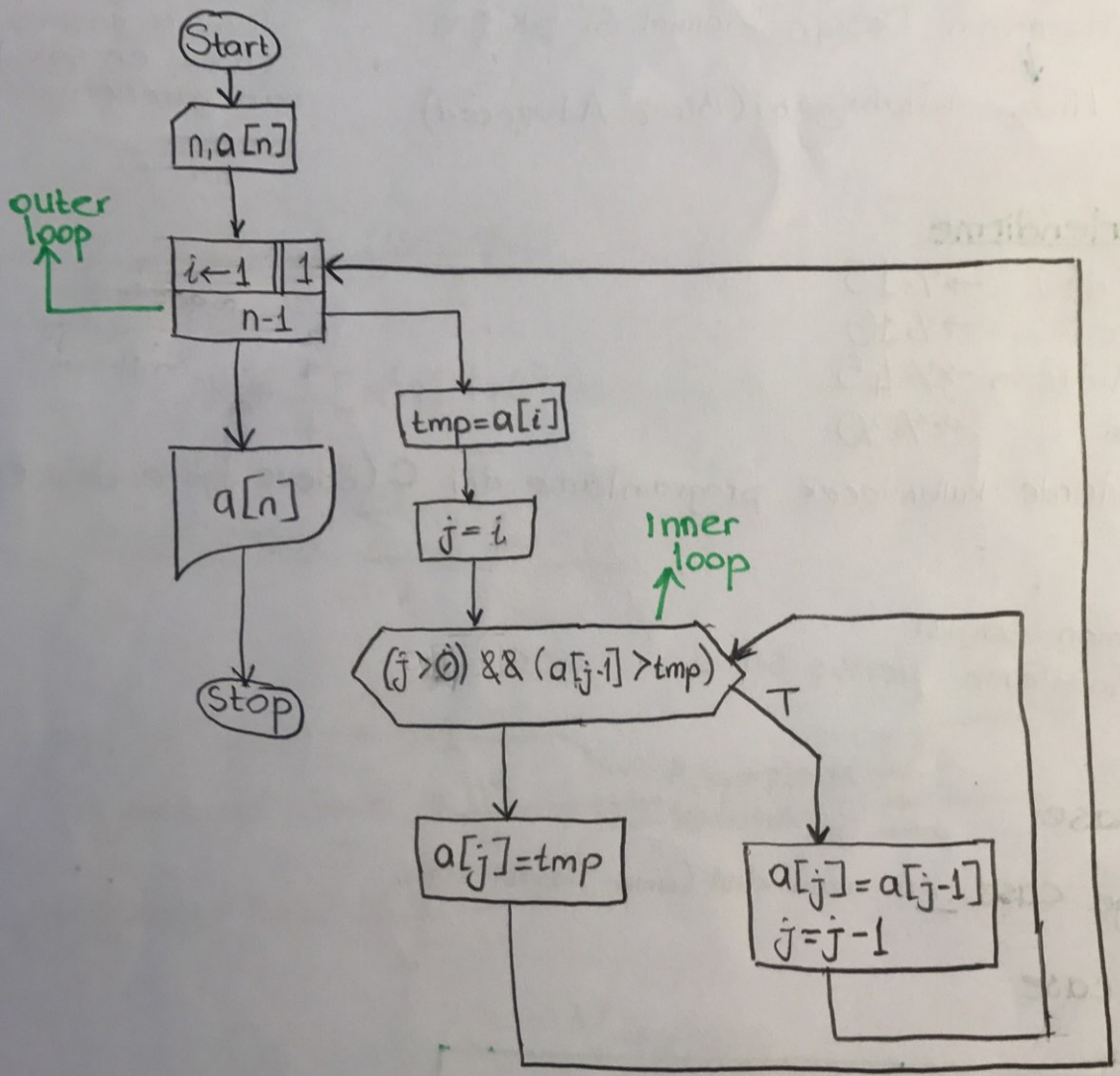
insertion sort

21	17	14	34	47	19	71	22	29	41	32	8

17 21 14

14 17 17 21 34 47 (19) 71 22 29 41 32 8

Sıralı | Sırasız

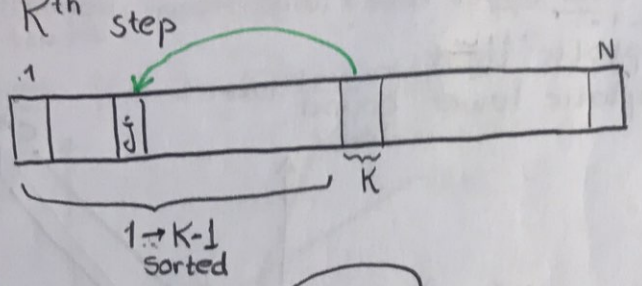


worst case: outer loop : 1 ... n-1

inner loop : j=1, j=2, j=3, ..., j=n-1 } n-1

$$\rightarrow \sum_{i=1}^{n-1} \sum_{j=1}^{i-1} 1 = \sum_{j=1}^{n-1} i = \frac{n*(n-1)}{2} \approx n^2$$

Avarage case:



$K-j+1$

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. gözdeki eleman için iç döngüye girme miktarı

outer loop

$$\sum_{K=2}^{N}\frac{K+1}{2}=\frac{1}{2}\cdot\frac{(N+1)(N+2)}{2}$$

$$=\frac{1}{4}[N^2+3N+4]\Rightarrow \text{average case}$$

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

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

→ Asymptotic Notation

$n \rightarrow O(n)$

$n^2 \rightarrow O(n^2)$

$n^2 \rightarrow O(n^2)$

Big-OH

Asymptotic Notation

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

katsayılar elenir

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

Algoritmanın karmaşıklık mertebesi

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

④

Asymptotic Notation

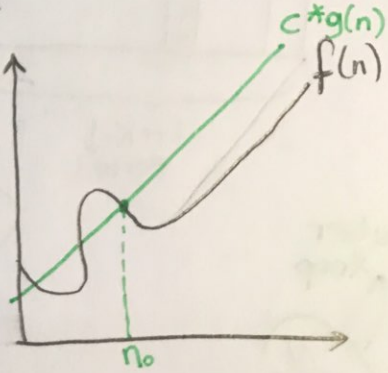
Big-Oh: O → asymptotic upper bound (algoritmanın karmaşıklığında en üst nokta)

Big-theta: Θ → asymptotic tight

Big-omega: Ω → asymptotic lower bound

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

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



Örnek: $f(n) = 2n+10$
$g(n) = n$?

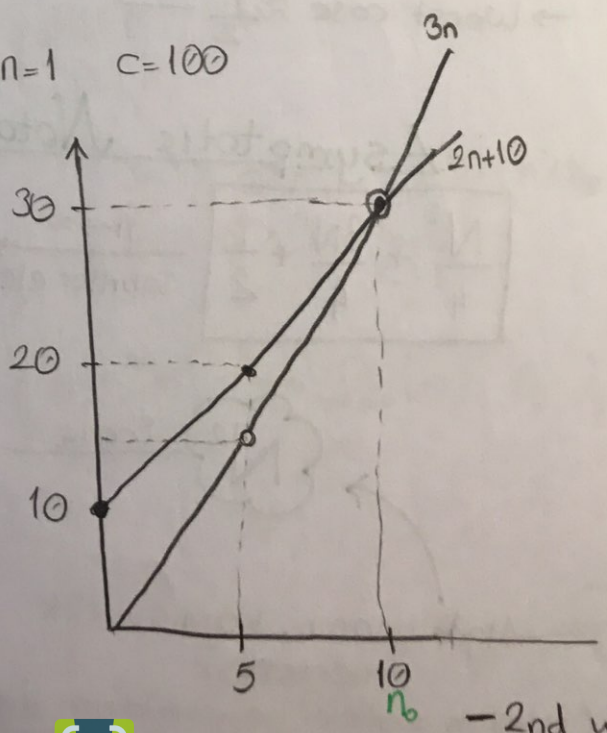
$2n+10 \leq c*n$ $\quad n \geq n_0$
$10 \leq (c-2)*n$ $\quad c=3$ $\quad n=10$
$f(n) = 2n+10 \longrightarrow O(n)$
$g(n)$

Örnek: $f(n) = 2n+10$
$g(n) = n^2$

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

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

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

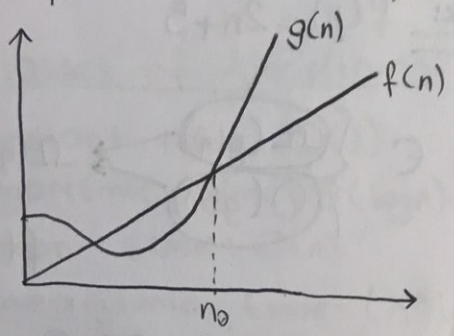


– 2nd week

Big-Oh Notation

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

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



$2n+10 \le c*n$ $\quad n \ge n_0$

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

$10 \le c-2 \longrightarrow 12 \le c$

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

$2n+10 \le c*n^2$ $\quad n > n_0 > 0$ $\quad n=3$

$16 \le c*9$ $\quad c=2$, $n=3$

Örnek: $f(n^2) \in O(n)$ ($g(n)$)

$n^2 \le c*n$

$n \le c$

Big-Omega Notation

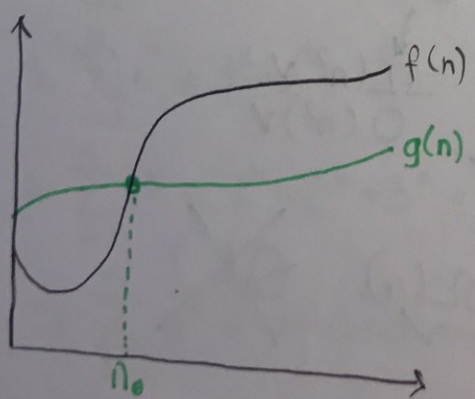
$f(n) \in \Omega(g(n))$ if $c > 0$, $n_0 > 0$, $n \ge n_0 \rightarrow f(n) \ge c*g(n)$

(Ω: asymptotic lower bound)

