Algoritma Analizi

BLM3021 Mine Elif KARSLIGIL & M. Amaç GÜVENSAN

Kaynaklar :

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

3 ödev : %15
Proje : %10
2 Vize : %45
Final : %30

** Programlama dili → C dili (ödeve göre değişebilir)

Algoritma Yer Karmasıklığı

- değişkenler
- sabitler
- program komutları
- fonk. çağırma vb. sebeplerle yığın işlemleri
- rekürsif fonksiyonları yığını doldurabiliyor.

Algoritmaların Karşılaştırması

Doğru mu?
Hızlı mı?
Az yer kaplıyor mu?

Algoritma Çalışma Zamanının Deneysel Ölçümü

3-sum problemi : Verilen N farklı sayı içinden toplamı Ø eden üçlülerin bulunması

a[i]	a[j]	a[k]	sum
-30	-10	40	Ø
-30	-10	50	10
-10	40	50	80
-10	40	-30	Ø

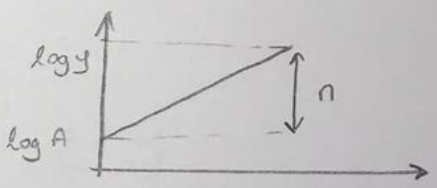
```
for(i=Ø ; i<N ; i++)
    for (j=i+1 ; i <N ; j++)
        for(k=j+1 ; j<N ; k++)
            if( a[i]+a[j]+a[k]==Ø)
                count++;
```

N'in çalışma zamanı T(N)'e göre

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

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

$$\log y = \log(A * x^n)$$
$$= \log(A) + \log(x^n)$$
$$= \log(A) + n \cdot \log x$$



Deneysel Çözümlerde Çalışma Zamanını Etkileyenler:

Donanım : CPU, bellek, cache
Yazılım : Derleyici
Sistem : OS, diğer uygulamalar vb.

) Sisteme bağlı

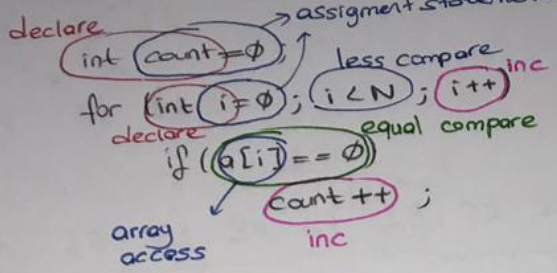
Bağımsız etkenler
- Algoritma
- Data (input)

! Deneysel çözümlerde her türlü donanım ve yazılım olanaklarının aynı olması gerekir.

Calisma Zamanının Matematik Modeli

* Bütün komutların calisma zamanlarının toplamı.

→ Her komutun calisma zamanı aynı varsayılırsa



Her kodu tek tek toplamak gerekiyor.

Calisma Zamanının Teorik Ölcülmesi

Ana Islem: Algoritmanın calisma zamanını en cok alan, en önemli işi belirle ve bu işin kac kez yapıldığını değerlendir.

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

Büyüme Derecesi: N büyüdükce algoritmanın calisma hızı nasıl değişiyor?

Hızı etkileyenler:

1) Giristeki elemen sayısı
2) Bu elemanların hangi düzende olduğu

! Bu etkenleri ölcmek için best, average ve worst senaryolarını hesaplamak gerekir.

Insertion Sort

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

17 21 14

14 17 21

⋮

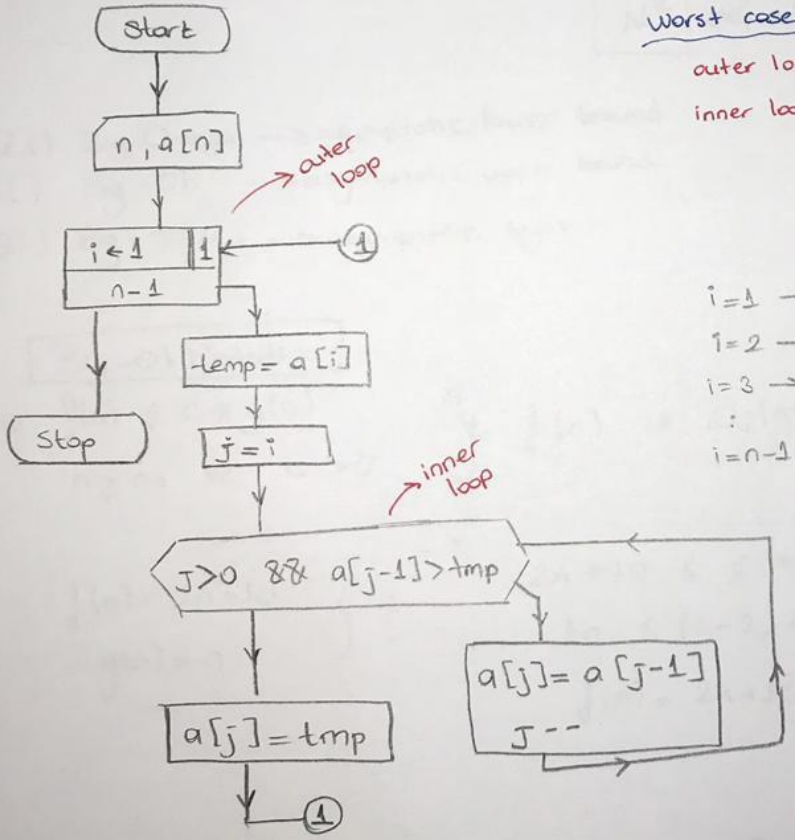
14 17 21 34 47 19 71 22 29 41 32 8

sorted | unsorted

Best Case:

$$\sum_{i=1}^{n-1} 1 = n-1 \sim \boxed{n}$$

best



Worst case:

outer loop: 1 ....(n-1)

inner loop: J=1, J=2, J=3, ⋮, J=n-1 } n-1

i=1 → J=1

i=2 → J=0,1

i=3 → J=0,1,2

⋮

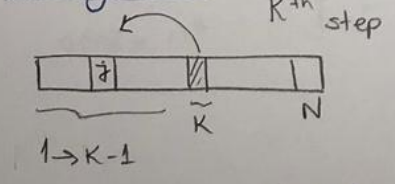
i=n-1 → J = 0,1,2 ..,(n-2)

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

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

worst

Average Case:



K-j+1

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

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

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

Average

Insertion sort için

Best $\approx n-1 \rightarrow n \rightarrow O(n)$

Average $\approx n^2/4 \rightarrow n^2 \rightarrow O(n^2)$

Worst $\approx n^2/2 \rightarrow n^2 \rightarrow O(n^2)$

} İndirgendiğini göstermek için kullanılan notasyon

! n, ∞'a giderken sabitler bizi ilgilendirmez.

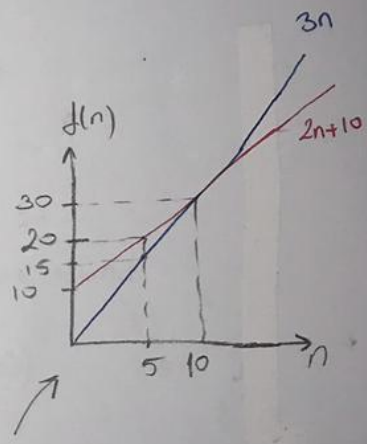
Asymptotic Notation

$\frac{N^2}{4} + \frac{3N}{4} + \frac{2}{4}$ $\xrightarrow{n \to \infty}$ ① Sabitler elenir. $\frac{N^2}{4} + \frac{3N}{4} + \frac{2}{4} \xrightarrow{n \to \infty}$ ② Kat sayılar elenir. $\frac{N^2}{4} + \frac{3N}{4}$ $\xrightarrow{n \to \infty}$ ③ En yüksek dereceli terim alınır. $N^2 + N$ $\rightarrow$ N^2

Ω() Big-Omega → asymptotic lower bound

O() Big-Oh → asymptotic upper bound

Θ() Big-Theta → asymptotic tight



Big-Oh Notation

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

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

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

$f(n) = 2n+10$

$g(n) = n$

} ?

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

$10 \leq (c-2) * n$ $\quad c = 3$ $\quad n = 10$ ✓✓

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

↑

$g(n)$

$f(n) = 2n+10$

$g(n) = n^2$

} ?

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

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

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

↑

$g(n)$

Big-Omega Notation

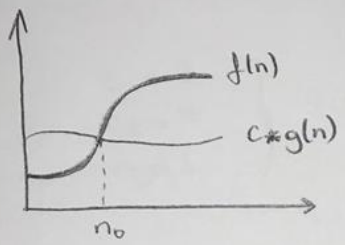
$f(n) \in \Omega(g(n))$ → asymptotic lower bound

if $c>0, n_0>0, n \geq n_0$ } $f(n) \geq c * g(n)$

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

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

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



$5n^2 \geq c * n^2$

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

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

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

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

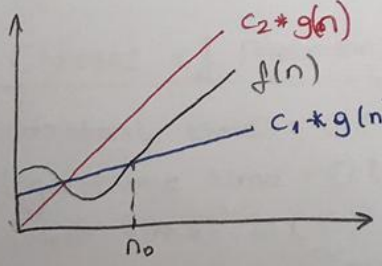
$c>0 \quad n>n_0 \quad n_0>0$

Bg-Theta Notation

$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

$c_1 \leq c_2, n_0>0 \quad n>n_0$

} $f(n) = \Theta(g(n))$ → tight bound



$f(n) \in \Omega(g(n))$
$\in O(g(n))$ } $\Omega(g(n)) = O(g(n))$

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

↳ $\Omega(1)$ diyemem çünkü $\Theta(1)$ değildir.

