4 ödev — 3 Lab — 1 Proje — 1 Final — 2 Vize
(%14 %6 (%40 (%40

İyi algoritma -> Zaman ve hafıza'yı iyi kullanır, adapte edilebilir, Basittir, Doğrudur, Optimum, düşük bağımlılık

Zaman ölçme

begin = clock()
*/
/*
end = clock()

timeSpent = (double) (end begin) / CLOCKS_PER_SEC

} Empirik analiz

Sistem bağımlı etkiler -> yazılım, donanım, işletim sistemi, internet vs.
" bağımsız " -> Algoritma -> Teorik analiz

2. Hafta

switch vs if → az elemanda daha hızlı.
↓
çok eleman olursa lookup table oluşturuyor ve hızlı oluyor
↳ hashle

Anlysis of Asymptotic Analysis Older of growth $n \to \infty$
↓
düşük değerlerde çok farketmez

Basic operation → En karmaşık işlem

```
for (i=0; i<n; i++){
  if(a[i]==0)
    count++;
}
```

- i=0 → assignment 1
- i<n → Less then compare n+1 times
- ==0 → equal to compare n //
- a[i] → array access n
- count++ → increment min n, max 2n (i++, count++)

Toplam 4n+2 min, 5n+2 max

$5n^2+3n+1$
↓
① ignore constant terms → $5n^2+3n$ → not 10^{10} gibi bir sayı varsa onu da atma!!! → bunu görmezden gelme!!!
② ignore coefficients → n^2+n → not $m * n$
③ pick the most significant term → n^2

Asymptotic Notation $n \to \infty$

$O()$: Big O → max

$\Theta()$: Big theta → average

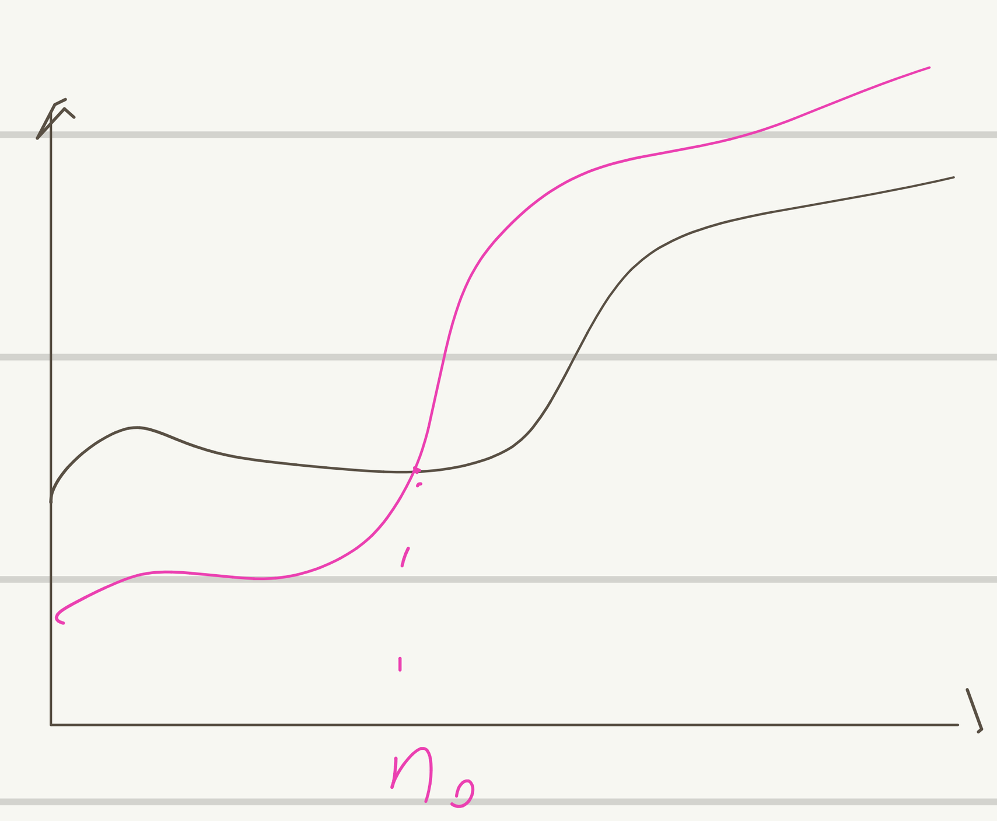
$\Omega()$: Big Omega → min

① Big-Oh notation $f(n) = 10n+30 \to O(n)$

$f(n) \in O(g(n))$ $f(n)$ şöyle yazılabilir ama ne zaman

if $0 \leq f(n) \leq c * g(n)$ for all $c > 0$, $n \geq n_0 > 0$

then $O(g(n))$: upper bound on the growth of a function $f(n)$



$0 \leq 10n+30 \leq c * g(n) \to c * n$

$c = 11$

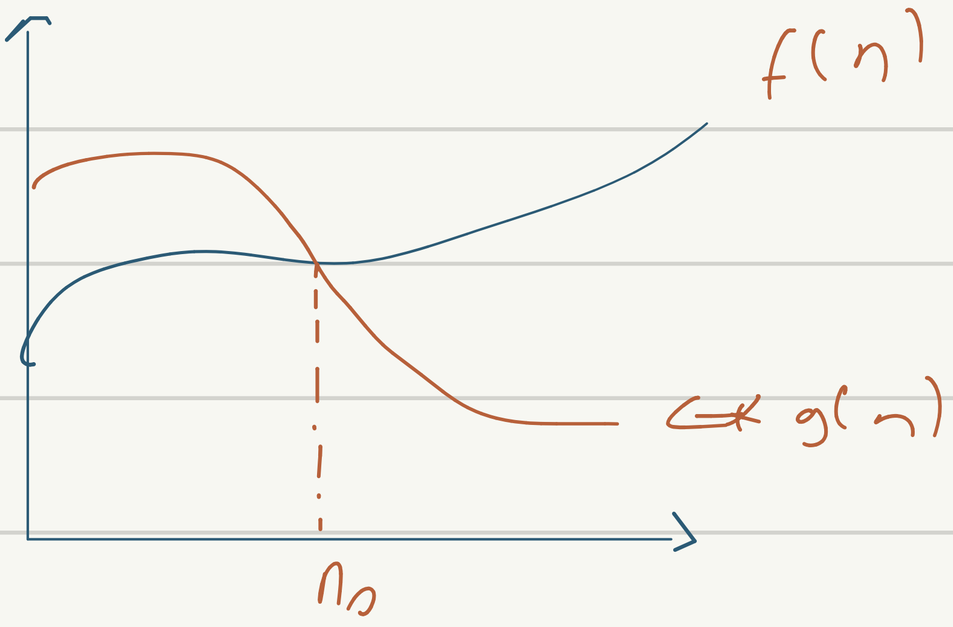
$30 \leq (c-10) * n$ $\quad \frac{30}{c-10} \leq n$ $\quad n = 30$ sağladı