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

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

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



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

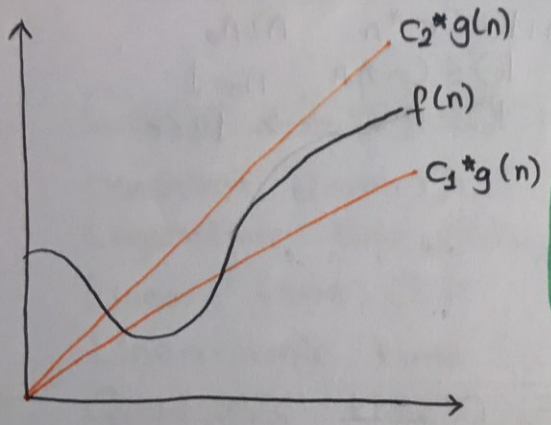
$(5-c)n^2 \ge 0$ $\quad n_0 = 1$, $c = 1$

$f(n) = 5n^2 \longrightarrow \Omega(n)$? ✓

⑥

Big-theta (Θ) Notation : tight bound → f(x) is asymptotically equal to g(x)

$0 \le c_1 * g(n) \le f(n) \le c_2 * g(n)$ $\quad c_1 \le c_2, n_0 > 0, n \ge n_0 \rightarrow f(n) = \Theta(g(n))$



Örnek: $f(n) = 2n+5$

$f(n) \in \{ \Omega(g(n)), O(g(n)) \} \rightarrow \Omega(g(n)) = O(g(n)) \rightarrow f(n) \in \Theta(g(n))$

$f(n) = 2n+5 \not\rightarrow \Theta(n^2)$

$f(n) \notin \Omega(n^2)$

$f(n) \notin \Theta(1)$ $\quad f(n) \notin O(1)$

Örnek: $f(n) = 2n+5 \in O(n)$

① $2n+5 \in ? \Omega(n) \rightarrow 2n+5 \ge c*n \rightarrow 5 \ge (c-2)*n$

$n_0 = 1, c = 3$ ✓

② $2n+5 \in ? O(n) \rightarrow 2n+5 \le c*n$

$5 \le (c-2)*n$

$n_0 = 1, c = 7$ ✓

$c_1 = 3, c_2 = 7, n_0 = 1 \rightarrow \boxed{c_1 * n \le 2n+5 \le c_2 * n} \rightarrow 2n+5 \in \Theta(n)$

Örnek: Insertion Sort

worst case $\sim n^2 \rightarrow O(n^2)$ ✓ $\quad \Theta(n^2)$ ✓

best case $\sim n \rightarrow \Theta(n)$ ✓ $\rightarrow \Omega(n^2)$ ✓

$\rightarrow O(n)$ ✓ $\quad O(n^2)$ ✓

$\Omega(n)$ ✓

Insertion Sort → $O(n^2)$ ✓ $\quad \Omega(n)$ ✓ $\quad \not\Theta(\,)$

Örnek: Merge Sort → $\Theta(n*\log n)$

Best case ~ $n*\log n$
Worst case ~ $n*\log n$

Classes of Algorithms

Constant time: $O(1)$	$a=b+c$
Logaritmic time: $O(\log n)$	binary search
Linear time: $O(n)$	loop
Linearitmic 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. $\log_a x = \log_a b * \log_b x$
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-elemet set: $C(n,k) = \frac{n!}{k!(n-k)!}$
3. Number of subsets of an n-elemet set: 2^k

-Important Summation Formulas

1. $\sum_{i=l}^{u} 1 = \underbrace{1+1+1+\dots+1}_{u-l+1} = u-l+1 \quad (l \le u)$
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. $\sum_{i=0}^{n} a^i = 1+a+a^2+\dots+a^n = \frac{a^{n+1}-1}{a-1} \quad (a \neq 1)$

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

(8)

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

7. $\sum_{i=1}^{n} \frac{1}{i} = 1+\frac{1}{2}+\frac{1}{3}+\dots+\frac{1}{n} \approx \ln n + \gamma \quad (\gamma \approx 0{,}5772)$

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

Sum Manipulation Rules

1. $\sum_{i=l}^{u} c*a_i = c*\sum_{i=l}^{u} a_i$
2. $\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$
3. $\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=(m+1)}^{u} a_i$ where $(l \le m \le u)$

- Floor and Ceiling Formulas

$\lfloor 3{,}8 \rfloor = 3$

$\lfloor -3{,}8 \rfloor = -4$

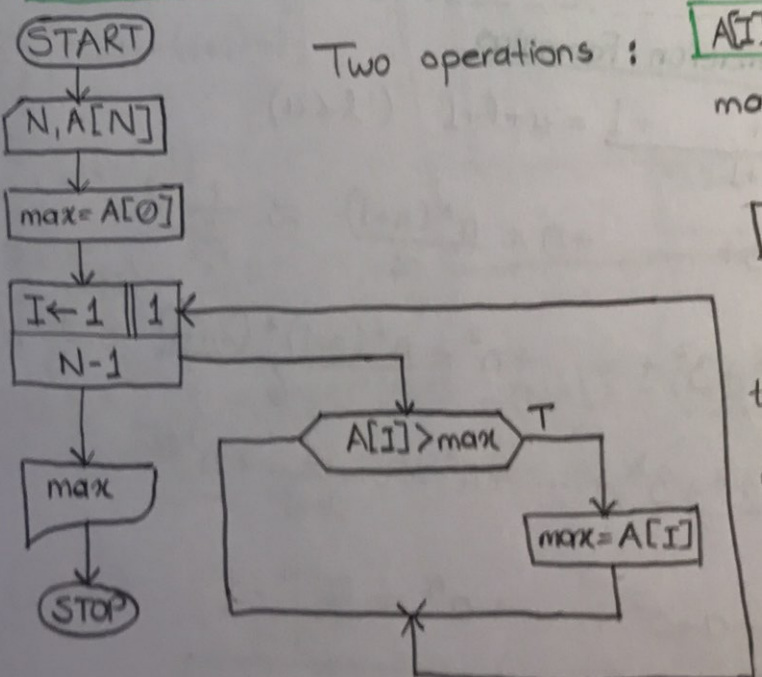
$\lceil 3{,}8 \rceil = 4$

$\lceil -3{,}8 \rceil = -3$

1. $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$
2. $\lfloor x+n \rfloor = \lfloor x \rfloor + n$ and $\lceil x+n \rceil = \lceil x \rceil + n$ (for real x and integer n)
3. $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$
4. $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$

Mathematical Analysis of Nonrecursive Algorithms

Örnek:



Two operations : `A[I] > max` → Basic operation (Çünkü if kontrolü her zaman yapılıyor, atama bazen oluyor.)

`max = A[I]`

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

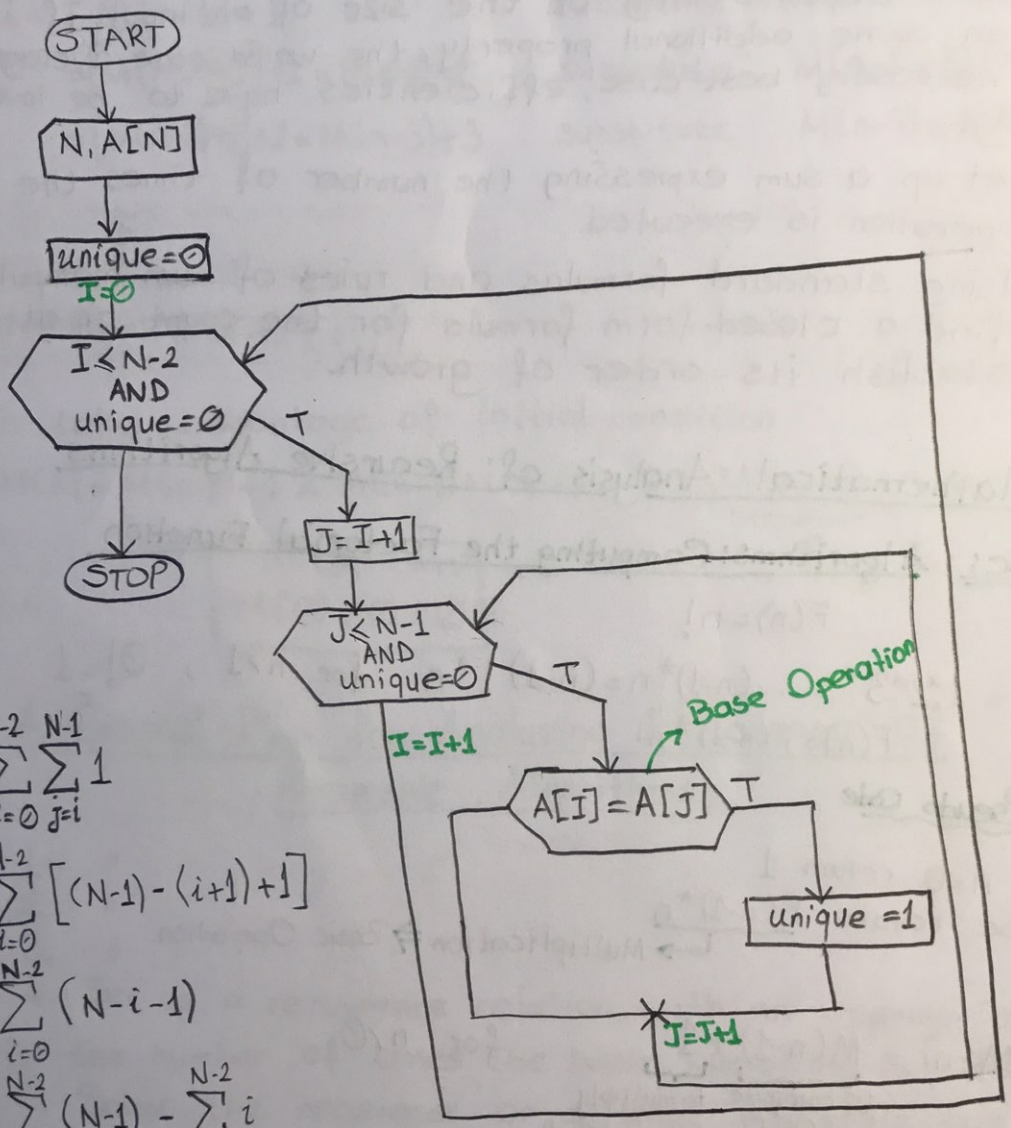
$C(n) = \sum_{i=1}^{n-1} 1 = n-1 \quad \Theta(n)$

Örnek: Algorithms: Element Uniquiness Problem

3	5	7	3	1	2	4

2	7	1	3	4	6	5

unique



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

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

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

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

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

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

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 seperately.
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

$$F(n) = n!$$

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

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

Pseudo Code

```
if n=0 return 1
else return F(n-1)*n
```

→ Multiplication ⇒ Basic Operation

$$M(n) = \underbrace{M(n-1)}_{\text{to compute } F(n-1)} + \underbrace{1}_{\text{to multiply } F(n-1) \text{ by } n} \quad \text{for } n>0$$