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

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

$5 \geq (c-2)n$

$n_0 = 1 \quad c_1 = 3$ ✓✓

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

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

$n_0 = 1 \quad c_2 = 7$ ✓✓

$c_1 = 3$
$c_2 = 7$
$n_0 = 1$
} $c_1*n \leq 2n+5 \leq c_2*n \rightarrow f(n) \in \Theta(n)$

Insertion Sort :

worst case $\sim n^2 \rightarrow \Theta(n^2)$
best case $\sim n \rightarrow \Theta(n)$
} insertion sort
$O(n^2) \rightarrow$ max n^2 olabilir
$\Omega(n) \rightarrow$ min n olabilir.
~~Θ~~

Merge Sort :

Best case $\sim n*\log n$
worst case $\sim n*\log n$
} $\Theta(n * \log n)$

Classes of Algorithms :

	example
constant time : $O(1)$	$a = b+c$
logarithmic time : $O(\log n)$	binary search
linear time : $O(n)$	1 Loop
linearthmic time : $O(n*\log n)$	merge sort
quadratic time : $O(n^2)$	double loop
cubic : $O(n^3)$	triple loop
exponantiel : $O(2^n)$	exhaustive search

8

Useful Formulas for the Analysis of Algorithms

Properties of Logarithms

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

Combinatorics

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

Important Summation Formulas

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

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

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

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

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

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

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

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

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

Sum Manipulation Rules

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

Floor and Celling Formulas

$\lfloor 3.8 \rfloor = 3$

$\lfloor -3.8 \rfloor = -4$

$\lceil 3.8 \rceil = 4$

$\lceil -3.8 \rceil = -3$

2. $\lfloor x+n \rfloor = \lfloor x \rfloor + n$ and $\lceil x+n \rceil = \lceil x \rceil + n$ $\Rightarrow$ for reel x and integer n

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

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

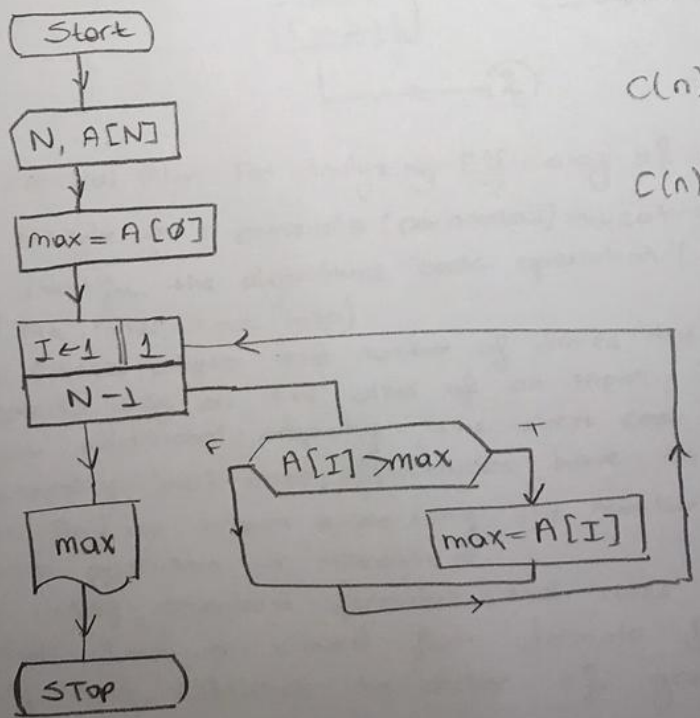
Mathematical Analysis of Non-Recursive Algorithms

Two operations : `A[I] > max` (Basic operation), `max = A[I]`

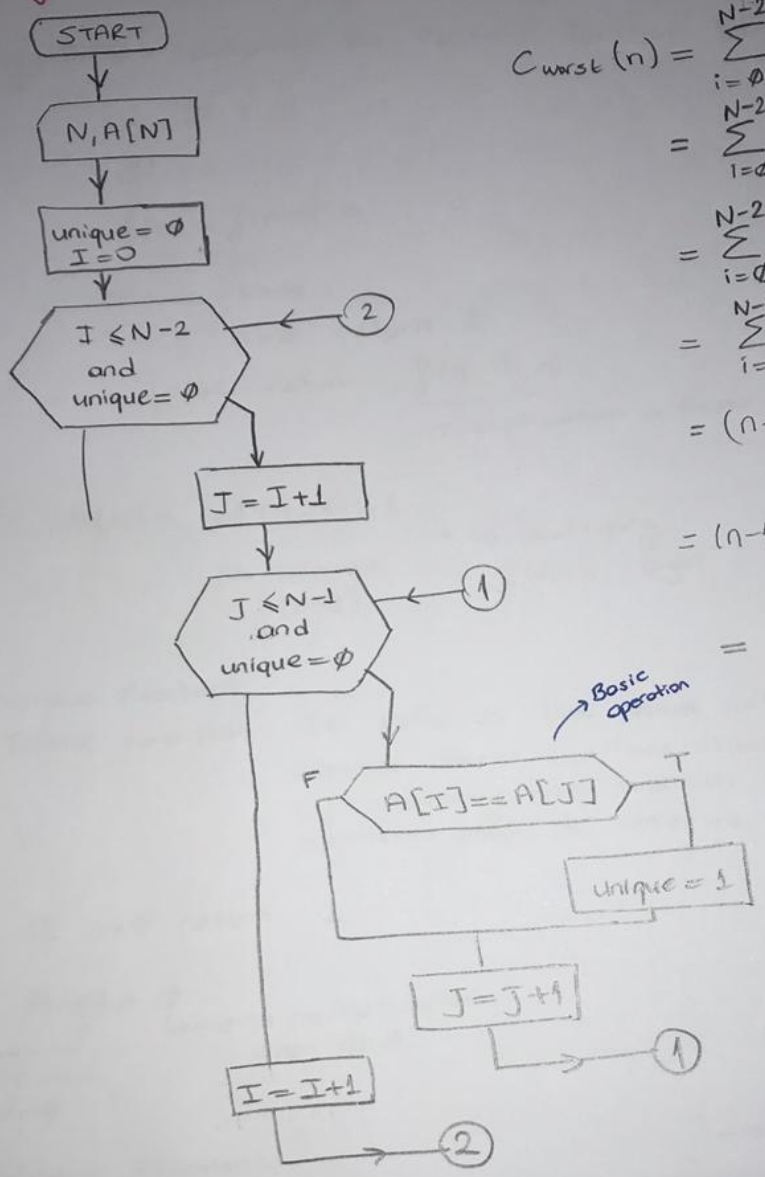
C(n) : the number of times this comperison is executed

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

$\Theta(n)$



Algorithms Element Unique Problem



$$C_{worst}(n) = \sum_{i=\emptyset}^{N-2} \sum_{J=i+1}^{N-1} 1$$

$$= \sum_{i=\emptyset}^{N-2} \left((n-1) - (i+1) + 1 \right)$$

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

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

$$= (n-1) \cdot \sum_{i=\emptyset}^{N-2} i - \frac{(N-2)(N-1)}{2}$$

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

$$= \frac{(n-1)\,n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2)$$

General Plan For Analyzing Efficiency of Non Recursive Algorithms

1. Decide on a parameter (parameters) indicating input's size
2. Identify the algorithms basic operation: (As a rule, it is located in its inner-most loop)
3. Check wheter 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

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

$n! = 1.2.3. \dots n$

$0! = 1$

$f(n) = f(n-1).n$

pseudo-code:

if $n = 0$ return 1

else return $\underbrace{f(n-1).n}$

multiplication = Basic operation !

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

Recursive Relations

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

if $n = 0$ return 1

$M(0) = 0$ → no multiplication when $n = 0$

↑ the calls stop when $n = 0$

Recurrence Relations

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

$M(0) = 0$

Method of Backward Substitution

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

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

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

$\vdots$

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

To take advantage of initial condition,

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

$n = \phi$, $i = n$

$M(\phi) + \phi = \phi$

A General Plan for Analyzing Efficiency of Recursive Algorithms

İlk 3 madde recursive fonksiyonlar ile aynı

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

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

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

$A(n)$: The number of Additions

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

Initial Condition:

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

$n = 2^k \rightarrow A(2^k) = A(2^{k-1}) + 1 \qquad A(2^\phi) = \phi$

Backward Substitution

çözümü

Backward substitution

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

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

$\rightarrow A(2^{k-2}) + 2$

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

$\vdots \qquad \vdots$

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

$\vdots$

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

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

$A(1) = 0$

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

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

Binary Search Algorithm

Input : $A[0 \ldots n-1]$ sorted array

Output : Index or -1

Pseudo Code :

```
l ← 0 ; r ← n-1
while (l <= r) do
    m ← ⌊(l+r)/2⌋
    if K = A[m] return m
    else if K < A[m]  r ← m-1
    else  l ← m+1
return -1
```

index	ø	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
							m						r
iteration 1	l									m			r
iteration 2								l					
iteration 3								l	r				
								m ✓					

C = compare
w = worst case

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

Initial Condition:

$C_w(1) = 1$

$n = 2^k$ → problem 2'ye bölünerek devam ettiği için.

Backward Substitution:

$C_w(2^k) = k + 1 = \log_2 n + 1$ (Bir önceki sayfada aynısı var)

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

even numbers

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

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

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

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

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

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

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

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

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

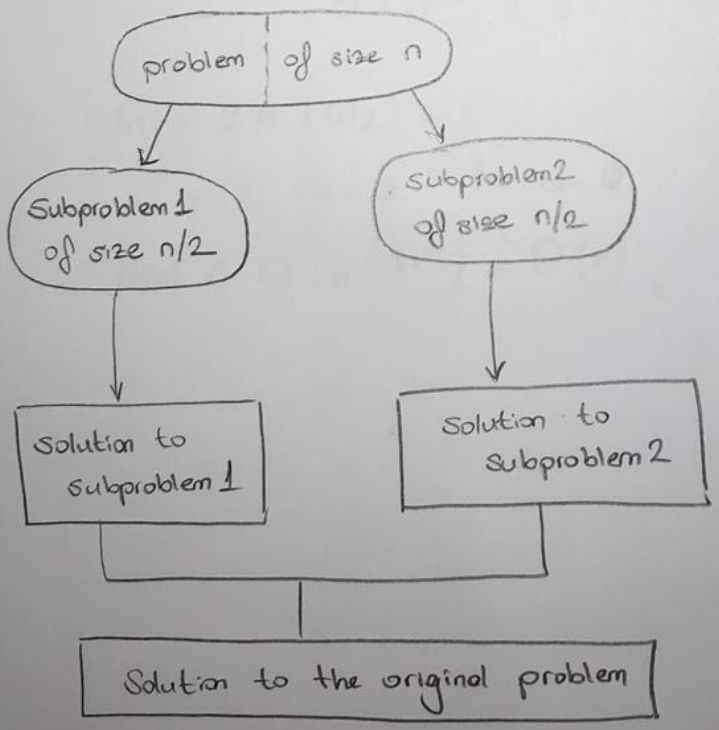
$\lfloor \log_2 10^6 \rfloor + 1 = 20$

Binary search'i recursive çözmeye çalış

Divide and Conquer Algorithms

They work according to the following general plan.