not n^2, n^3, n^4, n^5 nden yüksek her şey olur

(2) Big Omega notation lower bound

If $f(n) \geq c * g(n) \geq$ for all $c > 0$ $n \geq n_0 \geq 0$

then $f(n) \rightarrow \Omega(g(n))$



$5n^2 \in \Omega(n)$

$5n^2 \geq n * c \geq 0$

$n_0 = 1 \rightarrow c = 3$

$5 \geq c \geq 0$

$4n^2 \in \Omega(n^2)$

$4n^2 \geq c * n^2$

$c = 3$

$n_0 = 1$

n_0 en düşük olmak zorunda değil

(3) Big Θ tight bound

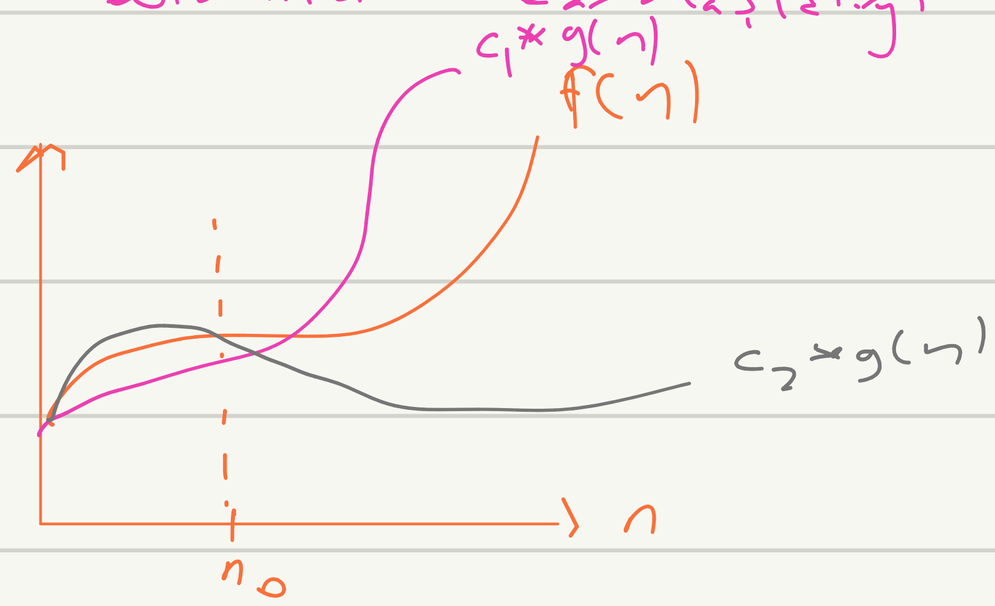
Bir dizinin en lüzüklü elemanını bulmanın karmaşıklığı

worst $\rightarrow N$ best $\rightarrow N$

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

for all $c_2 \leq c_1$ $c_1 > 0, c_2 > 0$

then $f(n) \ \Theta(g(n))$



f(n) is asymtbticly equal to g(n) $\rightarrow f(n) \in O(g(n))$ $\Theta(g(n))$

$f(n) \in \Omega(g(n))$

$f(n) = 2n+5 \rightarrow \Theta(n)$

$n_0 = 1$ $c_1 = 7$ $2n+5 \leq c_1 \cdot g(n) \rightarrow O(n)$

$n_0 = 1$ $c_1 = 1$ $2n+5 \geq c_1 \cdot g(n) \rightarrow \Omega(n)$

$7 > 1$

3. hafta

① Rekürsif olmayan algoritmaların analizi

```
int findMax(int n, int *A) {
    int i, max;
    max = A[0];
    for(i=1; i<n; i++){
        if(max<A[i])
            max=A[i]; }
    return max; }
```

(1) Algoritmanın basit operasyonunu belirle. → n

(2) Ω, Θ, O'sı n için belirle.

basit operasyon → karşılaştırma sayısı = $\sum_{i=1}^{n-1} 1 = n-1 \Rightarrow n$

$C(n) = n$ best

$C(n) = n$ worst

$C(n) = n$ average

$O(n)$

$\Omega(n) = \Theta(n)$

$\Theta(n)$

rastgele elemanların ortalaması

② Sequential - Linear search

```
int linearSearch(int n, int *A, int v) {
    int i=0;
    while(i<n && A[i]!=v)
        i++;
    if(i<n)
        return i;
    return -1; }
```

basic operation

$C(n)_{best} = 1$

$C(n)_{worst} = n$

$C_{avg} \rightarrow p$: $0 \le p \le 1$ elemanın var i^{th} pozisyon $\frac{p}{n}$

$$C_{avg}(n) = \left[1*\frac{p}{n} + 2*\frac{p}{n} + \dots + n*\frac{p}{n}\right] + n*(1-p)$$

$\rightarrow p\frac{(n+1)\cancel{n}}{\cancel{2}} \rightarrow \frac{pn+p}{2}$ → varsa

yoksa → $p = 1$ ise yani elemanın varsa

$\frac{1.n+1}{2} + n*(1-1) \rightarrow \Theta(n)$

$p=0$ için $\frac{0.(n+1)}{2} + n*(1-0) \rightarrow n$

$\rightarrow C_{avg}(n) = A \rightarrow O(n) \rightarrow$ (C_{best} / C_{worst} olduğu için $\Omega(1)$ → Θ yok)