→Recurrences Relations←

-Initial Condition: It tells us the 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:

$M(n) = M(n-1) + 1$ for $n > 0$

$M(0) = 0$

Method of Backward Substitution

$M(n) = M(n-1) + 1$ substitute $M(n-1) = M(n-2) + 1$

$= [M(n-2)+1] + 1 = M(n-2) + 2$ substitute $M(n-2) = M(n-3) + 1$

$= [M(n-3)+1] + 2 = M(n-3) + 3$ substitute $M(n-3) = M(n-4) + 1$

⋮ ⋮ ⋮

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

To take advantage of initial condition

$M(n) = M(n-1)+1 = M(n-i)+i = \cdots\cdots = \underbrace{M(n-n)}_{0} + n = n$

$n = 0,\ i = n$

$M(0) + 0 = 0$

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

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

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:

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

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

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

$$= A(2^{k-2}) + 2$$
$$= A(2^{k-3}) + 3$$
$$\vdots$$
$$= A(2^{k-i}) + i$$
$$\vdots$$
$$= A(2^{k-k}) + k$$
$$= A(2^0) + k$$
$$= k$$

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

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

Binary Search Algorithm

Input : A[0, ..., n-1]

Output : Index or -1

Pseudo Code:

```
l ← 0, r ← n-1
while (L <= r) do
    m ← L(l+r)/2 J
    if K = A[m] return m;
    else if K < A[m]  r ⇐ m-1
    else  l ← 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	l						m						r
iteration2							l		m				r
iteration3							l	r					

Search value = 70

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

Initial Condition

$$C_w(1) = 1$$

$n = 2^k$

Backward Substitution

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

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

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

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

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

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

$$C_{avg}(n) \approx \log_2 n \longrightarrow C^{yes}_{avg}(n) \approx \log_2 n - 1$$

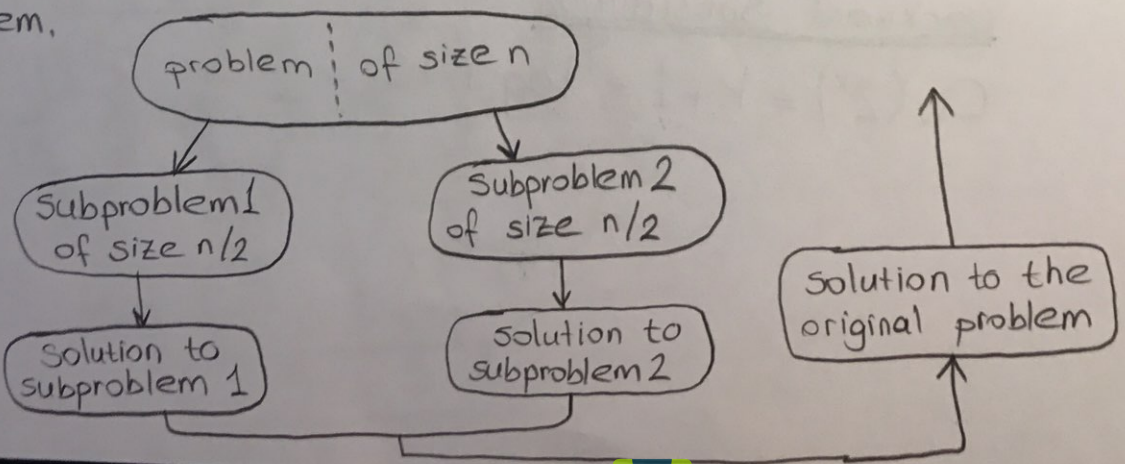
$$C^{no}_{avg}(n) \approx \log_2 (n+1)$$

$$\lfloor \log_2 10^3 \rfloor + 1 = 10$$
$$\lfloor \log_2 10^6 \rfloor + 1 = 20$$

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]

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

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

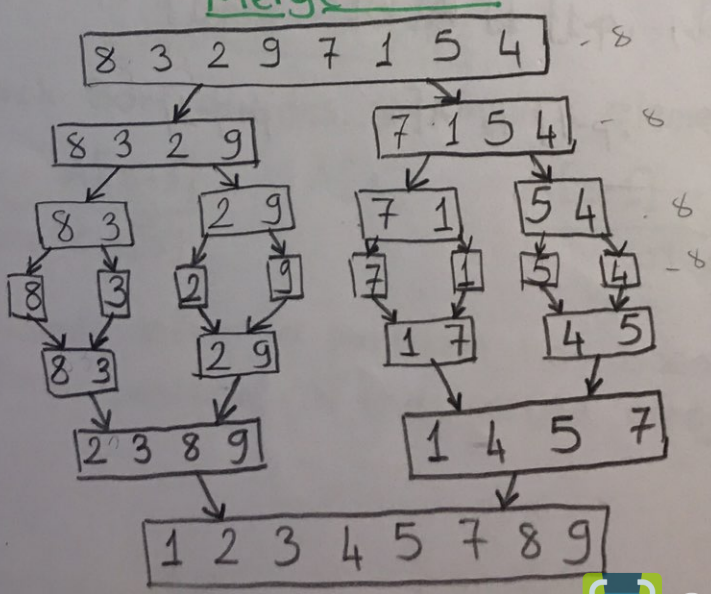
$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

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



$N \log_2 N$

16

Pseudo Code

```
Mergesort(A[0,.....,n-1])
  if (n > 1)
      copy A[0....⌊n/2⌋-1] to B[0.....⌊n/2⌋-1]
      copy A[⌊n/2⌋.....n-1] to C[0.....⌊n/2⌋-1]
      Mergesort(B[0.....⌊n/2⌋+1])
      Mergesort(C[0.....⌊n/2⌋+1])
      Merge(B,C,A)
```

$C(n) = 2C(n/2) + C_{merge}(n)$ $n>1$,

— 4. hafta / 18.10.2018

Initial Condition

$C(1) = \emptyset$

```
Merge(B[0,.....,p-1], C[0,.....,q-1], A[0,.....,p+q+1])
  i←0, j←0, k←0
  while i<p and j<q do
      if B[i] ≤ C[j]
          A[k] ← B[i], i←i+1
      else
          A[k]←C[j], j←j+1
      k←k+1
  if i=p
      copy C[j,.....,q-1] to A[k,.....,p+q-1]
  else
      copy B[i,.....,p-1] to A[k,.....,p+q-1]
```

- Recurrence Relation -

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

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

Initial Condition: $C(1) = 0$

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

- Backward Substitution -

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

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

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

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

$= 8C(n/8) + 3n \longrightarrow n = 2^i$, $i = \log_2 n$

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

$= nC(1) + \log_2 n * n$

$= n * \log_2 n$

space complexity'si diğerlerine göre çok yüksek.

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

Principal Shortcoming: Linear amount of extra storage.

Quick Sort

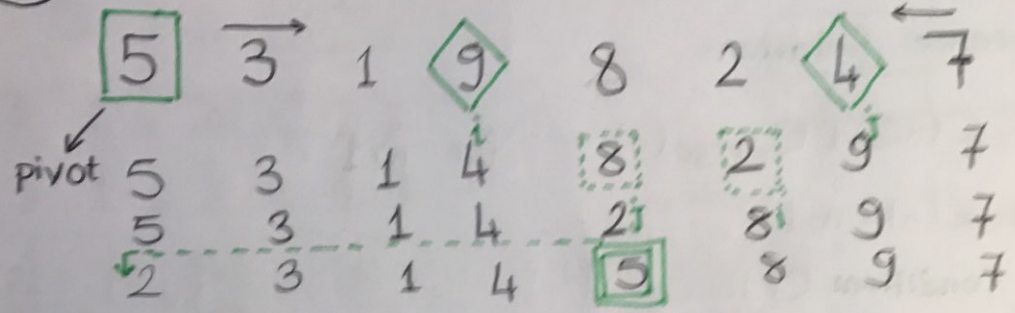
Quick Sort divides the input's elements according to their value.

A[0] ... A[s-1] | A[s] | A[s+1] ... A[n-1]

all are $\leq A[s]$ | all are $> A[s]$

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

(18)



(a shot example whole example on p.19)

Algorithms:

```
if l < r
    S ← Partition(A[l, ....., r])
    Quicksort(A[l, ....., s-1])
    Quicksort(A[s+1, ....., 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:

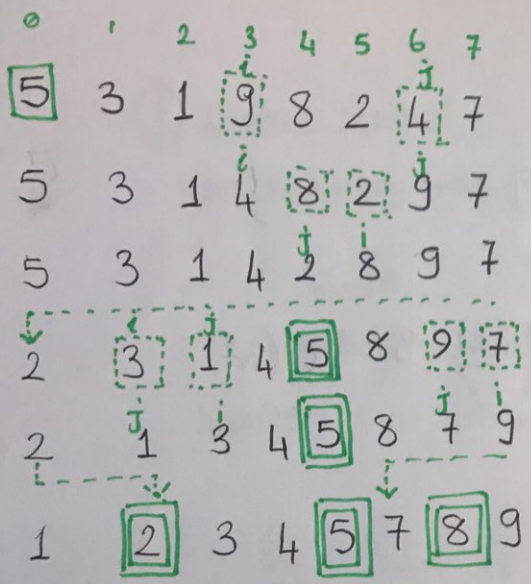
1. If scanning indicies i and j have not crossed e.g. $i<j$ we simply exchange $A[i]$ and $A[j]$ and resume the scans by incrementing i and decrementing j repeatively.