1. A problem's instance is divided into several smaller instances of the same problem, ideally about the same size.
2. The smaller instances are solved (typically recursively though sometimes a different algorithms 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.



İşimize yaramayacak ise alt problemlere ayırmak maliyeti arttırır.

Computing the sum of n

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

instance size : n

subproblems : b

the subproblems that need to be solved : a

$T(n) = a\,T(n/a) + f(n) \Rightarrow$ Running Time [General Divide and Conquer Recurrence]

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

Master Theorem

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

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

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

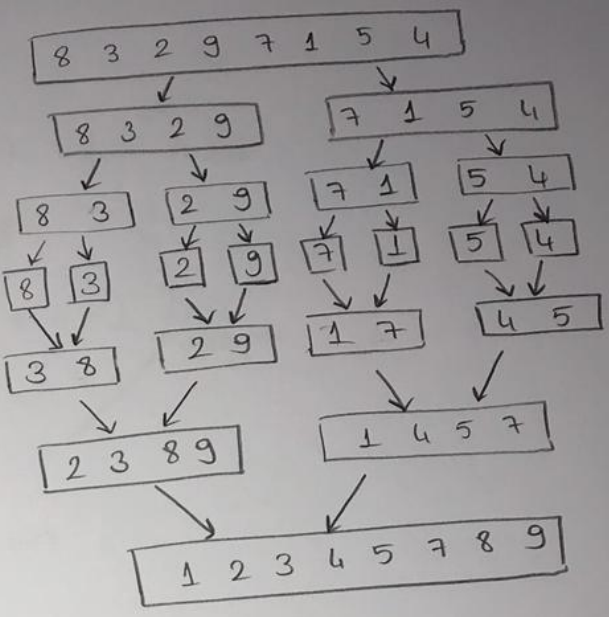
$a = 2 \quad b = 2 \quad \hookrightarrow d = 0$

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

Merge Sort Algorithm

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

```
if (n>1)
    copy A[∅ ... ⌊n/2⌋-1] to B[∅ ... ⌊n/2⌋-1]
    copy A[⌊n/2⌋ ... n-1] to C[∅ ... ⌊n/2⌋-1]
    mergeSort (B[∅ ... ⌊n/2⌋-1]
    mergeSort (C[∅ ... ⌊n/2⌋-1]
    merge (B, C, A)
```



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

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

Partition Algorithm:

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

```
i ← ∅ ; j ← ∅ ; k ← ∅
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

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

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

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

($n/2$ versiyonu)

Backward Substitutions

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

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

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

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

$n = 2^i \rightarrow i = \log_2 n$

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

↓ initial condition'a bağlı olarak

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

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

! Principal Shortcoming : Linear amount of extra storage

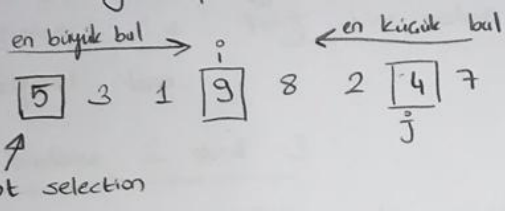
Quick Sort Algorithm

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

$A[0] \dots A[s-1]$ $A[s]$ $A[s+1] \dots A[n-1]$

All are $\leq A[s]$ All are $\geq A[s]$

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



Simplest strategy
$p = A[l]$

pseudo Code :

```
if  l < r
    s ← Partition ( A[ l ... r ])
    Quick Sort ( A [ l ... s-1 ])
    Quick Sort ( A [ s+1 ... r ])
```

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

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

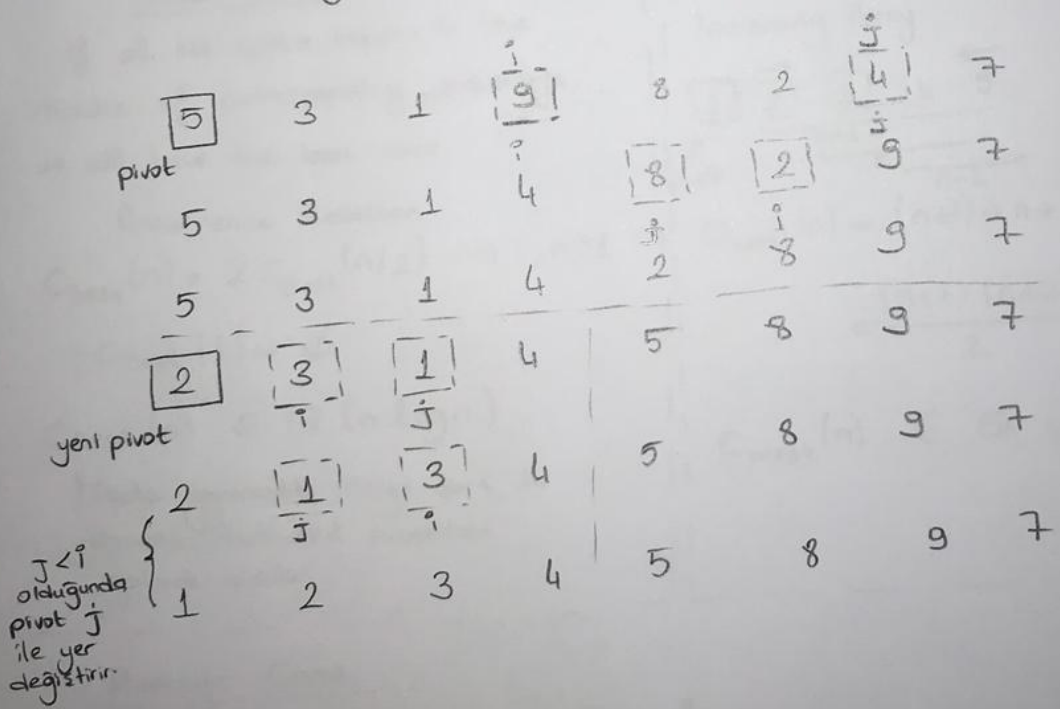
① If scanning indices i and j have not crossed i.e. $i < j$ we simply exchange $A[i]$ and $A[j]$ and resume the scans by incrementing i and decrementing j respectively.

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

③ If the scanning indices stop while pointing to the same element i.e. $i = j$ the value they are pointing to must be equal top.

Combine 2 and 3

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



Partition Algorithm:

```
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 last swap when i ≥ j
swap (A[l], A[j])
return j
```

Best Case

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

Recurrence Relation

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

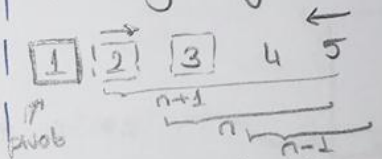
$$C_{Best}(1) = 0$$

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

! Analiz yapmadık. Merge sort'ta yapmıştık. Backward substition yapılarak çözülür.

Worst Case

Increasing Array



A[0 ... n-1]
Already sorted

$$C_{worst}(n) = (n+1) + n + \dots + 3$$

$$= \frac{(n+1)(n+2)}{2} - 3$$

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

Average Case

$$C_{avg}(n) \approx 2n \ln n \approx 1.38\, n \log n$$

Sequential Search:

0	1	2	3	4
3	5	1	4	2

$O(n)$

Binary Search:

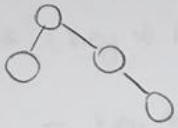
0	1	2	3	4	5	6
2	7	9	12	17	19	22

$O(\log_2 N)$

x=7

Binary Search Tree:

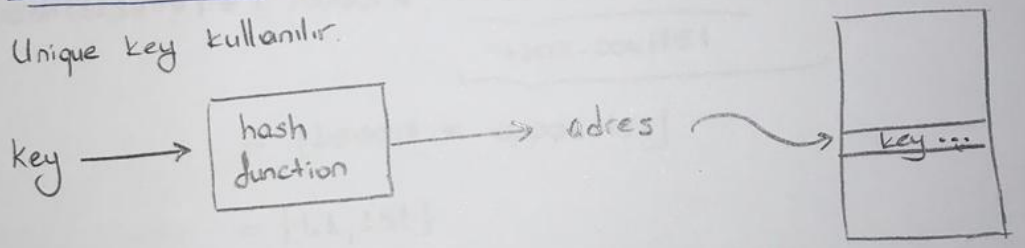
$O(n)$



denge bozulmuş olabilir.

HASHING

Unique key kullanılır.



① Hesabı kolay.

② Tutarlı olmalı.

③ Adresin olabildiğince uniform dağılması sağlanmalı

key mod m ← tablo uzunluğu

$m=10 \rightarrow 2^d$

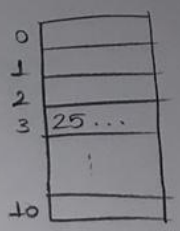
key → 100, 10, 40, 50 → ∅

En çok kullanılan Hash fonksiyonları

① Division Method

$hash(key) = key \bmod m$ (m: tablo uzunluğu)

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



② Multiplication Method

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

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

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

$m = 10000 \quad k = 123456$

$hash(123456) = \lfloor 10000 * (123456 * 0,618033) \bmod 1 \rfloor$

$123456 * 0,618033 = 76300.004151$

$= \lfloor 10000 * 0,004151 \rfloor$

$= \lfloor 41,151 \rfloor$

$hash(123456) = 41$

123456 → [hash] → 41 ← adres

Birthday Paradox

$365 \longrightarrow \frac{365-1}{365}$ → 1. kişinin doğum gününün aynı olmama olasılığı

$365 \longrightarrow \frac{365-1}{365} * \frac{365-2}{365}$ → üç kişinin doğum gününün aynı olmama olasılığı

$365 \longrightarrow \frac{365-1}{365} * \frac{365-2}{365} * \dots * \frac{365-k}{365}$ → k kişinin

Hash map 'te
table size : m

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

← k sayının çakışmama olasılığı

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

Hash collusion probability

Ex : 32 bit hash values

$M = 2^{32}$ ⟶ after 250000 insert % 100
77163 insert %. 50

Key

① Integer
② Floating number
③ String

"ELMA" "mALE"
↑ ↑
aynı sayıyı üretmemeli
Bunu engellemek için
Horner methodu kullanılır.

Horner's Method :

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

L ⇒ lenght of string

str[0] ⇒ string'in sayı karşılığı bulunmalı
str[0] - 'A' + 1

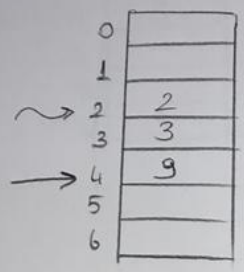
④ Compaund Keys: 31.08.1997
↓
$31 * R + 8 * R^2 + 1997 * R^3$

Collusion Problemi Çözümü

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

1.1 Linear Probing :

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



$M = 7$ $\quad i = 0, \dots, (m-1)$

$key = 9$

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

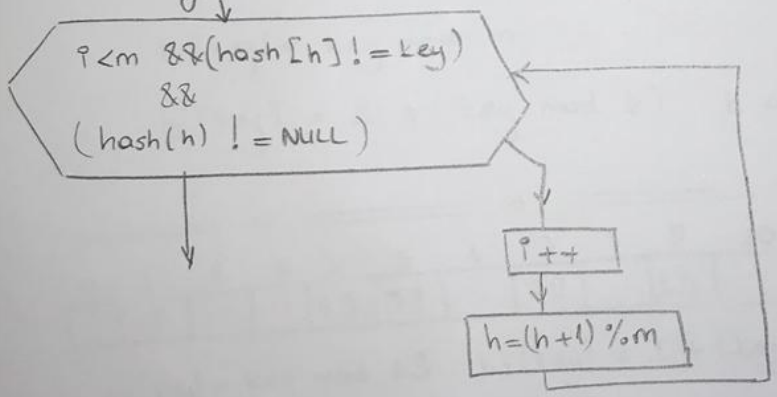
$hash(9, 1) = (9+1) \bmod 7 = 3$ (i=1)

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

Yerleştirilmek istenen yer doluysa boş yer arama işlemidir.

Search

$h = key \bmod m$



Delete

① Silinen eleman yerine bir flag konulmalıdır. Başka bir eleman ararken $hash(h) != NULL$ koşulunu sağlamak için bu yapılmalıdır.

② Silinen elemanın yerini doldurmak için modu aynı olan satırlar kaydırılabilir.

1.2 Quadratic Probing

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

0	
1	
2	2
3	3
4	
5	
6	9

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

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

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

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

$c_1 = 2 \quad c_2 = 1$

$i = 1$

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

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

1.3 Double Hashing

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

(i * $h_2(key)$: adım büyüklüğü)

$h_1(key) = key \bmod m$

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

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

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

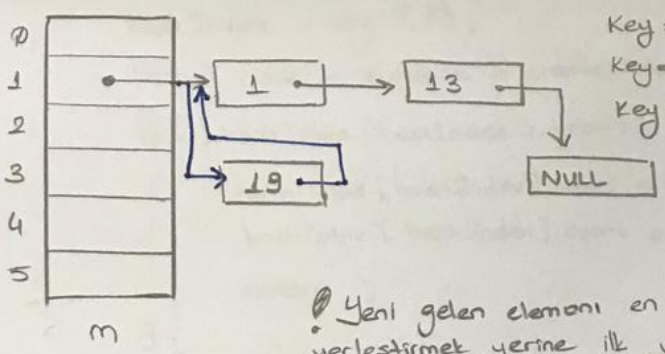
$hash(14) = ?$

$hash(14,0) = h_1(14) = 14 \bmod 13 = 1$ dolu.

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

$i=2 \Rightarrow hash(14,2) = 1 + [\underbrace{1 + 14 \bmod 11}_{4}] * 2 = 9$ boş ✓

(2) Separate Chaining



Key = 13 → 13 % 6 = 1
Key = 1 → 1 % 6 = 1
Key = 19 → 19 % 6 = 1

! Yeni gelen elemanı en sona kadar gidip yerleştirmek yerine ilk yere bağlarız.

$m < N$

maximum $m = N$

m too small → chains too long

$m \sim N/5$ olursa chain'nin çok uzamadığı hesaplanmış.

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

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

struct hash *hashTable = NULL;

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

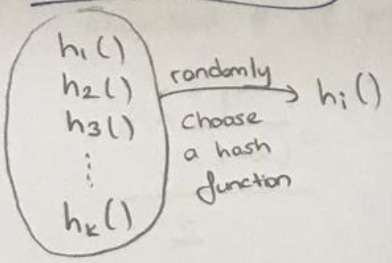
```
void insertToHash (int key, char name[]) {
    int hashIndex;
    hashIndex = key % M;
    struct node *newNode = createNode (key, name);
    if (!hashTable[hashIndex].head){
        hashTable[hashIndex].head = newNode;
        hashTable[hashIndex].count = 1;
        return ;
    }
    newNode ->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

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

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

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

③ Universal Hashing



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

$a, b, p \rightarrow$ 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

$\alpha \Rightarrow$ genellikle 0.5 alinir.
yer ve zaman dengesi saglanir.

α	# of probe (average)
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	50.5

Unsuccessful 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 :

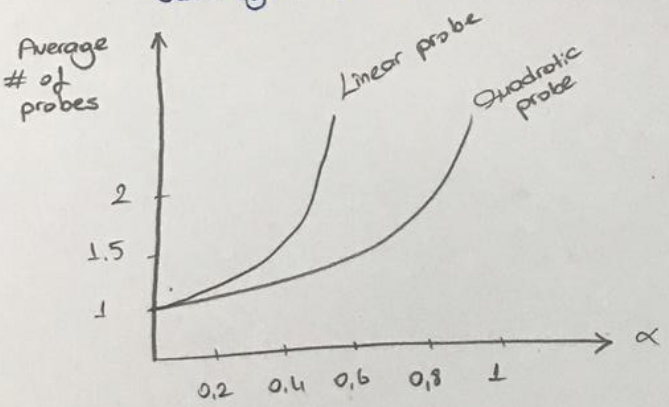
$\sum_{k=1}^{m} k * \alpha^{k-1} * (1-\alpha) \Rightarrow$ geometric series $= \frac{1}{1-\alpha}$

insertion :

expected # of probes

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

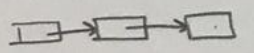
Successfull Search



Separate Chaining

① Unsuccessful Search

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



worst case : Bütün elemanlar aynı adreste

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

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

$\Theta(1+\alpha)$

→ search

→ hash function

$\alpha = \frac{N}{m}$ → sabit

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

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

tableSize = 8

hash(key) = x mod 8

64, 128, 200, 400 → 0

100, 300, 500 → 4

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

Dynammic Programming → Planning Richard Bellman (1950's)

Fibonacci Numbers

0 1 1 2 3 5

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

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

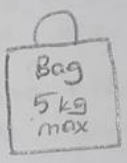
fibonacci

0-1 Knapsack Problem

→ abandon
→ select

Hırsız, yükte hafif pahada ağır şeyleri almaya çalışıyor.

Brute Force : 2^n



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

W = 5 kg max

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

item \ weight	∅	1	2	3	4	5
∅	∅	∅	∅	∅	∅	∅
1	∅	∅	12	12	12	12
2	∅	10	12	22	22	22
3	∅	10	12	22	30	32
4	∅	10	15	25	30	37

max kazanç

Amacımız:

maximize ΣV_i subject to W

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

case 1 : thief takes item i

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

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

$w_i = 2$ ↙ tablo ile alakası yok

eşya ↔ 3kg'ye kadar kazanç

$w_3 = 2$

Case 2 : NOT takes item i

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

$w_1 = 2$ $P(1,1) = \emptyset$ $2 > 1$

$P(1,2) = \max\{12+0, \emptyset\} = 12$

$P(1,3) = \max\{P(0,1)+12, P(0,3)\}$

$w_2 = 1$ $P(2,1) = \max\{P(1,0)+10, P(1,1)\}$

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

$P(2,3) = \max\{P(1,2)+10, P(1,3)\}$
12+10 tl ; 12 tl

$P(3,3) = \max\{P(2,\emptyset)+20, P(2,3)\}$
0+20 tl ; 22 tl

$P(4,3) = \max\{P(2,1)+20, P(2,4)\}$
10+20 tl ; 22 tl

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

initialization

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

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

Hangi eşyaları aldığımızı nasıl buluruz?