insertion Sort

9	3	7	1	4	5
3	9	7	1	4	5
3	7	9	1	4	5
1	3	7	9	4	5

⋮

1 3 4 5 7 9

C(n) → sıralı dizi → n
best

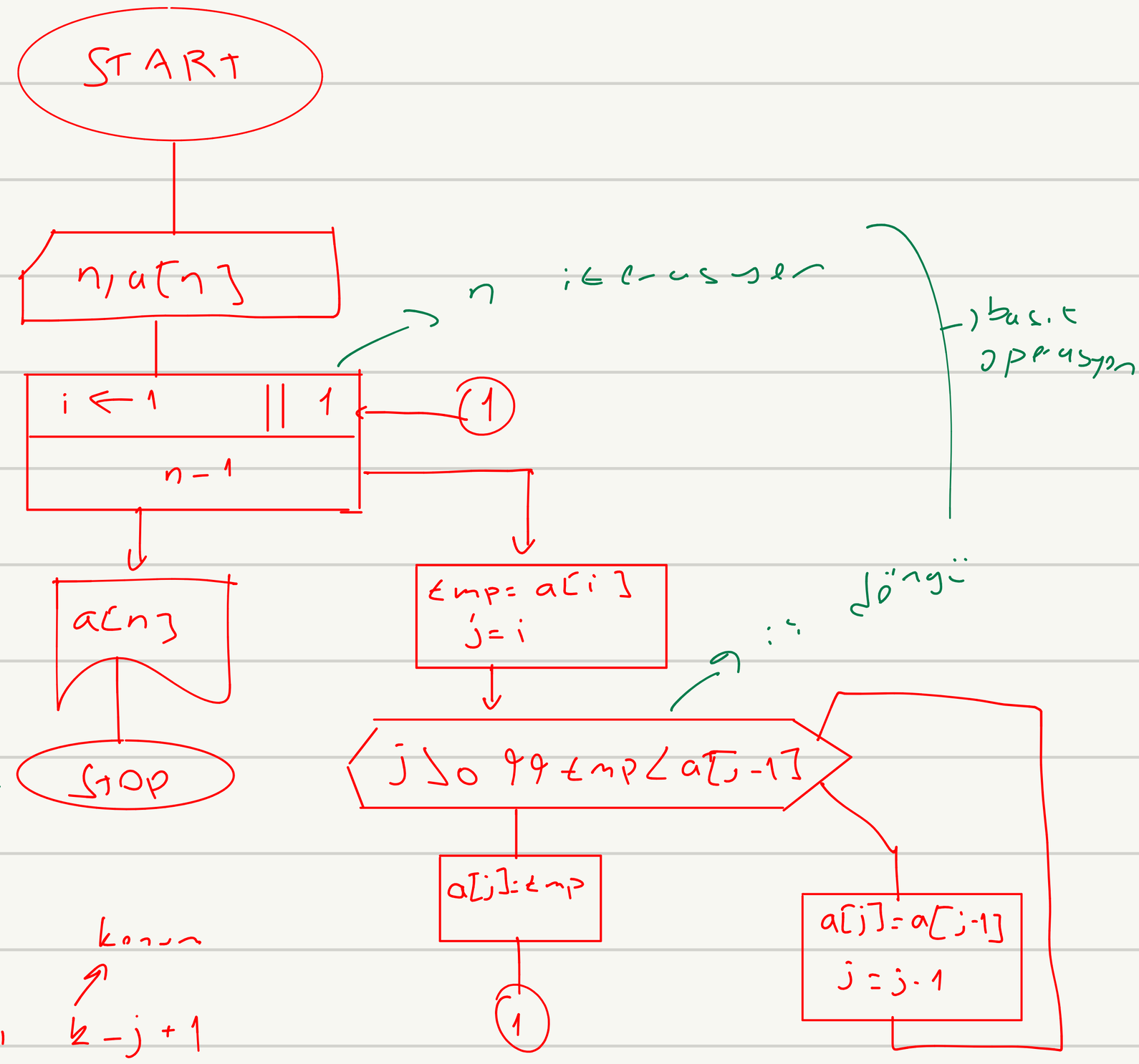
i=1 → 1
i=2 → 1
i=3 → 1
⋮
i=n-1 → 1

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

Cworst(n) → tersten sıralı → n²
karşılaştırma sayısı

i=1 → 1
i=2 → 2
⋮
i=n-1 → n-1

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



Caverage → karşılaştırma sayısı → k - j + 1
k → konum

$\frac{1}{k} \sum_{j=1}^{k} (k-j+1) \rightarrow \frac{1}{k}\left(k^2 - \frac{k(k+1)}{2} + k\right) \rightarrow k - \frac{(k+1)}{2} + 1 \rightarrow \frac{k+1}{2}$ → inner loop

d.s. döngü $\sum_{k=1}^{n-1} \frac{k+1}{2} \rightarrow \frac{n.(n-1)}{2} + \frac{n-1}{2} \approx n^2$

$C(n)_{avg} \rightarrow n^2$ $\quad \Omega(n)$ $\quad O(n^2)$

Rekürsif için

(1) Factorial

$n! = 1 * 2 * 3 \dots n$
$n! = n.(n-1)!$
$0! = 1$

$f(n) = \begin{cases} 1 & n = 0 \\ n * f(n-1) & n > 0 \end{cases}$

```
int factorial( int n) {
    if (n==0)
        return 1;
    else
        return n * factorial(n-1);
}
```

basic operation

Rekürsif Bağıntısı

çarpma karmaşıklığı M(n)

$\Omega(n) = \Theta(n) = O(n) = n$

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

$M(0) = 0 \rightarrow \sum_{i=1}^{n} 1 \rightarrow n$

(1) Basit operasyon bul

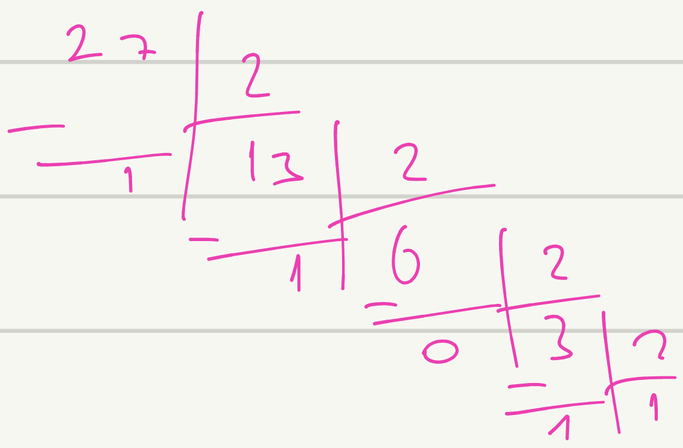
(2) Rekürsif bağıntısı yaz-çöz

2 yol var 1.'si: Backward Substitution → Geriye doğru çözme

$M(n) = M(n-1) + 1$
$M(n-1) = M(n-2) + 1$
$M(n-2) = M(n-3) + 1$
⋮
$M(0) = 0$

$M(n) = M(n-3) + 3 \rightarrow M(n) = M(n-i) + i$
$\rightarrow M(n) = M(n-n) + n$
$M(n) = n$