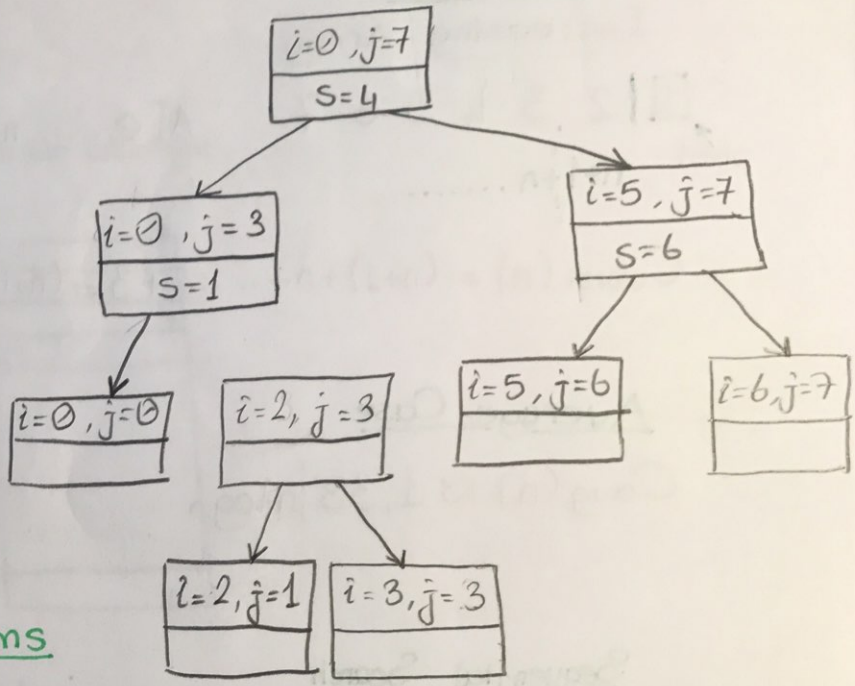
2. If the scanning indicies have crossed over i.e. $i>j$ we have partioned the array after exchanging the pivot with $A[j]$

3. If the searching indicies 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 $i \geq j$





Partition Algorithms

```
p ← A[l]
i ← l , j ← r+1
repeat
    repeat i ← i+1 until A[i] ≥ p
    repeat j ← j-1 until A[j] ≤ p
    swap (A[i], A[j])
until i ≥ 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~

$C_{best}(n) = 2C_{best}(n/2) + n$ $\quad C_{best}(1) = 0$

$C_{best}(n) \in \Theta(n \log n)$

(20)

Worst-Case

Increasing Array

1	2	3	4	5	6	7

pivot

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

$n+1 + n \dots\dots$

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

Average Case

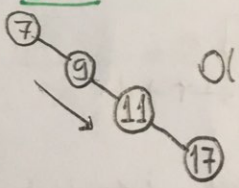
$$C_{avg}(n) \approx 1{,}38\ n \log n$$

Sequential Search

0	1	2	3	4	5
3	5	7	4	2	1

$O(n)$

BST



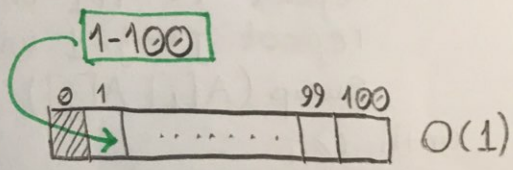
$O(n)$

Binary Search

2	7	9	12	17	19	22

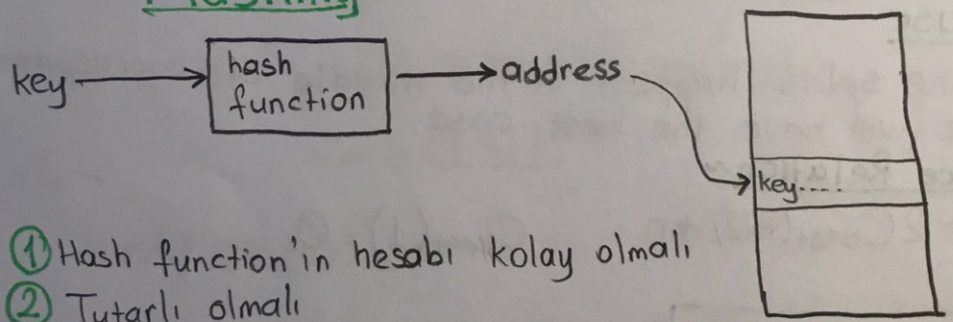
$O(\lg_2 n)$

x=7



$O(1)$

Hashing



1. Hash function'ın hesabı kolay olmalı
2. Tutarlı olmalı
3. Adresin olabildiğince uniform dağılması sağlanmalı

key mod (m) ← tablo uzunluğu

m = 10 $\longrightarrow 2^{\alpha}$

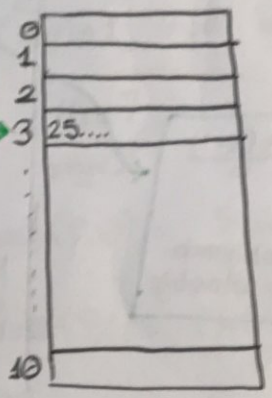
key $\longrightarrow$ 1000, 10, 40, 50

* m'in asal sayı olması performansı artırır

① Divison Method

hash(key) = key mod m
↳ tablo uzunluğu

m=11
key=25
hash(25)=3



② Multiplication Method

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

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

tablo uzunluğu

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

m = 10.000 k = 123456

$hash(123456) = \lfloor 10.000 * (123456 * 0,618033) \bmod 1 \rfloor$

$123456 * 0,618033 = 763.000,004151$

$= \lfloor 10.000 * 0,004151 \rfloor$

$= \lfloor 41,51 \rfloor$

$hash(123456) = 41$

41 ↑ address

22

Birthday Paradox

$$365 \longrightarrow \frac{365-1}{365} * \frac{365-2}{365} * \cdots\cdots * \frac{365-(k-1)}{365}$$

3 kişinin doğum günü farklı

table size: m

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

m tablo uzunluğunda çakışmama olasılığı

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

$N = 2^{32}$

32-bit hash values

after 250.000 insert %100

77163 insert %50

Key

1. Integer
2. Floating number
3. String

ikisi farklı değer döndürmeli — "ELMA" / "MALE" → Horner's Method:

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

- R: 31 (olarak seçilir)
- L-1: length of string
- str[0]: Büyük harf ise: str[0] - 'A' + 1

4. Compound key

31.08.1997

↳ $31 * R + 8 * R^2 + 1997 * R^3$

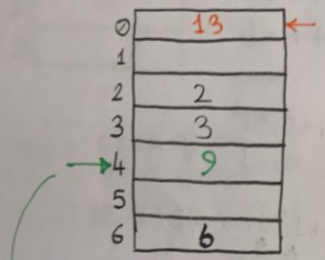
R = 31

Çakışma Problemine Çözüm Önerileri

1 Open Addressing → table size $m \sim 2*N$ olduğunda rahat çalışıyor.
- delete işlemi sıkıntılı (ya yeniden düzenlenmeli ya da aramaya devam etmesi için uygun flag konur.)

1.1 Linear Probing

$hash(key, i) = [h'(key) + 1] \bmod m$ $\quad i = 0, \ldots, m-1$



$m = 7$

$key = 9$

i: $hash(9,0) = 9 \bmod 7 = 2$ ⇒

i+1: $hash(9,1) = (9+1) \bmod 7 = 3$

i+2: $hash(9,2) = (9+2) \bmod 7 = 4$

$key = 13$

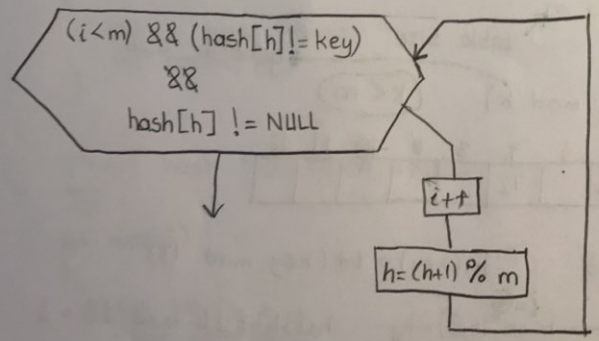
$hash(13,0) = 6$

$hash(13,1) = (13+1) \bmod 7 = 0$ ⇒

key

$i = 0$

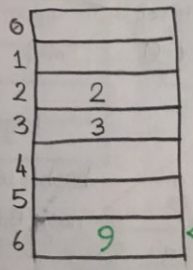
$h = key \bmod m$



1.2 Quadrate Probing

Linear Probing'de her yerleşecek eleman için örneğin 1'er pozisyon kayma durumu olabilir. Bunu önlemek için arada boşluk bırakarak atlama yapabilecek bir çözüm sunar.

$$h(k,1) = (h'(k) + i^2) \bmod m$$



$hash(9,0) = 9 \bmod 7 = 2$

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

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

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

$c_1 = 2 \quad c_2 = 1$

$hash(9,1) = (2 + 2*1 + 1*1) \bmod 7 = 5$

$c_1=2$, $i=1$, $i=1$, $c_2=1$

1.3 Double Hashing

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

$i * h_2(key)$ → adım büyüklüğü

$h_1(key) = key \bmod m$ ← table size

$h_2(key) = 1 + (key \bmod k)$ → genelde asal, $k < m$

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

$h_1(key) = key \bmod 13$ $\qquad$ $h_2(key) = 1 + (key \bmod 11)$

Ex: $hash(14) = ?$ $\longrightarrow$ $i=0$, $hash(14,0) = h_1$ $\qquad$ $h_1(14) = 14 \bmod 13 = 1$

$k = 11$

$i = 1$

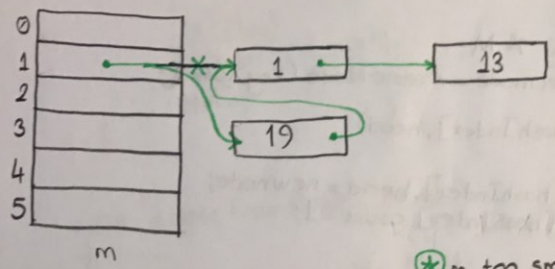
$hash(14,1) = 1 + [1 + \underbrace{14 \bmod 11}_{3}] * 1 = 1 + 4 = 5$

$i = 2$

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

$h_1(key)$ $\qquad$ $h_2(key)$

2 Separate Chaining



key = 13 % 6 → 1
key = 1 % 6 → 1
key = 19 % 6 → 1

M < N
maximum → M = N

* m too small → chains too long
* statistical optimum value → M ≈ N/5

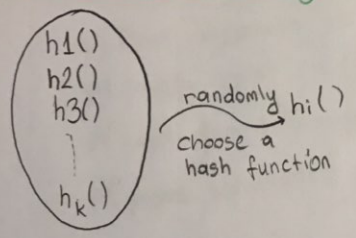
```
Struct node
{
    int key;
    char name[40];
    Struct node *next;
};

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)
    {
      ≡
    }
    insertToHash(key, name);
    searchInHash(key);
    deleteFromHash(key);
    return 0;
}
```

(26)

```
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++;
    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");
        return;
    }
    while(myNode != NULL && flag == 0)
    {
        if(myNode->key == key)
        {
            printf("VoterID :%d\n", myNode->key);
            printf("Name : %s\n", myNode->name);
            flag=1;
        }
```