	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	✗
4	0	10	15	25	30	37	✓ $w_4=2$

Reconstructing Optimal Solution

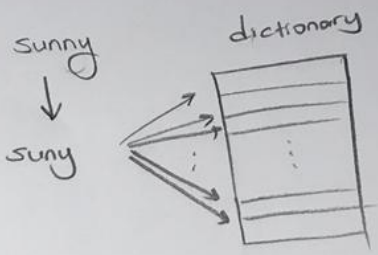
Not taken : ↑

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

taken : ←

Edit Distance

Mesaj yazarken yanlış yazılan kelimeyi düzeltme önerileri



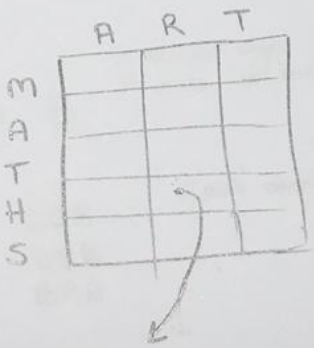
İlk farklı durum var

dist(MATHS, ARTS)

① sunny
Sumny

② sunny
suny

③ sunny
sunnyy



MATHS
_ARTS

MATHS
ART_S

ED[i,j]:

X[i] y[j] gelene kadarki min mesafe

X[1...m] y[1...n]

① copy $x[i] \rightarrow y[j]$
$ED[i,j] = ED[i-1, j-1]$

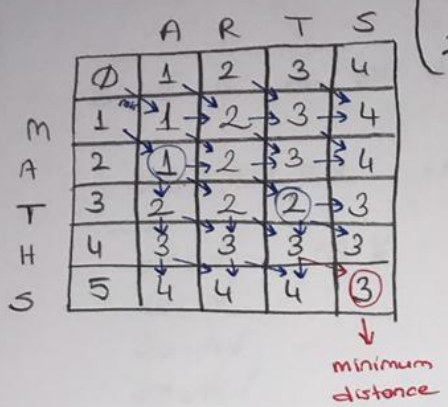
② change $x[i] \rightarrow y[j]$
$ED[i,j] = ED[i-1, j-1] + cost$

③ insert $x[i] \rightarrow$ —
$ED[i,j] = ED[i-1, j] + cost$

④ delete — $\rightarrow y[j]$
$ED[i,j] = ED[i, j-1] + cost$

Recurrence Relation

$$ED[i,j] = \min \begin{cases} ED[i-1, j-1] + \emptyset & : \text{copy} \\ ED[i-1, j-1] + cost & : \text{change} \quad \text{if}(x[i] \neq y[j]) \\ ED[i-1, j] + cost & : \text{insert} \quad \text{if}(x[i] \neq y[j]) \\ ED[i, j-1] + cost & : \text{delete} \quad \text{if}(x[i] \neq y[j]) \end{cases}$$



Longest Common Subsequence

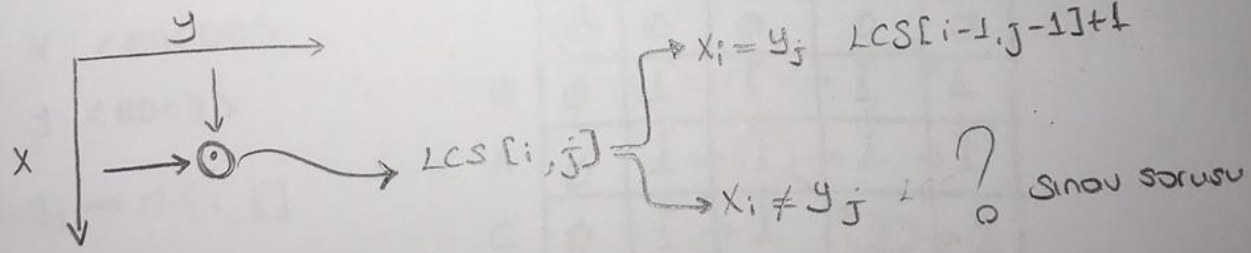
M A R K A

A R T K S A B

↓ içinde aynı sıra ile bulundurması

DNA dizilimlerinde kullanılır.

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



10.hafta / 29 Kasım

Longest Common Subsequence

$\langle A, C, G, T \rangle$ → DNA dizilimleri

$X = ATCTGAT$

$y = TGCATA$

) Longest Common Subsequence of X and y → $LCS(X,y)_1 = TCTA$, $LCS(X,y) = TGAT$

Ortak en uzun alt dizilim nedir? Bu soruya cevap arayacağız.

KOZALAK kelimesinin subsequence 'lerini bulun.

→

KALV AZ X

KOLV

KOLAV

KALK ✓

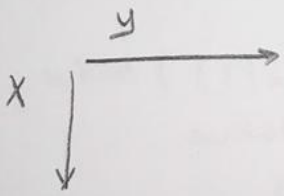
$X = \langle x_1, x_2, x_3, \ldots, x_m \rangle$

$y = \langle y_1, y_2, \ldots, y_n \rangle$

} LCS $S = \langle s_1, s_2, \ldots, s_k \rangle$

$s_{i_1}, s_{i_2}, \ldots s_{i_k}$

$i_1 < i_2 < \ldots < i_k$



Matris yapısı kullanırsa karşılaştırma işlemi daha rahat olur.

X : <BACDB>

y : <BDCB>

$X_i y_j \rightarrow c[i,j]$

$X_m y_n$ → tamamının uzunluğunu bulcaz.

Geriye giderek neler olduğumuza bakıcaz.

		B	D	C	B
	∅	∅	∅	∅	∅
B	∅	1	1	1	1
A	∅	1	1	1	1
C	∅	1	1	2	2
D	∅	1	2	2	2
B	∅	1	2	2	3

Initial

if $i = \phi \longrightarrow c[\phi, j] = \phi$

if $j = \phi \longrightarrow c[i, \phi] = \phi$

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

ADD : ϕ
SKIPX : 1
SKIP y 2

$X_i = Y_j \longrightarrow B[i,j] = \text{ADD}$

$X_i \neq Y_j \longrightarrow \langle [i-1, j] \rangle = C[i, j-1]$

then
X[i] not in LCS
$B[i,j] = \text{SKIP X}$

else
$B[i,j] = \text{SKIP Y}$
Y[j] is not in LCS

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

Vize - 1 Soruları

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

N elemanlı dizi → 1 saat

2N elemanlı dizi → ? saat

Bellek ve çalışma hızının → x2 olduğu ortamda çalıştırılacak.

ilgilendiğimiz kısım → N^2

N yerine 2N koyarsak $(2N)^2 = 4N^2$ yani normal şartlarda 4 saat sürerdi.

Çalışma hızı x2 olduğu için 2 saat sürecektir.

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

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

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

$\downarrow$

$28 \quad n=1 \quad \surd\surd$

$28 \leq 28$

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

Time complexity

- Best case
- Average Case
- Worst Case

Space Complexity

- Therotical Analysis
- Empirical Analysis

4) a) Bauble sort algoritmasının aşağıda verilen sözde kod parçasına göre karmaşıklık analizini yapın. (wrst-case için)

b) Bouble sort algoritmasının best case durumu için algoritmasını O(n) olacak şekilde yazınız.