Find the number of binary digits positive



→ rek(n) = rek(n/2) + 1

```
int binaryDigits(int n){
    if(n==1)
        return 1;
    else
        return 1+binaryDigits(n/2);}
```

<> basic op

$A(n) = A(n/2) + 1 \quad n > 1 \rightarrow A(n/2) = A(n/4) + 1$

$\rightarrow A(n) = A(n/8) + 3$

$A(1) = 0$

$\rightarrow A(n) = A(\frac{n}{2^i}) + i \quad 2^i = n \quad (i = \log_2 n) \quad A(n) = \log_2 n \quad \in \Theta(\log_2 n)$

$A(n/4) = A(n/8) + 2 \quad A(n) = A(\frac{n}{2^i}) + i$

---

forward substitutions

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1)+1 & \text{otherwise} \end{cases}$$

n	T(n)
1	1
2	T(1) + 1 → 2
3	3
4	4
⋮	⋮
n	n

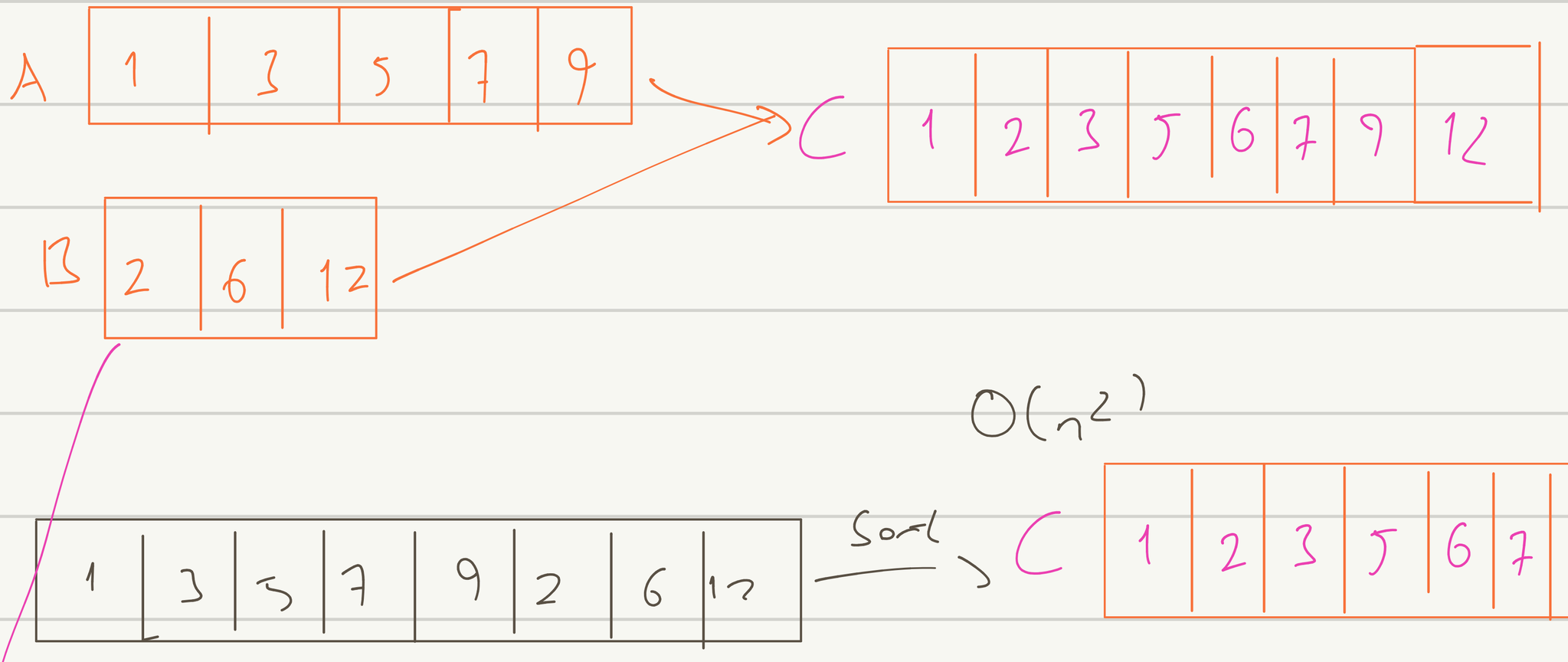
4. Hafta

Insertion Sort

10^8 karşılaştırmalı saniye yapabilen bilgisayarda

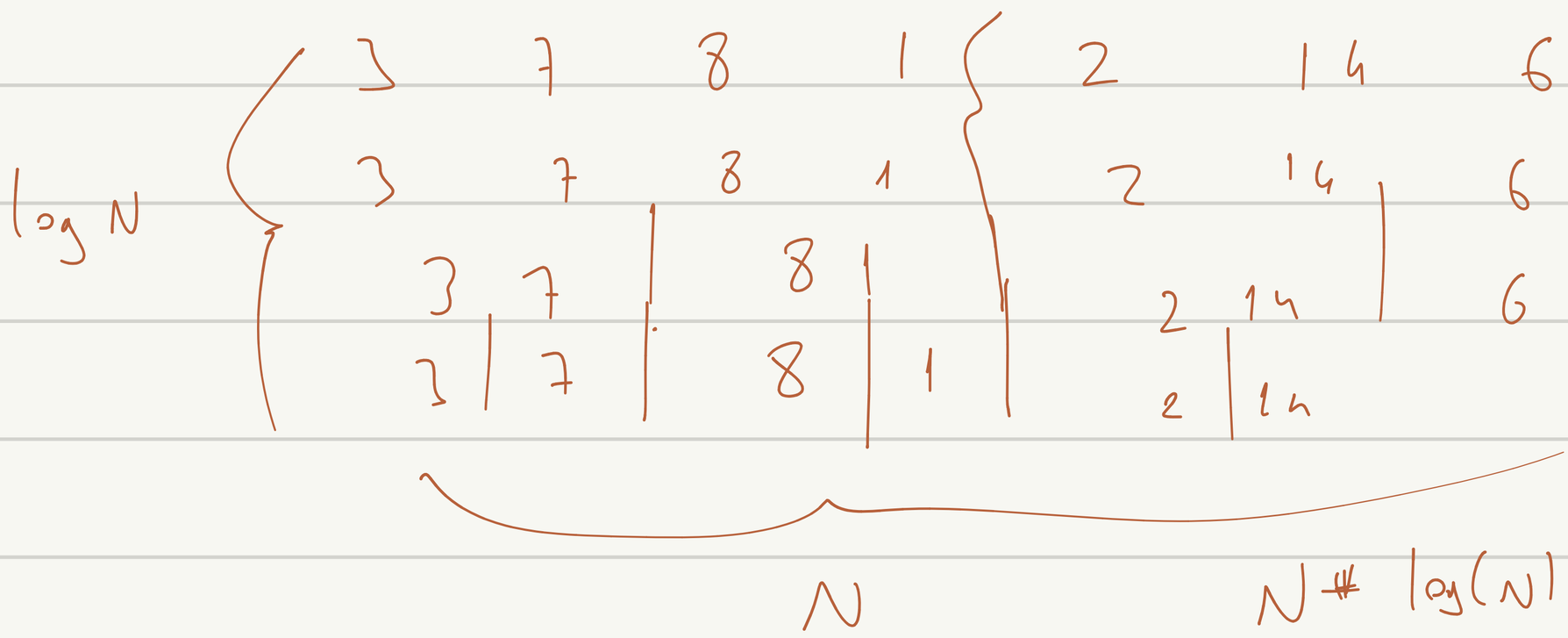
n	1000	1 milyon	1 milyar
$O(n^2)$	anlık	2.8 saat	317 yıl
merge sort $O(n \log n)$	anlık	1 saniye	18 dakika
Quick sort $O(n \log n)$	anlık	0.6 saniye	12 dakika