```
            myNode = myNode->next;
        }
        if (!flag)
            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 && flag != 1)
    {
        if(myNode-> key == key)
        {
            flag=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(flag)
        printf("Data deleted succesfully from Hash Table\n");
    else
        printf(" Given data is not presented in hash Table\n");
}
```

3 Universal Hashing



15, 22, 29 → (7) → hash → 1

- Kullanılan Hash Fonksiyonları -

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

a, b, p → choosen 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

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

open addressing linear probe

α	# of probe
0,1 →	1,11
0,2 →	1,28
0,3 →	1,52
0,4 →	1,89
0,5 →	2,5
0,6 →	3,62
0,8 →	13,00
0,9 →	50,50

α = 0,1 → $\frac{1}{10}$ (N / M)

$\frac{1}{2}$ ✓

Unsuccesful Search

p(first location is occupied) = α

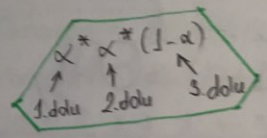
p(empty cell) = $(1-\alpha)$

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

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

(α: 1st step, $(1-\alpha)$: 2nd step)

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

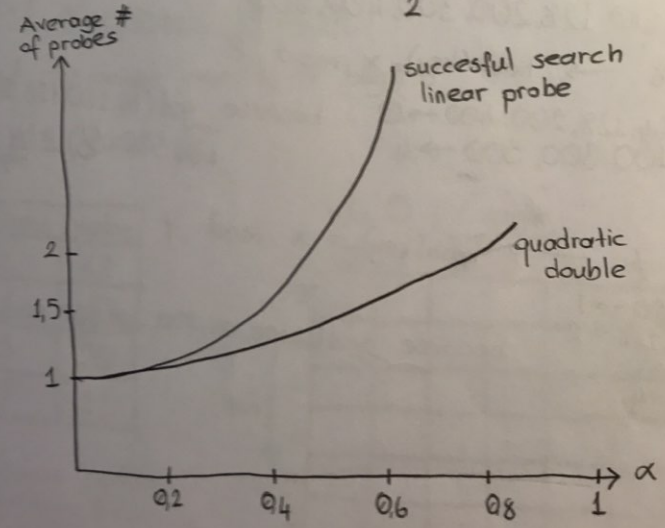


Average # of steps:

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

Expected number of probes in unsuccesful search.

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

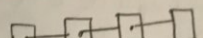


30

Separate Chaining

1 Unsuccesful Search

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



Worst case: Bütün elemanlar aynı adreste.

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

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

$\Theta(1+\alpha)$ → search, hash function

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

$\Theta(1+\alpha) = \Theta(1)$

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

table size = 8 → hash(key) = x mod 8

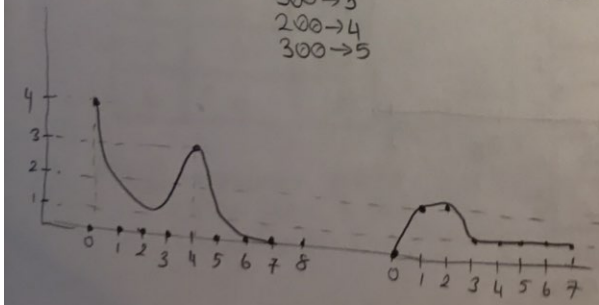
64, 128, 200, 400 → 0
100, 300, 500 → 4

because gcd(64, 100, 128, 200, 300, 400, 500, 8) = 4

table size = 7 → hash(key) = x mod 7

64, 400 → 1
100, 128 → 2
500 → 3
200 → 4
300 → 5

because gcd(64, 100, 128, 200, 300, 400, 500, 7) = 1



Dynamic Programming

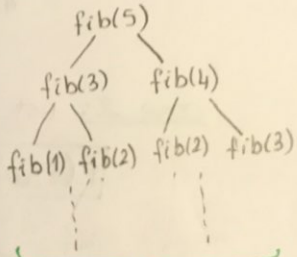
→ Planning Richard Bellman (1950's)

Example: Fibonacci Numbers

0 1 1 2 3 5

→ n=0 : fibonacci(n) = 0
n=1 : fibonacci(n) = 1
n>1 : fibonacci(n) = fibonacci(n-1) + 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:

fibonacci(n) = fibonacci(n-1) + fibonacci(n-2) $(1,6)^n$ summation

← çok fazla tekrarlanan hesap var

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

fibonacci

(*) hesaplamaları tek tek yeniden değil, bu dizideki gözde daha önce hesaplanmış değerleri kullanıyor.

→ abandon
→ select

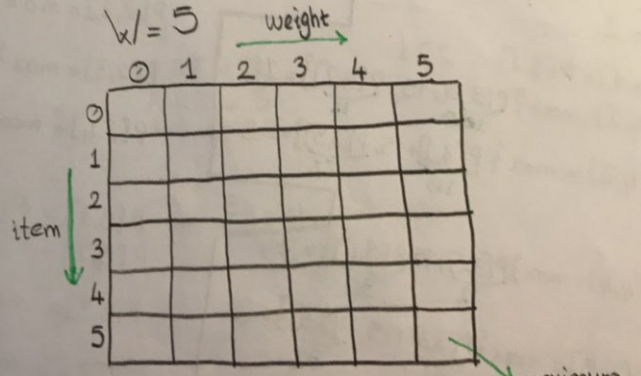
Example: 0-1 Knapsack Problem

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



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

maximize $\sum v_p$

subject to W

(32)

Case 1: thief takes item i

$$P(i,\omega) = P(i-1, \omega-\omega_i) + v_i$$

$i=3$, $\omega_i = 2$

$$P(3,4) = P(2, \underbrace{2}_{\rightarrow 2+\omega_i}) + v_3$$

↓ 4 kiloya gelindiğinde maximum kazanç

$i=3$

$P(3,3) = P(2,1) + v_3$

↓ eşya — ↳ 3 kg'a kadar kazanç

Case 2: not takes item i

$$P(i,\omega) = P(i-1,\omega)$$

$$P[i,\omega] = \begin{cases} P[i-1,\omega] & \text{if } \omega_i > \omega \\ \max\{v_i + P[i-1,\omega-\omega_i],\ P[i-1,\omega]\} & \text{others} \end{cases}$$

initialization → $P[0,\omega] = 0$, $P[i,0] = 0$

weight

item	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	(37)

→ maximum kazanç

$\omega_1 = 2$

$P[1,1] = 0$ 2>1

$P[1,2] = \max\{12+0, 0\} = 12$

$P[0,0]+12$ $P[0,2]$

$P[1,3] = \max\{P[0,1]+12, P[0,3]\} = 12$

$P[2,1] = \max\{P[1,0]+10, P[1,1]\}$

$P[2,2] = \max\{P[1,2], P[1,1]+10\}$ (12, 0)

$P[2,3] = \max\{P[1,2]+10, P[1,3]\} = 22$ (12, 12)

$P[2,4] = \max\{P[1,3]+10, P[1,4]\} = 22$

$\omega_2 = 1$

$P[3,1] = P[2,1]$ → 3>1

$P[3,2] = P[2,2]$ → 3>2

$P[3,3] = \max\{P[2,0]+20, P[2,3]\} = 22$ (0, 22)

$P[3,4] = \max\{P[2,1]+20, P[3,4]\} = 30$ (10, 22)

$P[3,5] = \max\{P[2,2]+20, P[3,5]\} = 32$ (12, 22)

$\omega_4 = 2$

$P[4,1] = P[3,1]$ 2>1

$P[4,2] = \max\{P[3,0]+15, P[3,2]\} = 15$ (0, 12)

$P[4,3] = \max\{P[3,1]+15, P[3,3]\} = 25$ (10, 22)

$\omega_3 = 3$

$P[4,4] = \max\{P[3,2]+15, P[3,4]\} = 30$ (12, 30)

$P[4,5] = \max\{P[3,3]+15, P[3,5]\} = 37$ (22, 32)

	0	1	2	3	4	5	
0	0	0	0	0	0	0	
1	0	0	12	12	12	12	✓ $w_1=2$
2	0	10	12	22	22	22	✓ $w_2=1$
3	0	10	12	22	30	32	✗ $w_3=3$
4	0	10	15	25	30	37	✓ $w_4=2$

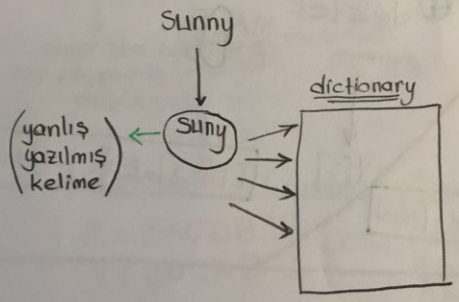
Hangi item'ler alınmış?

Reconstructing Optimal Solution

not taken ↑ $P[i,j] = P[i-1,j]$

taken ↖ $P[i,j] \rightarrow P[i-1, j-w_i]$

Edit Distance



The Levenshtein Algorithm (Edit Distance)

sunny → sumny

sunny → sun-y

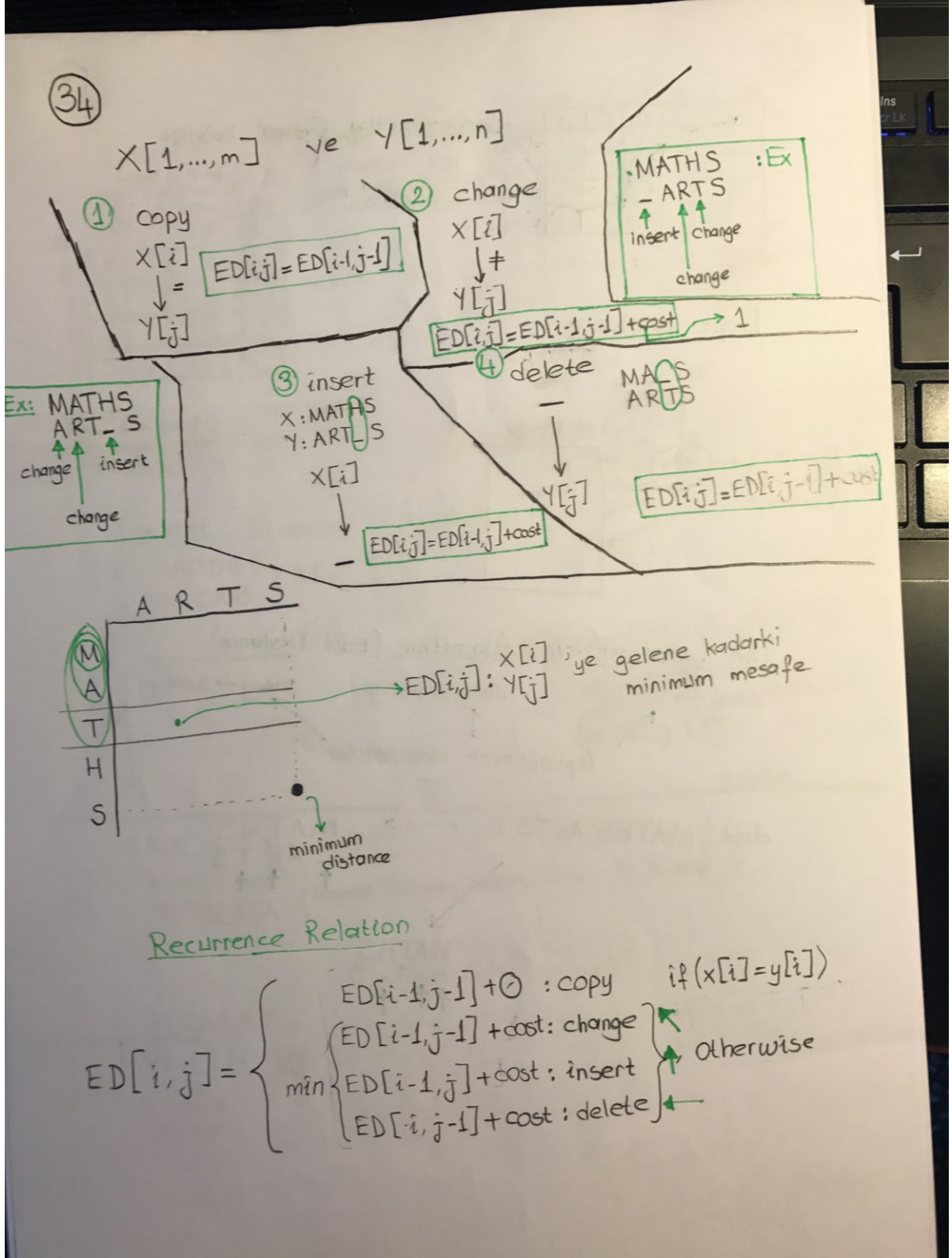
sunny → sunnyy

Yapılabilecek olası hatalar

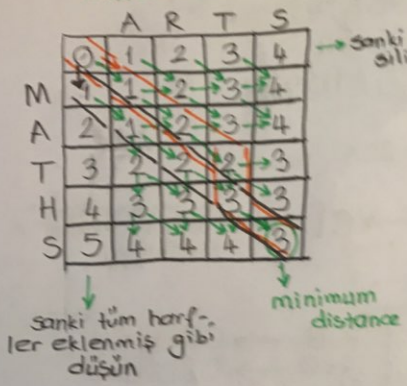
dist(MATHS, ARTS) →

MATHS
-ARTS

MATHS
ART-S



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



MATHS ne oldu da ARTS oldu sorusunun cevabı bulunulabilir.

1. deneme: MATHS / -ARTS

2. deneme: MATHS / ART-S

Longest Common Subsequence

X: BACDB
Y: BDCAB
} LCS: BCB, BDB, BAB

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

X: ↓ , Y: →

8. Hafta

<A, C, G, T>

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

$LCS(X,Y) = TCTA$
$LCS(X,Y) = TGAT$

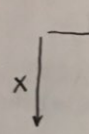
subsequence → KOZALAK

- KAL
- KOL
- KOLA
- KALK
- ~~AE~~

36

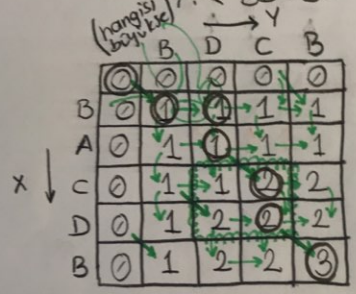
$X = \langle x_1, x_2, \ldots, x_m \rangle$
$Y = \langle y_1, y_2, \ldots, y_n \rangle$
} LCS

$S = \langle s_1, s_2, \ldots, s_k \rangle$
$s_{i_1}, s_{i_2}, \ldots, s_{ik}$
$i_1 < i_2 < \ldots < i_k$.



Örnek: $X: \langle BACDB \rangle$
$Y: \langle BDCB \rangle$

$X_i\ Y_j \longrightarrow C[i,j]$ → o noktaya kadar ortak elemanların uzunluğu



Initial: if $i = 0 \longrightarrow C[0,j] = 0$
if $j = 0 \longrightarrow C[i,0] = 0$

$$C[i,j] = \begin{cases} 0, & \text{if } i=0 \text{ or } j=0 \\ C[i-1,j-1]+1, & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max\{C[i-1,j],\ C[i,j-1]\}, & \text{otherwise} \end{cases}$$

Hangi harfleri aldığımıza dair [M,N] gözünden ilerlemeye başladığımızda harf ortak olarak alındı mı alınmadı mı belli olmuyor. Karmaşa oluyor.

↓
ÇÖZÜM
↓
Yeni bir Matris
B[6,5]

ADD: 0
SKIPX: 1
SKIPY: 2

$X_i = Y_j \longrightarrow B[i,j] = ADD$
$X_i \neq Y_j \longrightarrow C[i-1,j] \geq C[i,j-1]$ then
X[i] not in LCS
B[i,j] = SKIPX
else
B[i,j] = SKIPY
Y[j] is not in LCS

B[6,5]

BCB

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 SKIPX:
        {
            i--;
            break;
        }
        case SKIPY:
        {
            j--;
            break;
        }
    }
}
```