Best – case ⟶ already sorted

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

5) $n(n+5)$ ve $5000n^2$ Eşit

2^{n-1} ve 2^{n+1} Eşit

$\log_2(n)$ ve $\ln(n)$ Eşit

$1000n^2$ ve $0.001n^3$ Küçük

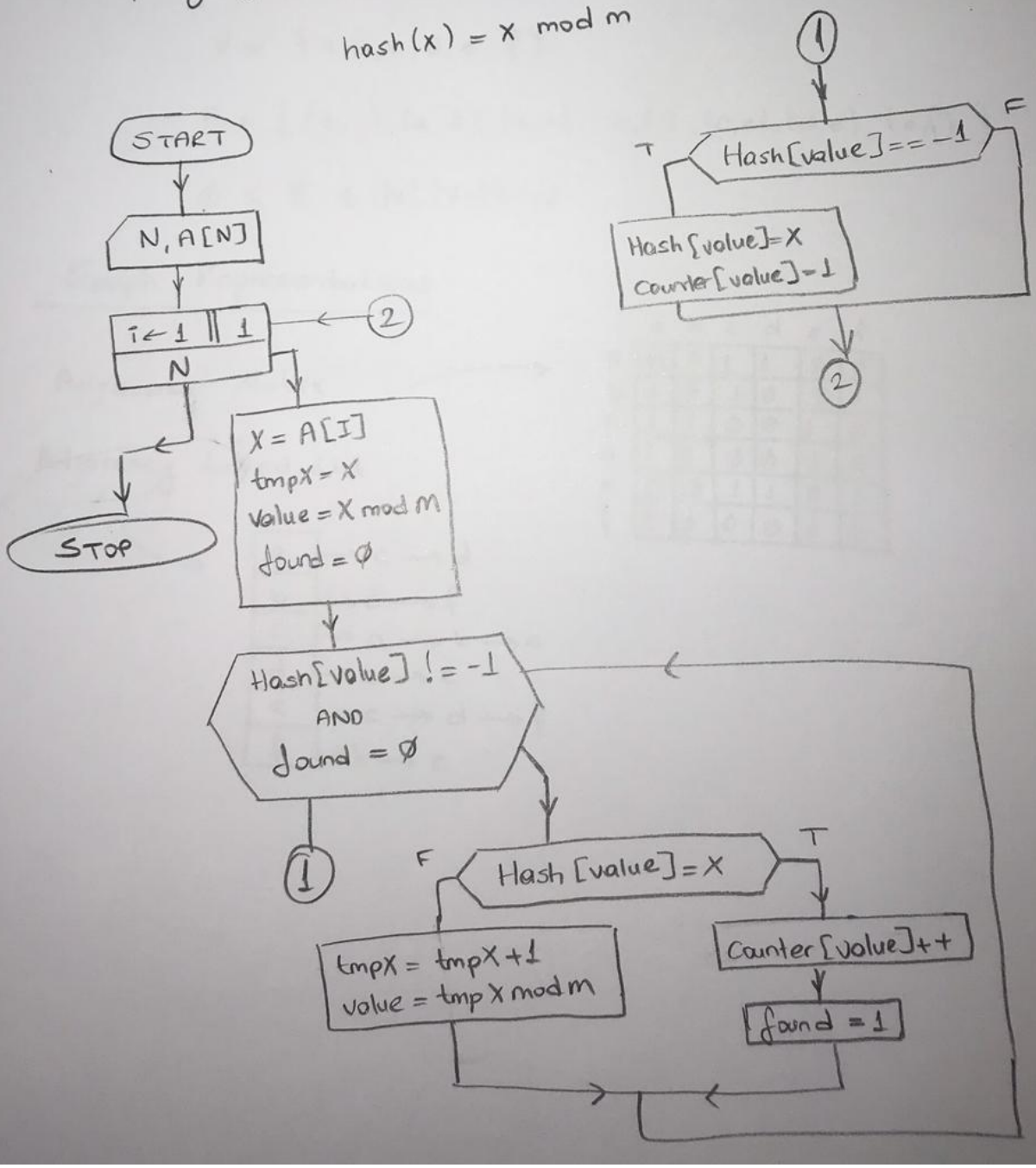
$(n-5)!$ ve 2^n Büyük

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

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

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

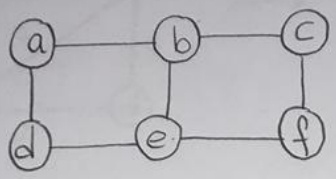
$$hash(x) = x \bmod m$$



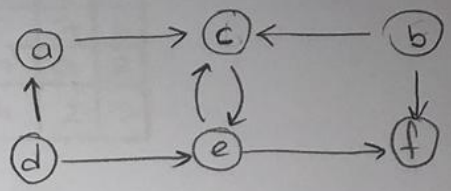
06 Aralık 2018

Graph

→ $G = \langle V, E \rangle$



Undirected



Directed

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

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

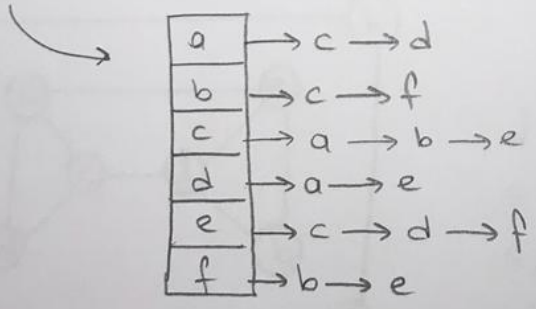
$\emptyset \leq E \leq (|V|\,|V-1|)/2$

Graph Representations

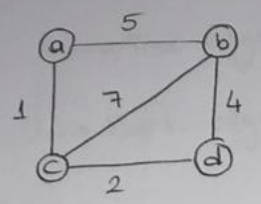
Adjacency Matrix →

	a	b	c	d	e	f
a	∅	∅	1	1	∅	∅
b	∅	∅	1	∅	∅	1
c	1	1	∅	∅	1	∅
d	1	∅	∅	∅	1	∅
e	∅	∅	1	1	∅	1
f	∅	1	∅	∅	1	∅

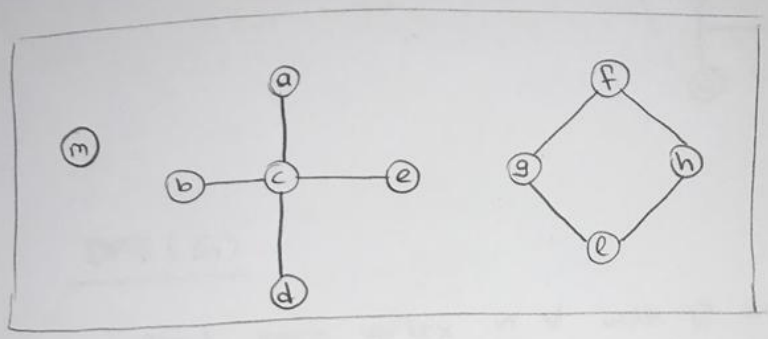
Adjacency Linked List



Weighted Graph



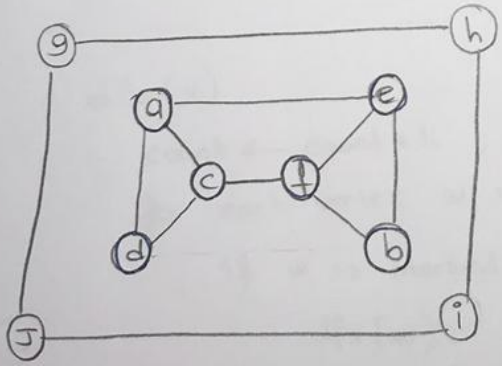
	a	b	c	d
a	0	5	1	∞
b	5	0	7	4
c	1	7	0	2
d	∞	4	2	0



connected component durumu

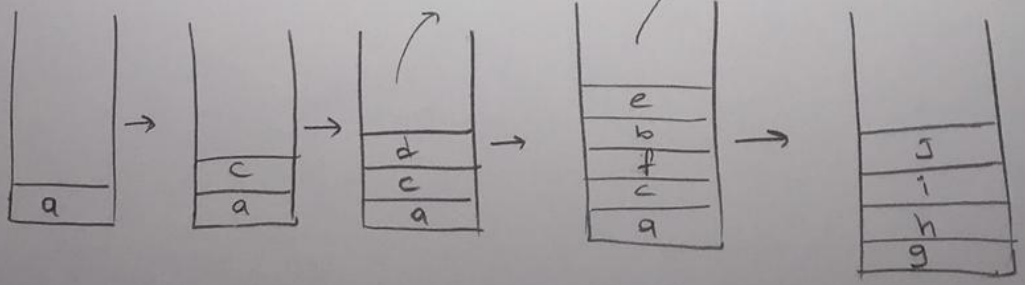
Dept - First - Search

Decrease-by-one technique

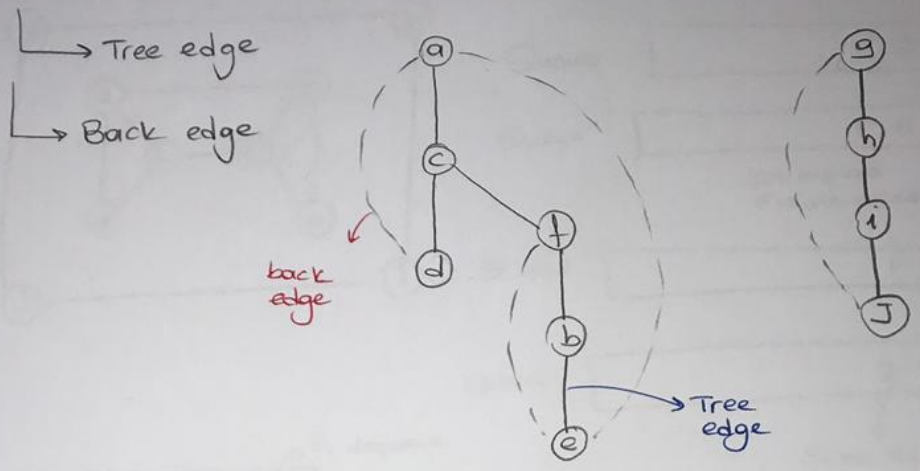


e 6,2
b 5,3
f 4,4
d 3,1
c 2,5
a 1,6
J 10,7
i 9,8
h 8,9
g 7,10

Stack → LIFO



Dept-First-Search Forest



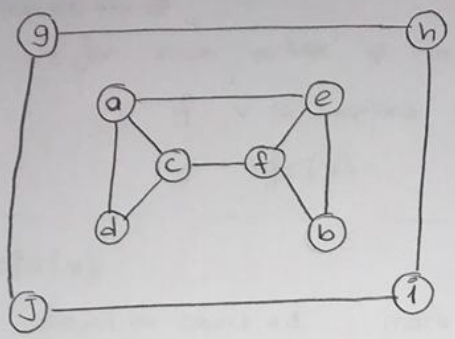
DFS (G)

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)

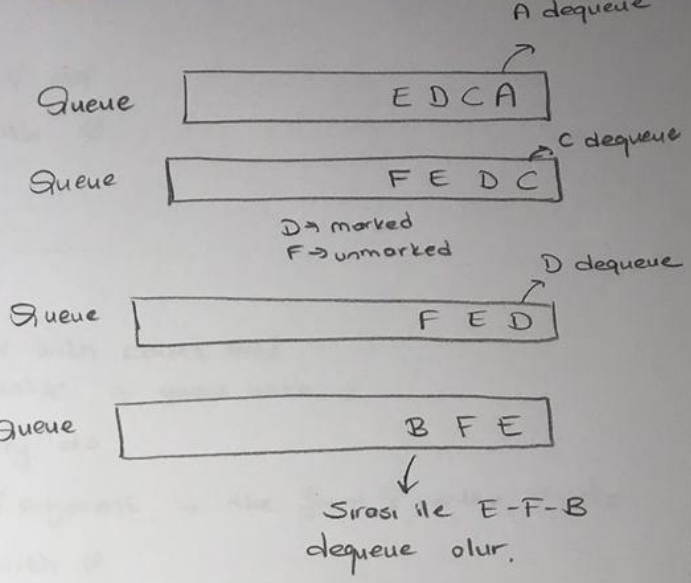
---

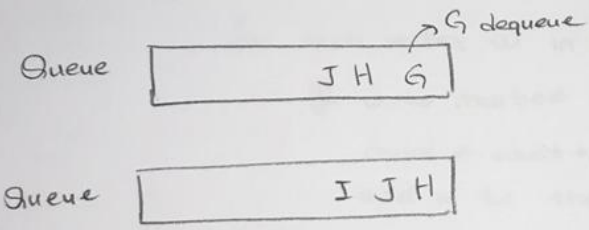
dfs(v)
    count ← count+1 ; mark v with count
    for each vertex w in V adjacent to do
        if w is marked with ∅
            dfs(w)

Breadth-First-Search



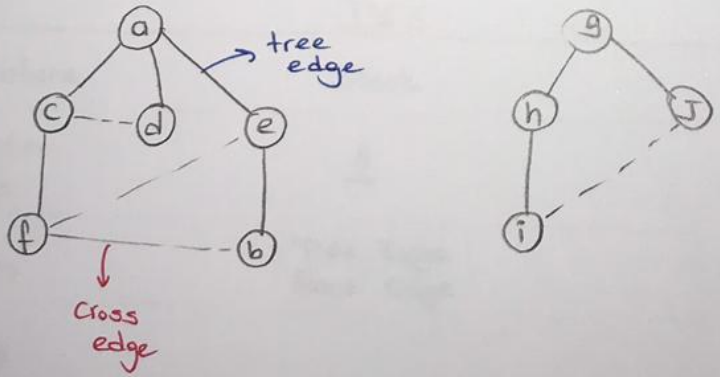
Queue → FIFO





Sirasi ile H-J-I dequeue olur.