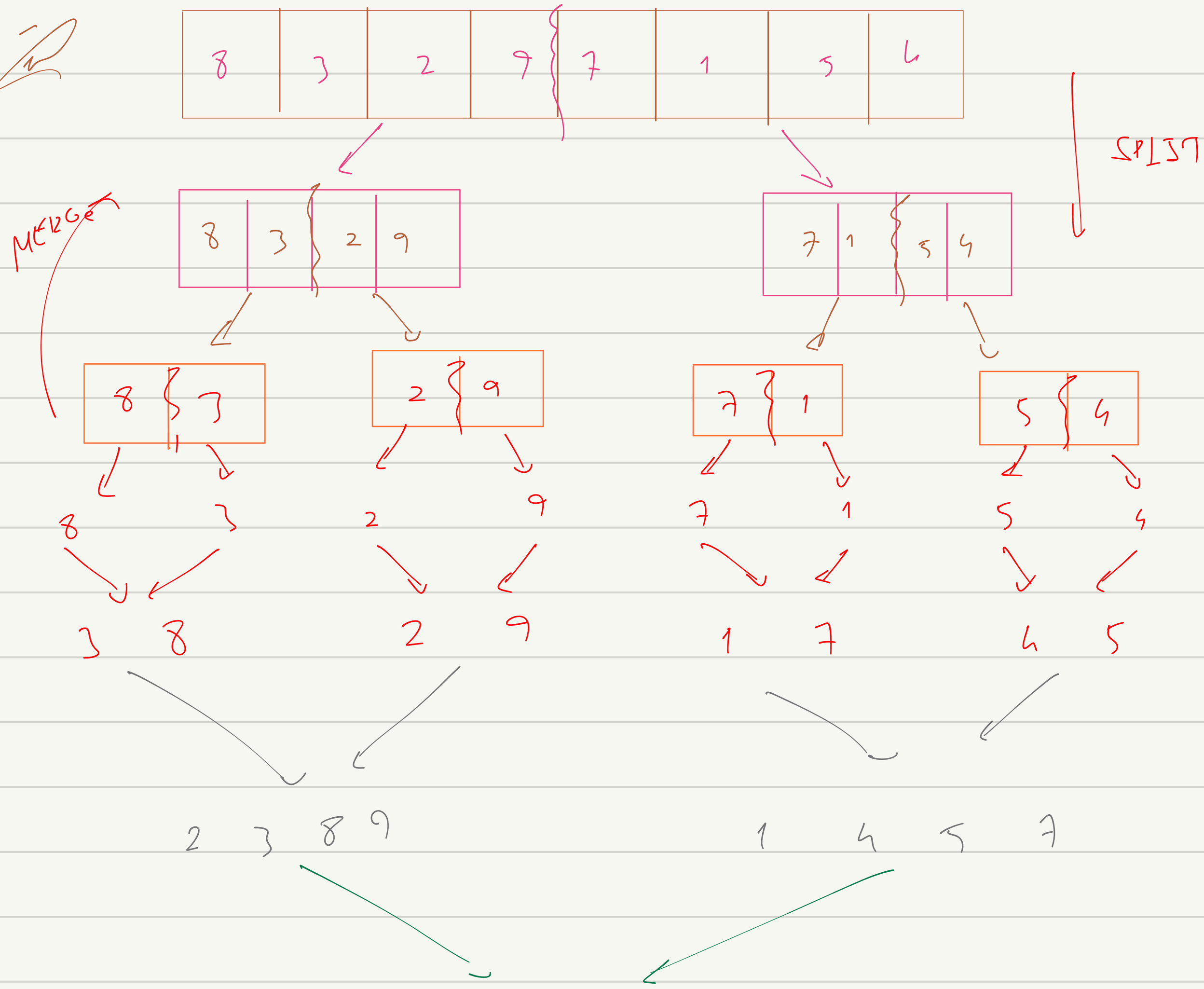
Divide and conquer



→ B'deki kontrolle üst üste eklemek merge ve O(n²)'de olur.

Merge sort bu mantıkla en düşük parçalara bölüp merge eder.