- 1. Vize Cevaplar -

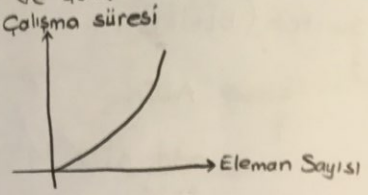
5/① Çalışma zamanı : ~~$8N^2$~~ $+13N^{4/3}+12N+3$

$N \rightarrow 1$ saat

$2N \rightarrow ?$ → 4 saat

Cevap: 2 saat

(Bellek ve çalışma zamanının hızı → x2)



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

~~$3n^3+20n^2+5$~~ $\rightarrow O(n^3)$ ⟶ BÖYLE SAÇMALANMIŞ

$f(n) \leq c\,g(n)$

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

$n=1$

$c=28$

$3+20+5 \leq 28$

$28 \leq 28$

10/③ İki algoritmanın performans karşılaştırması kıstası.

Time complexity, Space complexity

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

↳ Theoretical Analysis

↳ Empirical Analysis (Girdiyi artırıp denemeler yapmak)

15/④

10/⑤

```
void bubleSort(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(j<N-2-i){
                swap=1;
                tmp=dizi[j+1];
                dizi[j+1]=dizi[j];
                dizi[j]=tmp;
            }
            j++;
        }
    }
    i++;
}
```

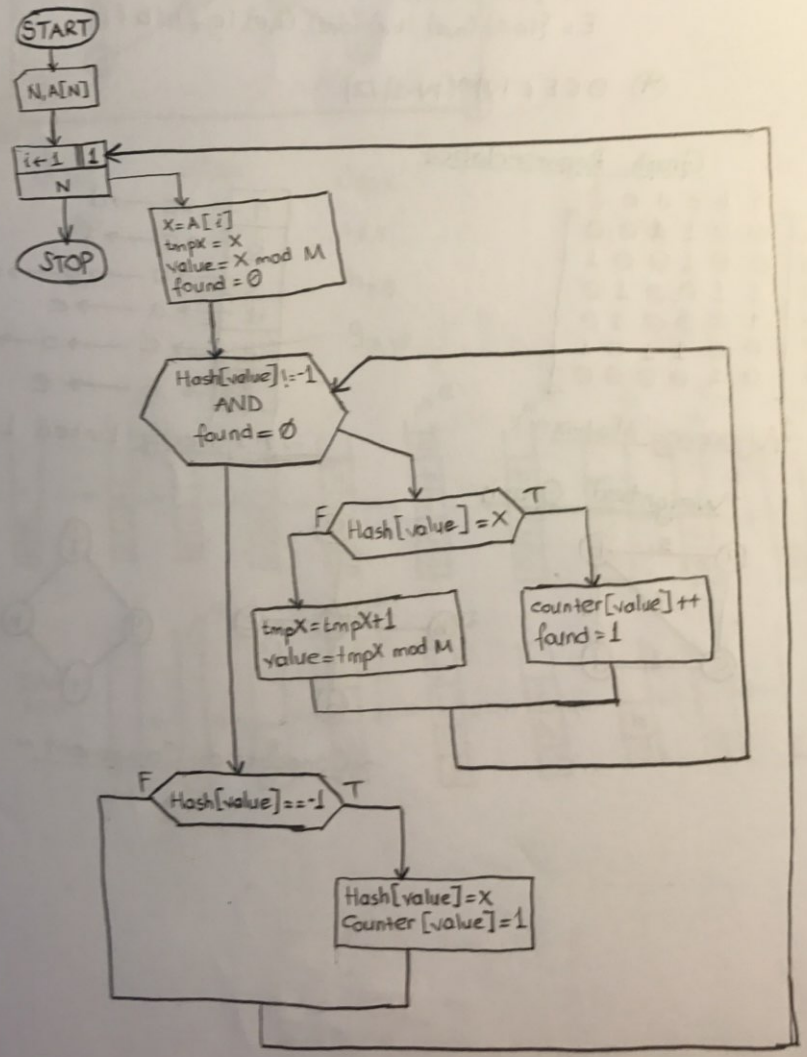
6/(6)

(1. ifade 2.'ye göre,)

$n(n+5)$	ve	$5000n^2$	→ E
2^{n-1}	ve	2^{n+1}	→ E
$\log_2(n)$	ve	$\ln(n)$	→ E
$1000n^2$	ve	$0{,}001n^3$	→ K
$(n-5)!$	ve	2^n	→ B
$(10n+5)\log_2 n$	ve	$100n+(n+n)\log_2 n$	→ E

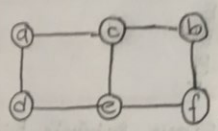
(E: eşit, B: büyük, K: küçük)

25/(7)