Breadth-First-Search Forest



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)

---

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 ∅

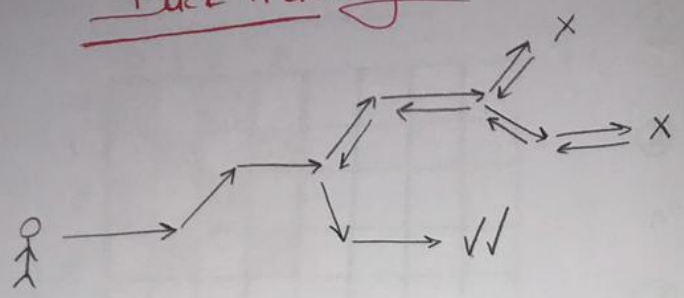
count ← count + 1 ; mark w with count

add w to the queue

remove vertex v from the front of the queue

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

Back Tracking



Aslında yaptığımız şey DFS
Ancak DFS'nin iyileştirilmiş halidir.

= DFS with pruning =

= Stack ya da recursion kullanılır =

Subset Sum Problem

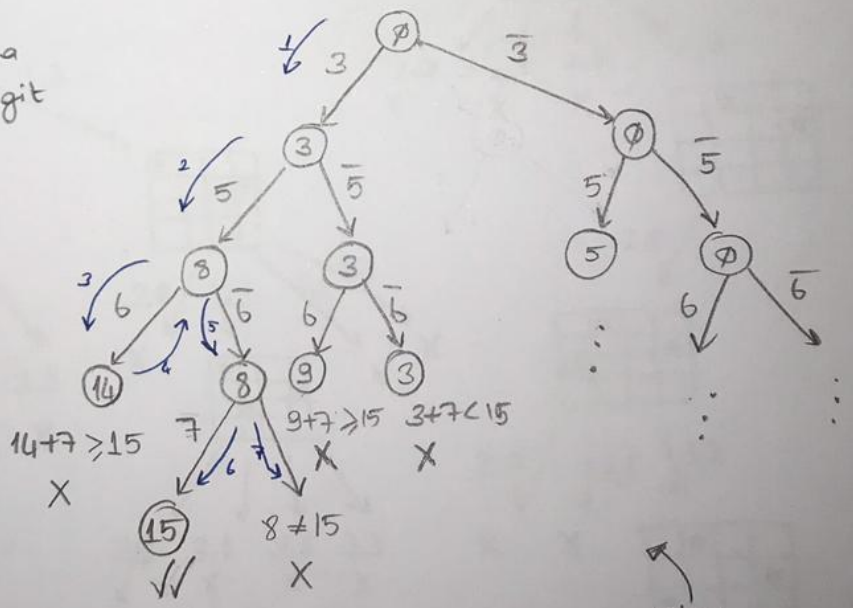
$S = \{ s_1, s_2, s_3 \dots s_n \}$ → Pozitif tam sayılardan oluşan sample'lar

$s_1 \leq s_2 \leq s_3 \dots \leq s_n$

Toplamı d eden alt kümeleri bulma problemi

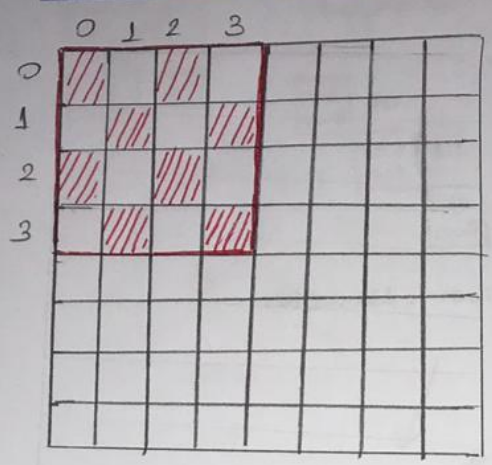
Ex/ $S = \{3, 5, 6, 7\}$ $d = 15$

- En küçük sayıdan başla üzerine ekleye ekleye git



state space tree deniyor

8 Vezir Problemi



① $\binom{64}{8} = \frac{64!}{8!(64-8)!} = 4\,426\,165\,368$

② One per row

$n^n \Rightarrow 8^8 \Rightarrow 16\,777\,216$

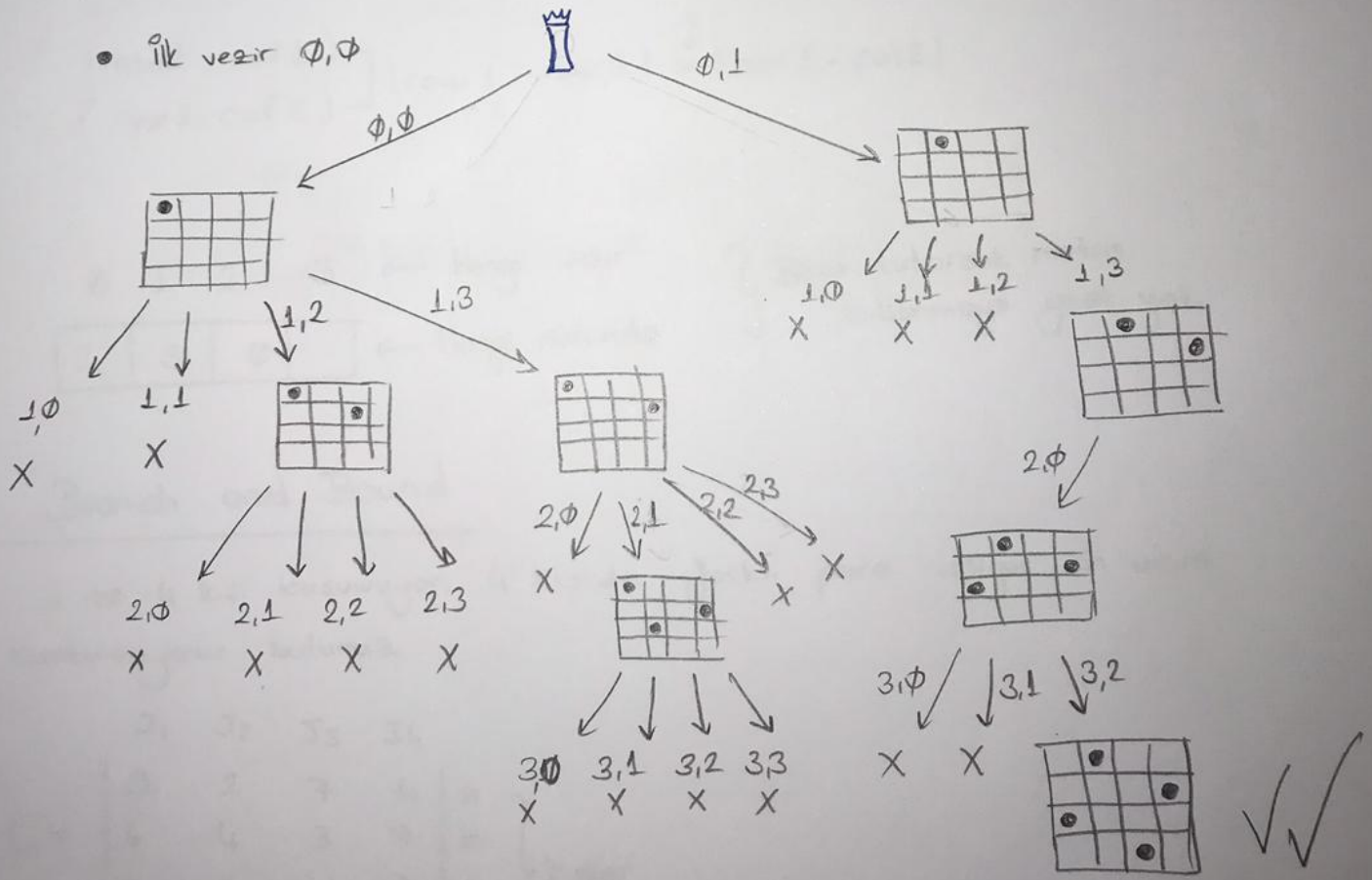
③ One per row and per column

$n! = 8! = 40\,320$

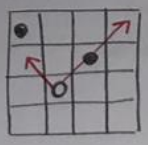
④ Backtraking → 2057 adım 8x8 için

! 8x8 lik zor olcagından 4x4 lük yapılacak derste.

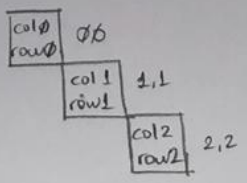
- İlk vezir Ø,Ø

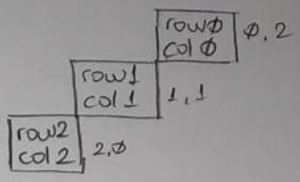


● → koyulmuş vezir
○ → koyulacak vezir



Diagonaldeki kontrolleri nasıl yaparız?