DIVIDE and Conquer

1: DIVIDE → Aynı algoritmayı uygulayabileceğiniz küçük parçalara bil

2: Bildiğin parçaları işle

3: " " birleştir

Merge Sort: 1-) SPLIT the array 2.) Sort each subsequence

1-) merge the two sorted sub sequences

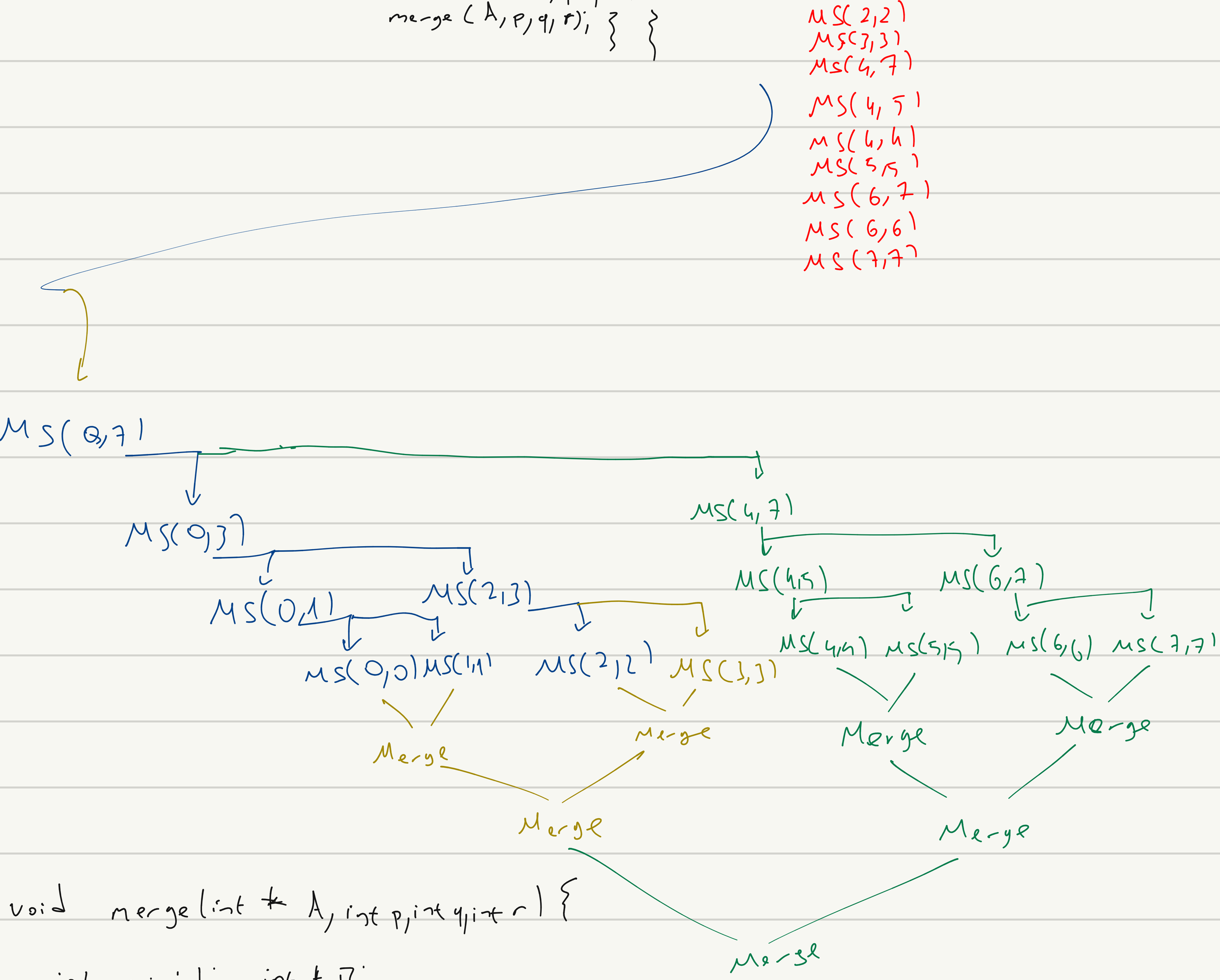
$A[1 \dots n-1] \rightarrow A[1 - n/2]$ ... $A[n/2 - n-1]$

```
void mergeSort(int *A, int p, int r) {
    if (p<r) {
        int q=(p+r)/2;
        mergeSort(A,p,q);
        mergeSort(A,q+1,p);
        merge(A,p,q,r); } }
```

başlangıç → p, son → r

p=0, r=7, q=3

MS(0,3)
MS(0,1)
MS(0,0)
MS(1,1)
MS(2,3)
MS(2,2)
MS(3,3)
MS(4,7)
MS(4,5)
MS(4,4)
MS(5,5)
MS(6,7)
MS(6,6)
MS(7,7)




```
void merge(int *A, int p, int q, int r) {
    int n,i,j,k; int *B;
    n= r-p+1; // eleman sayısı
    B= int*)malloc(n*sizeof(int));
    i = p;  j=q+1; k=0;
    while( i<=q and j<=r) {
        if(A[i] < A[j])
            B[k++] = A[i++];
        else
            B[k++]=A[j++]; }
```

```
while(i<=q) B[k++]=A[i++];
while(j<=r) B[k++]=A[j++];
for(i=p, k=0; i<=r; i++,k++)
    A[i] = B[k];
free(B); }
```

Recurrence Relation of Merge Sort

$$T(n) = \begin{cases} O(1) & n = 0,1 \\ 2 * T(n/2) + n & \text{if } n > 1 \end{cases}$$

$\rightarrow O(\log_2 n * n) \rightarrow \Theta(\log_2 n * n)$

Backward Substitution

$T(n) = 2 * T(n/2) + n$

$T(n/2) = 2 * T(n/4) + n/2$

$T(n/4) = 2 * T(n/8) + n/4$

$T(n/2) = 4 * T(n/8) + n$

$T(n) = 8 * T(n/8) + 3n$

$= 2^i * T(\frac{n}{2^i}) + i * n$

$n = 2^i \quad = n * T(1) + \log(n) * n$

$= n + n * \log(n)$

Ezber bilgi

Genel böl ve yönetlerin recurrence bağıntıları

$T(n) = a * T(\frac{n}{b}) + f(n^{\alpha})$ ve $T(1) = c$

(a: kaç defa çağırdın; b: kaça böldük; n^{α}: iş yükü)

If $f(n) \in \Theta(n^{\alpha})$ when $\alpha \geq 0$ then $T(n)$

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

not: Ω ve O'da da olur.