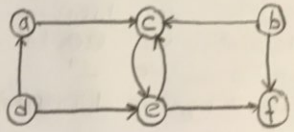


40

Graph → $G = \langle V, E \rangle$



-undirected graph-



-directed graph-

(a) → Can be graph

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

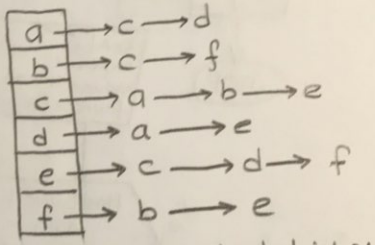
$E = \{(a,c),(a,d),(b,c),(d,e),(c,e),(e,f),(b,f)\}$

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

Graph Representation

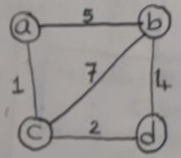
	a	b	c	d	e	f
a	0	0	1	1	0	0
b	0	0	1	0	0	1
c	1	1	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	1	0	1
f	0	1	0	0	0	0

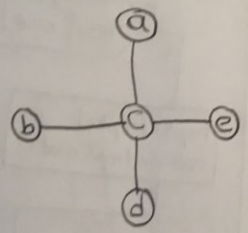
~Adjacency Matrix~

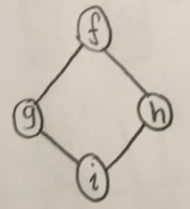


~Adjacency Linked List~

Weighted Graph







~Connected Component~

Depth-First-Search

Decrease-by-one technique

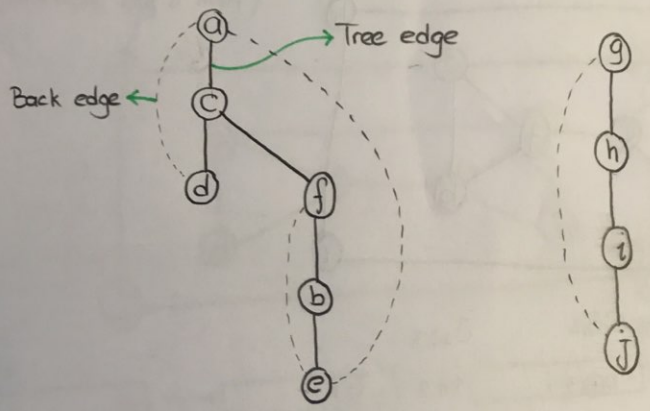
LIFO
↑
STACK kullanılır.
(push & pop yapılacak)

d 3,1 e 5,2 j 10,7
c 2,5 b 6,3 i 9,8
a 1,6 f 4,4 h 8,9
g 7,10

(42)

Dept-First-Search Forest

- Tree edge
- Back edge



DFS(G)

```
mark each vertex in V with ∅ as a mark of being unvisited
count ← ∅
for each vertex v in V do
    if v is marked with ∅
        dfs(v)
```

recursive

```
dfs(v)
    count ← count+1, mark v with count
    for each vertex w in V adjacent to do
        if w is marked with ∅
            dfs(w)
```

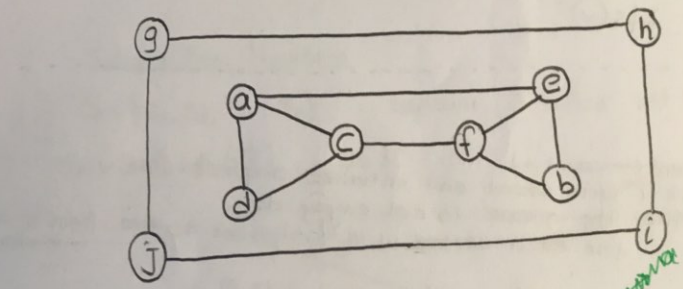
Breadth-First-Search

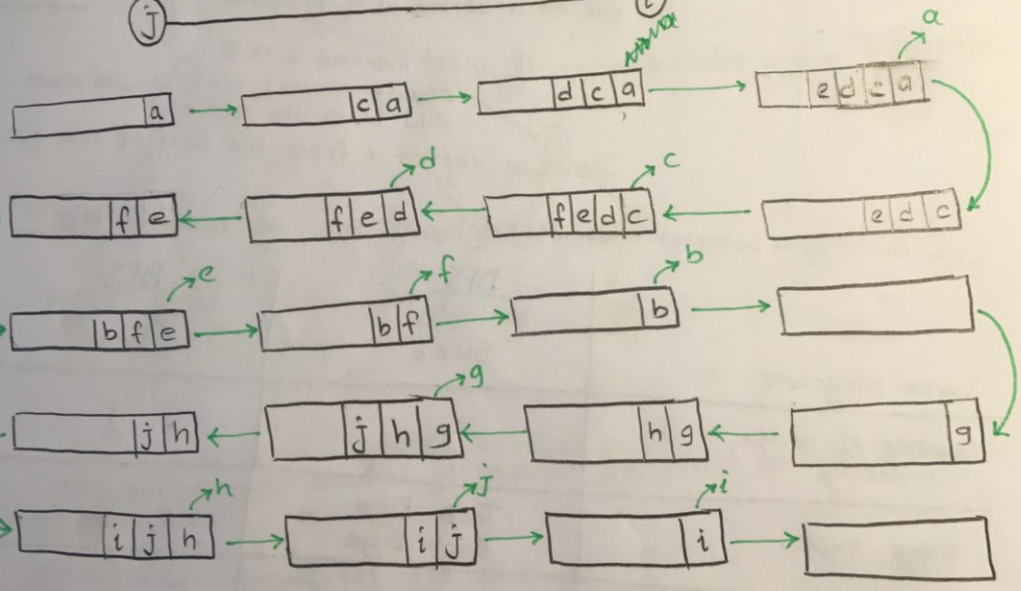
cesur / brave → depth
tedbirli / cautious → breadth

FIFO
↑
QUEUE kullanılır.
(queue & enqueue yapılacak)

$a_1 c_2 d_3 e_4 f_5 b_6$

$g_7 h_8 j_9 i_{10}$





İŞLEM TAMAM

(44)

BFS(G)

mark each vertex in V with ∅ as a mark of being unvisited
count ← ∅
for each vertex v in V do
    if v is marked with ∅
        bfs(v)

---

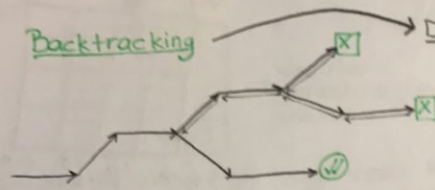
iterative

bfs(v)
    count ← 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 v do
            if w is marked with ∅
                count ← count+1, mark w with count
                add w to the queue
        remove vertex v from the front of the queue

web crawling → İnternet sitesinde link içinde link ve buradaki bilgileri çekme

	DFS	BFS
Data structure	Stack	Queue
Number of vertex orderings	2	1
Edge types	Tree Edge Back Edge	Tree Edge Cross Edge
Applications		Web crawling (Belirli sınır vermek için) Social network Map
Efficiency for adjacent matrix	$\Theta\|V^2\|$	$\Theta\|V^2\|$
Efficiency for adjacent linked-list	$\Theta\|V\|+\|E\|$	$\Theta\|V\|+\|E\|$

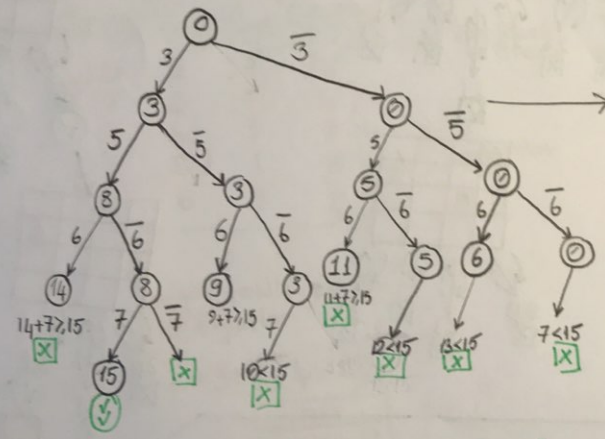
Backtracking → DFS with pruning



Subset Sum Problem

$S = \{s_1, s_2, \ldots, s_n\}$, toplamı d olan alt kümeler? $(s_1 \leq s_2 \leq s_3 \leq \ldots \leq s_n)$

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



8-vezir problemi (n-queen problem)

Satranç tahtasına birbirini görmeyecek(yemeyecek) şekilde 8 vezirı yerleştirme.