$col2 - col1 = row2 - row1$

$\underbrace{col0 - col2}_{2} = \underbrace{row0 - row2}_{2}$

$\underbrace{col1 - col2}_{1} = \underbrace{row1 - row2}_{-1}$

$$|col_i - col_j| = |row_i - row_j|$$

$\left.\begin{matrix}(row1, col1)\\(row2, col2)\end{matrix}\right] |row1 - row2| \overset{?}{=} |col1 - col2|$

0 1 2 3 ← Hangi vezir

2	3	0	

← Hangi sütunda

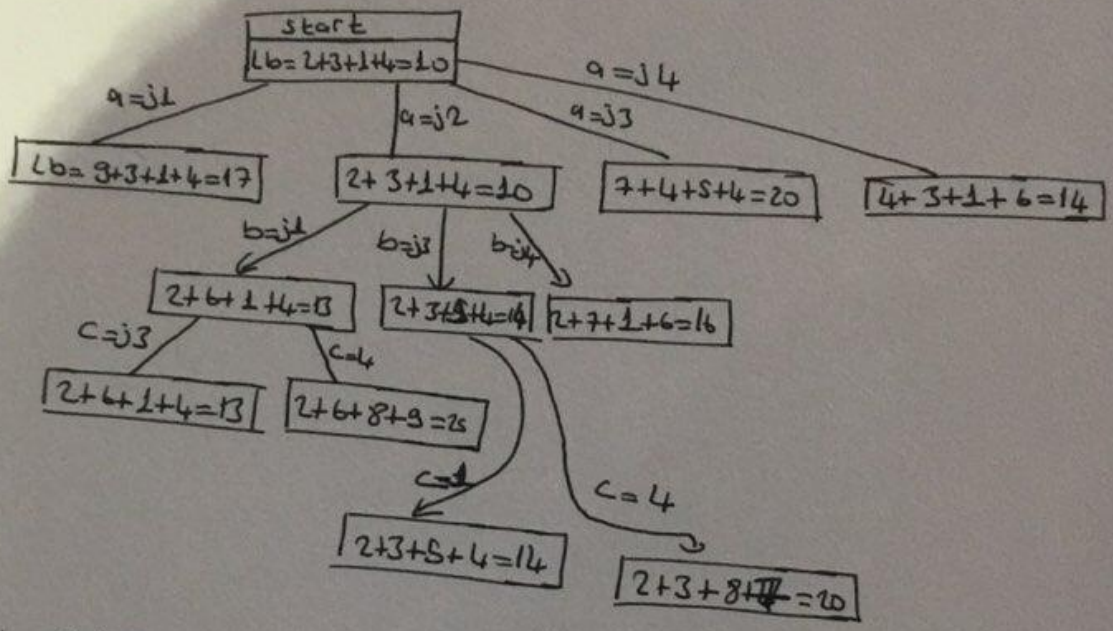
} Böyle tutarsak matris kullanmaya gerek yok.

Branch and Bound

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

$$C = \begin{array}{c|cccc|c} & J_1 & J_2 & J_3 & J_4 & \\ \hline & 9 & 2 & 7 & 4 & a \\ & 6 & 4 & 3 & 7 & b \\ & 5 & 8 & 1 & 8 & c \\ & 7 & 6 & 9 & 4 & d \\ \hline & \multicolumn{4}{c}{\text{işler}} & \end{array} \quad \} \text{kişiler}$$

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



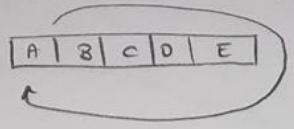
Ilk olarak job fark etmeksizin en küçük sayıları topla 2+3+1+4=10 a kişisi için bütün joblar için yeni sonuçlar bul. A=j3 olduğunda demek oluyor ki diğerleri j3'ü alamaz. Diğerlerinin çakışıp çakışmadığını kontrol etmeden en düşük 2. fiyatını alıyoruz. A=j2 en düşük sonuç ordan devam edip aynı işlemleri b kişisi için yapıyoruz. b j1 için 13, b j3 seçerse a=2 böyle olmak zorunda. b=3 böyle böyle devam ediyoruz. c=5 d=4

Priority Queue

Queue

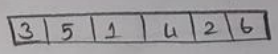
First In First Out

A ← B ← C ← D



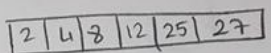
* array
* linked list

① Unsorted Array



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

② Sorted Array



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

↓

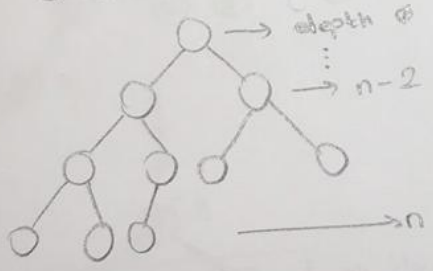
circle array ya da linked list kullanilirsa

! Max. kazanc için nasil bir yapi kullanabiliriz?

— Heap Tree yapisi kullanabiliriz.

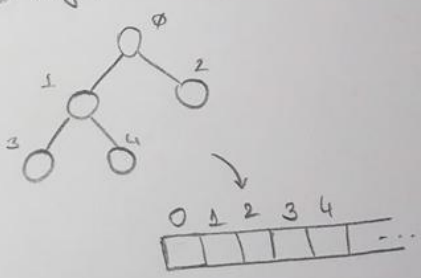
Heap Tree

(a) Balanced



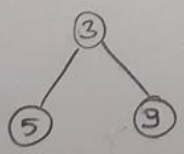
All the nodes at depth ∅ through (n-2) have two children!

(b) Left Justified

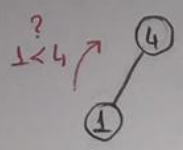


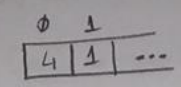
① Complete Binary Tree

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

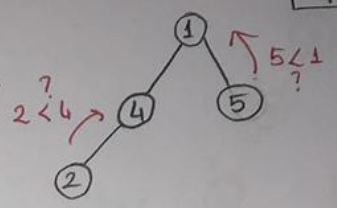


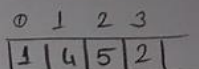
Heapify





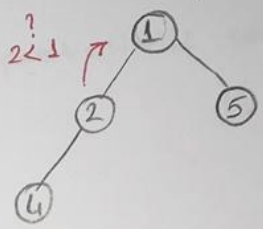
yer değiştirme yapılır.

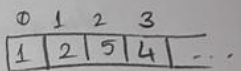




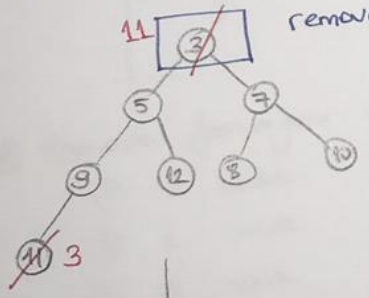
yer değiştirilir.

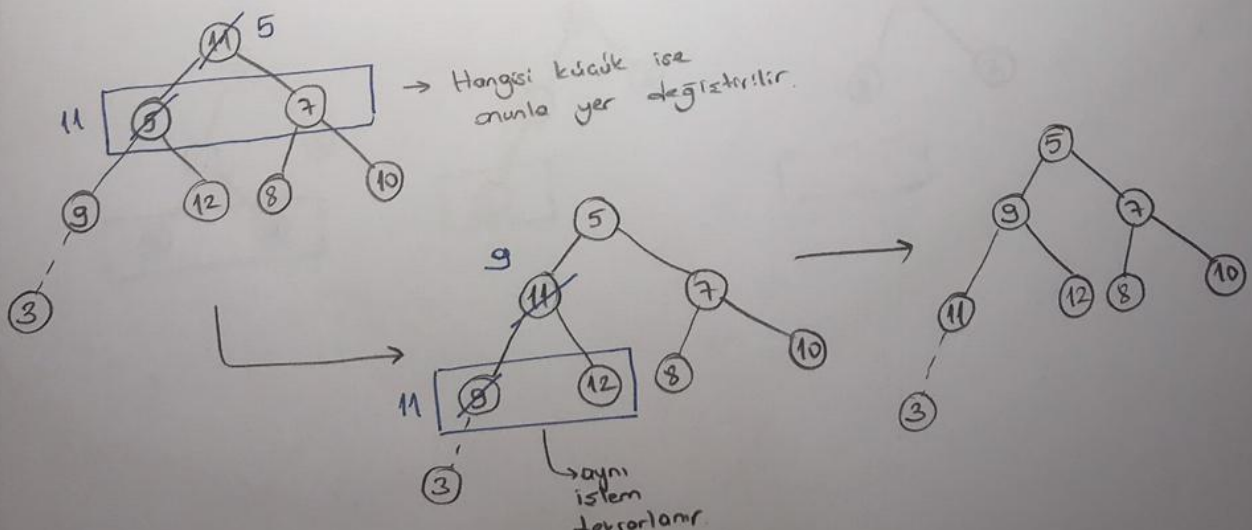
Up heap işlemi denir.
Bu işlemi insert için kullanabilirim.
$O(\log_2 n)$



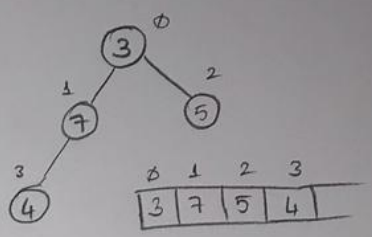


Hizmet verileni (kök değeri) en sondaki elemanla yer değiştiririz.
Sırasında down heap ile yerleştiririz.





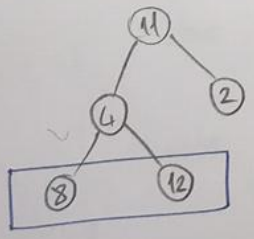
```
Void Insert ( int newItem, int n, int pQueue[]) {
    pQueue[n] = newItem;
    int i=n, stop=0;
    n++;
    while ( (i>0) && (!stop)) {
        if ( pQueue[i] <= pQueue[(i-1)/2]) {
            pQueue[i] ⟷ pQueue[(i-1)/2];
            // swap işlemi
            i = (i-1)/2 ; // parent = (i-1)/2 tanımlanabilir.
        }
        else {
            stop = 1;
        }
    }
}
```

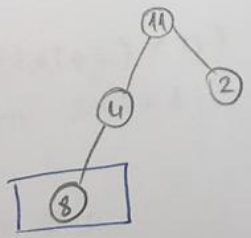


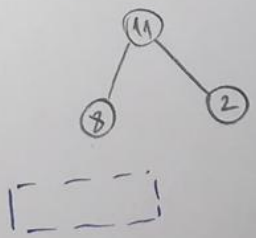
(*) Remove fonksiyonu için gerekli kontroller

① 2 çocuk
② 1 çocuk
③ 0 çocuk

} Bu durumlara göre işlem yapılmalıdır.







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

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