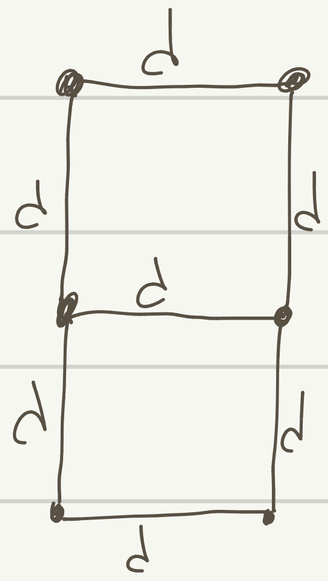
Ör: $T(n) = 1 \cdot T(\frac{2}{3} n) + 1$

$\alpha = 0 \quad a = 1 \quad b = \frac{3}{2}$

$1 = (\frac{3}{2})^0 \rightarrow \Theta(\log_2 n \cdot n^{\alpha}) = \Theta(\log_2 n)$

LAB 1

n elemanlı dizi var ve en yakın 2 elemanı bul deniyor

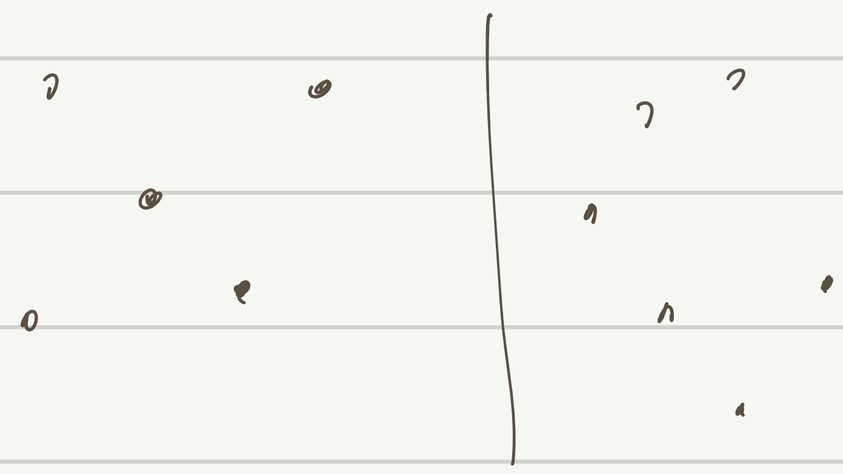


→ noktalar y'lerine göre sıralı bundan dolayı 2d+d'lik bir alanda en yakın 6 elemanı bulmuz. Onunla kıyaslarız.

$T(n) = 2T(n/2) + 6N$

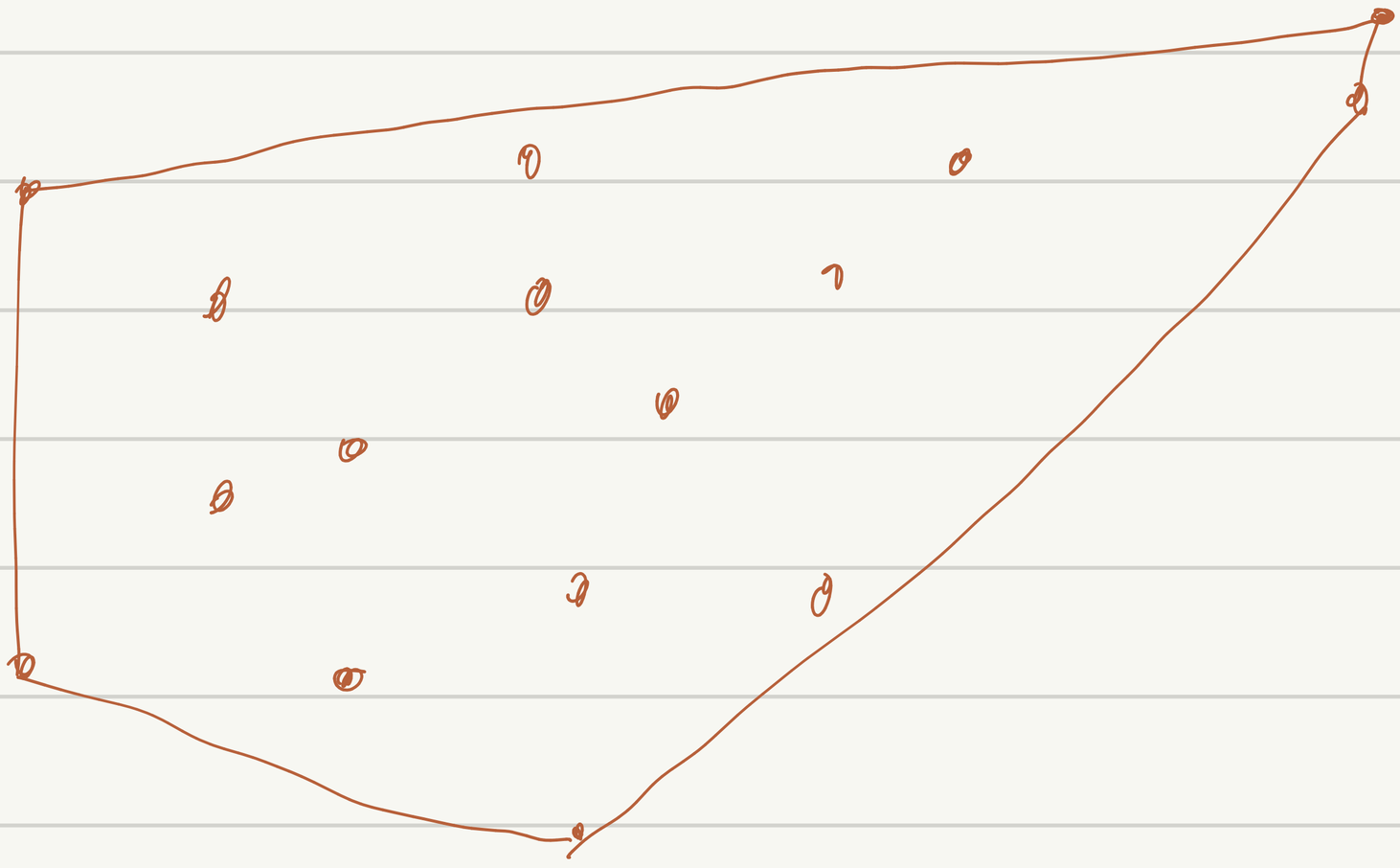
$a=2 \quad b=2 \quad d=1 \quad a=b^d \quad n \cdot \log n$

$a < 3$



kapsum

noktalar kümesindeki en küçük dış bükey

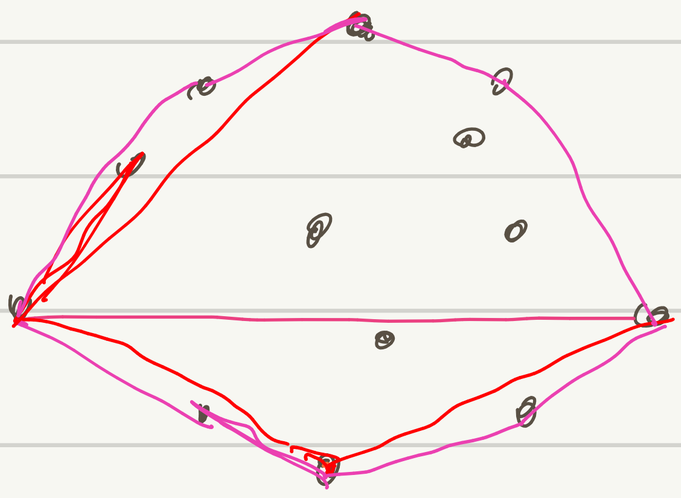


brute force

$\binom{n}{3} \binom{n}{4} \binom{n}{5} \binom{n}{6} \cdots \binom{n}{n}$

$= 2^n$

X eksenine göre en uzak 2 noktadan bölüz, sonra n dışı deneyerek Σ noktaya en uzak noktayı seçiyoruz. Eğer dışta kalan varsa başka nokta seçiyoruz.