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

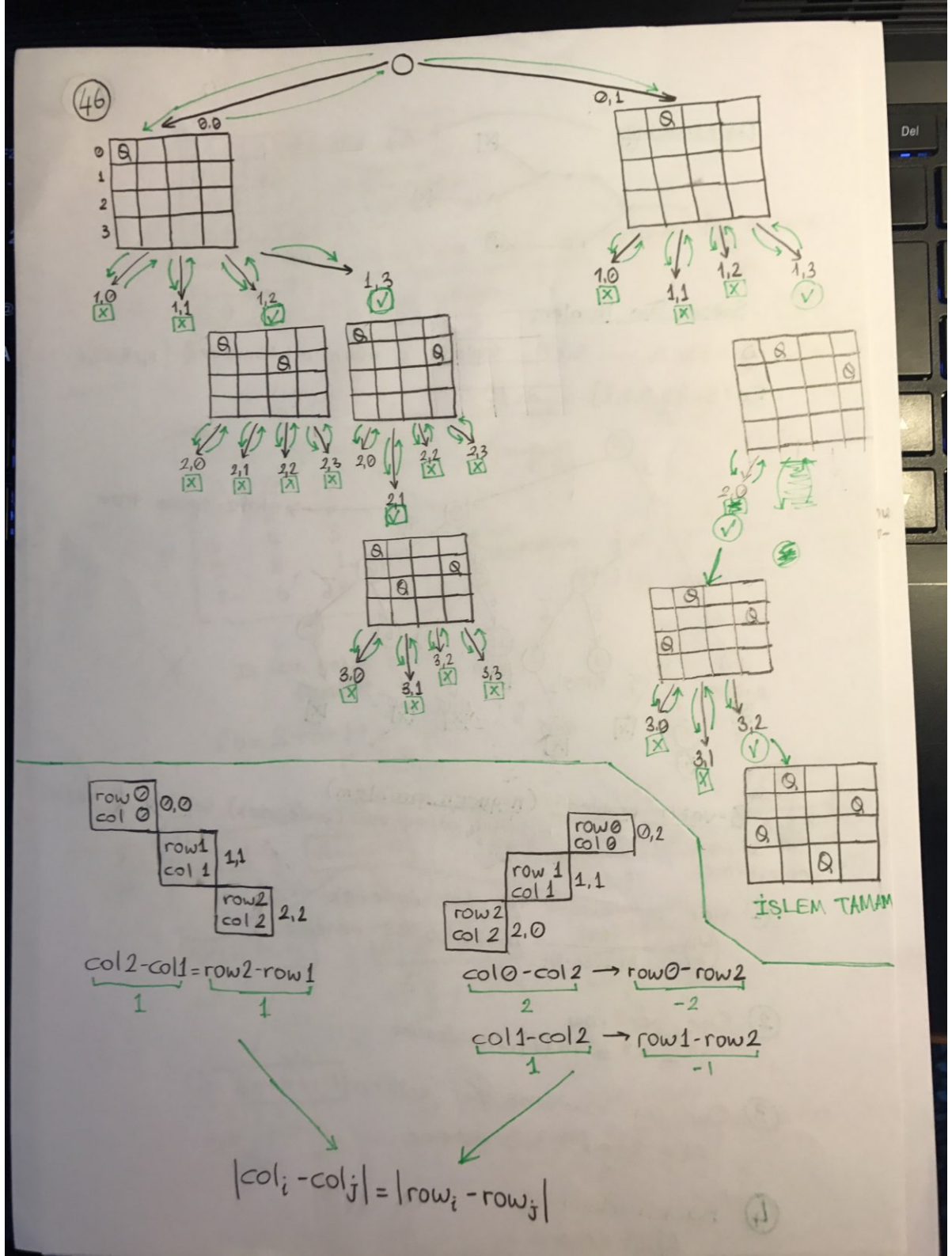
$n^n = 8^8 = 16.777.216$ deneme

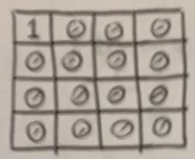
3. One per row and per column

$n! = 8! = 40.320$ deneme

4. Backtracking

2047 deneme





→ Vezirleri yerleştirdiğim matris

★ Ekstra bir diziyi hangi vezir hangi satır/sütunda onu saklarız.

0	1	2	3
2	4	1	3

← Hangi vezir (. satır veziri)

← Hangi sütunda

Branch and Bound → BFS

→ Yapılacak işler

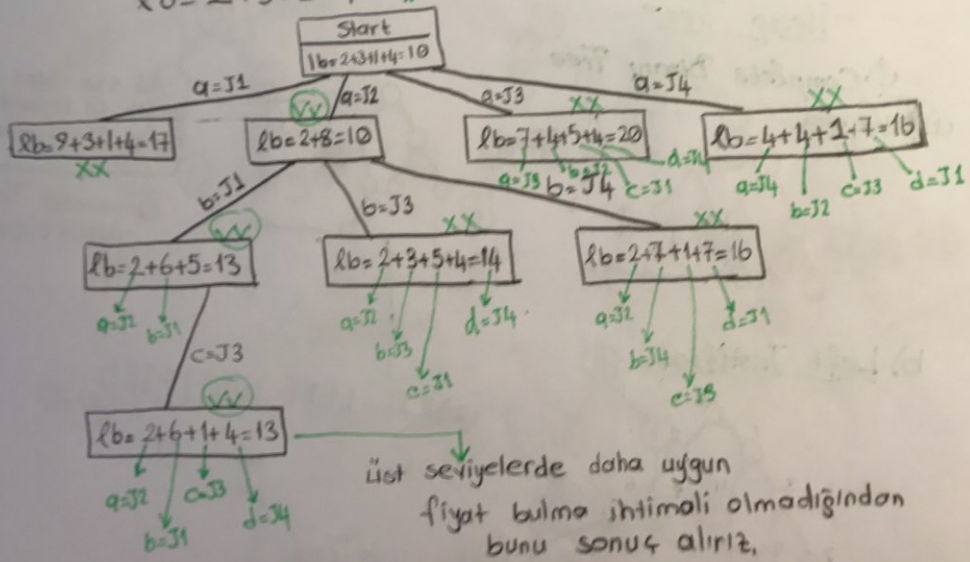
$$C = \begin{bmatrix} J1 & J2 & J3 & J4 \\ 9 & 2 & 7 & 4 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \begin{matrix} a \\ b \\ c \\ d \end{matrix}$$

→ İşi yapacaklar

İş için teklif edilen fiyatlar

Elimizde az para ve yapılacak işler var bunu en az maaliyetle yapmaya çalışıyoruz.

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



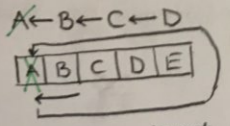
List seviyelerde daha uygun fiyat bulma ihtimali olmadığından bunu sonuç alırız.

48

Priority Queue

First-in-first-out

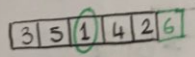
A ← B ← C ← D



- array
- linked list

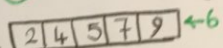
① - unsorted array -

- insert
- remove

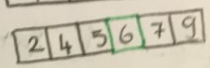


- insert O(1)
- remove O(N)

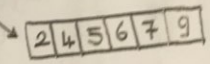
② - sorted array -



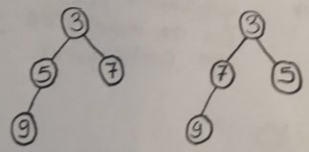
← 6 - insert O(n)



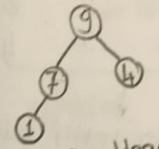
linked list
circular array



- remove O(1), O(N)



~ min Heap ~

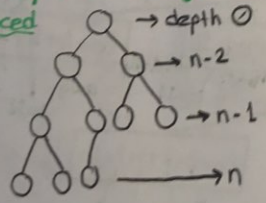


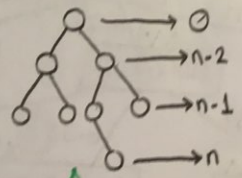
~ max Heap ~

- Heap Tree -

◊ Complete Binary Tree

a) Balanced

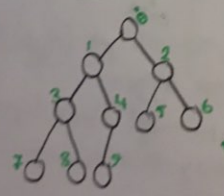




All the nodes at depth ⊘ through n-2 have 2 children !

Balanced Tree condition

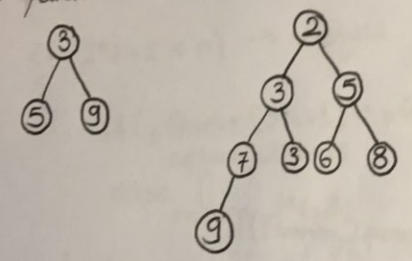
b) Left Justified



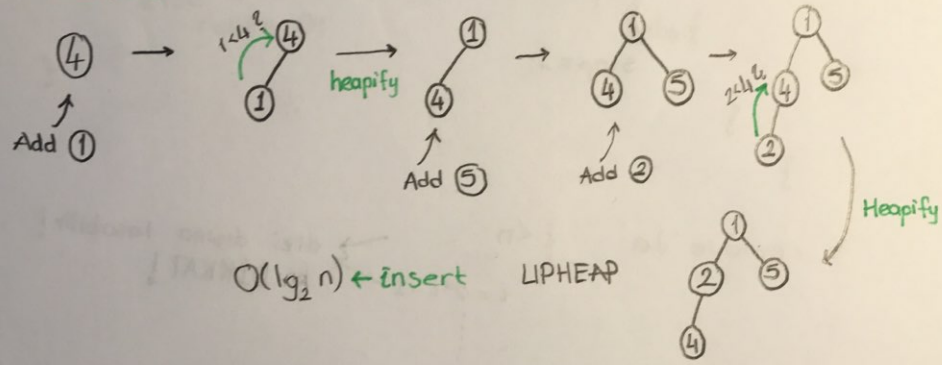
← left-justified

NOT LEFT-JUSTIFIED

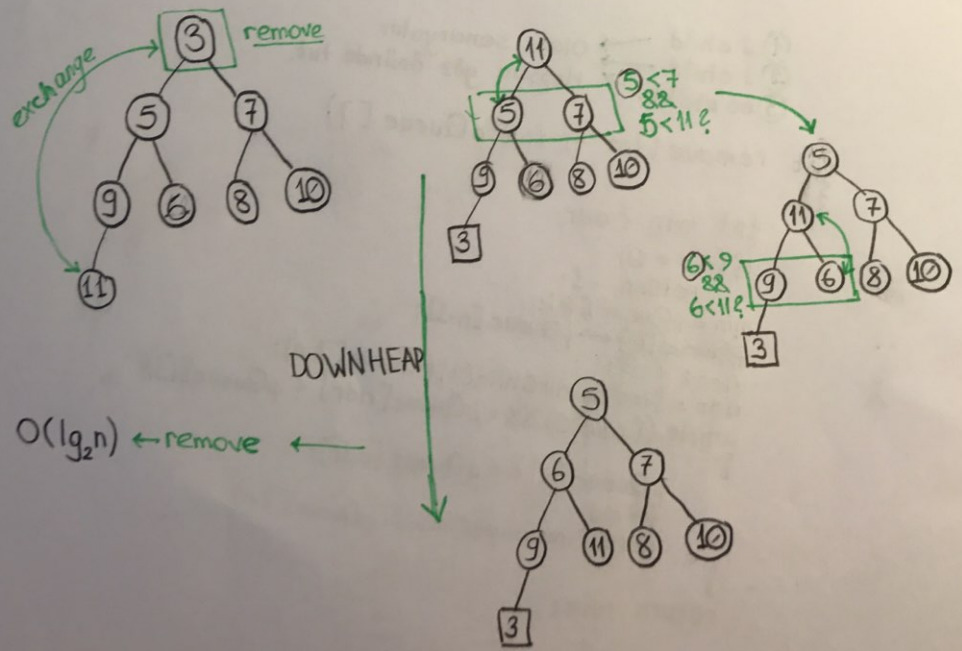
② a parent's value ≤ it's children's values : min Heap



Heapify



$O(\lg_2 n)$ ← insert LIPHEAP



$O(\lg_2 n)$ ← remove ←

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;
    }
}
```

remove'da `i < n` → dizi dışına taşabilir! DİKKAT!

```
{
    i = 2*i + 1
    ⋮
```

1. 2 child → Olası senaryolar
2. 1 child → Hepsini göz önünde tut.
3. no child

```
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;
}
```

```
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
        return 0;
}
```