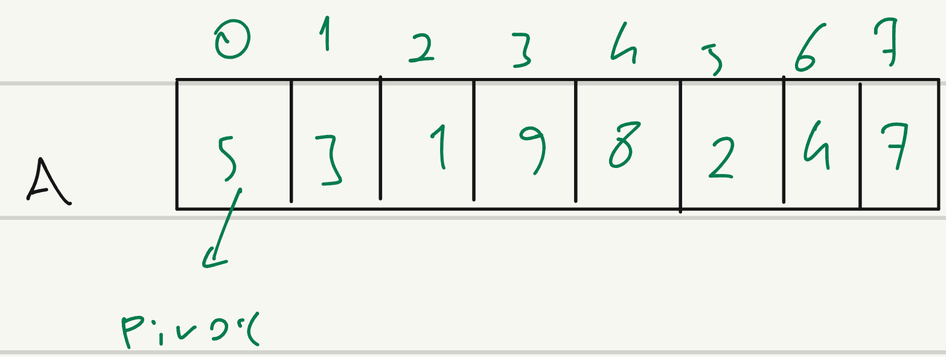


$T(N) = 2T(N/2) + N$

$= n \log n \rightarrow$ ortalama

Hafta 5

Quick Sort → In Place

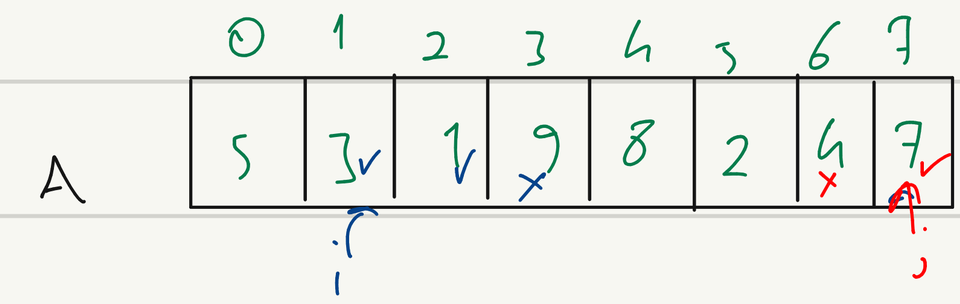


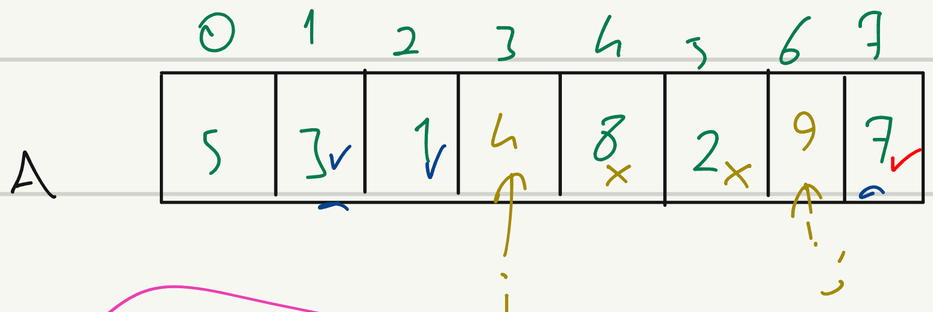
A[n]

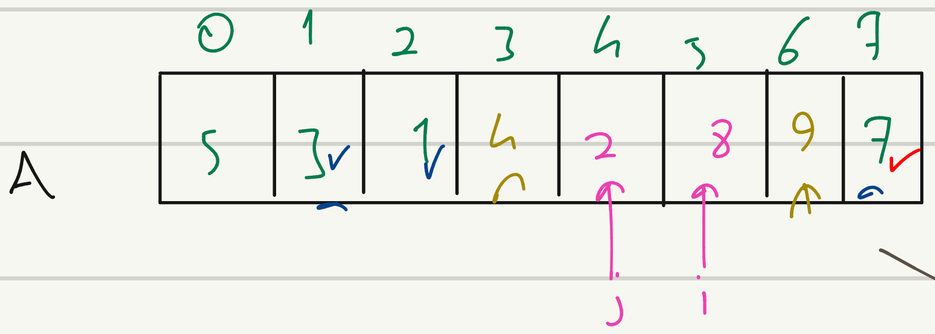
A[0...s-1] pivot A[s+1...n-1]

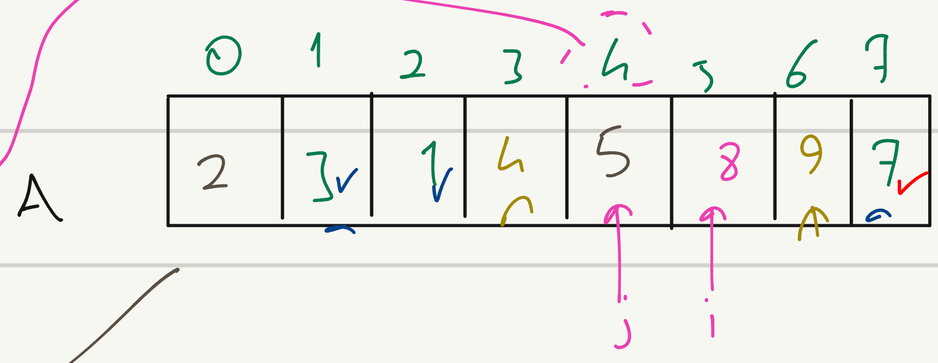
A ≤ pivot pivot ≤ A

3 2 1 4 7 11 13 8 12









L=0 R=7

S=4 pivot=4

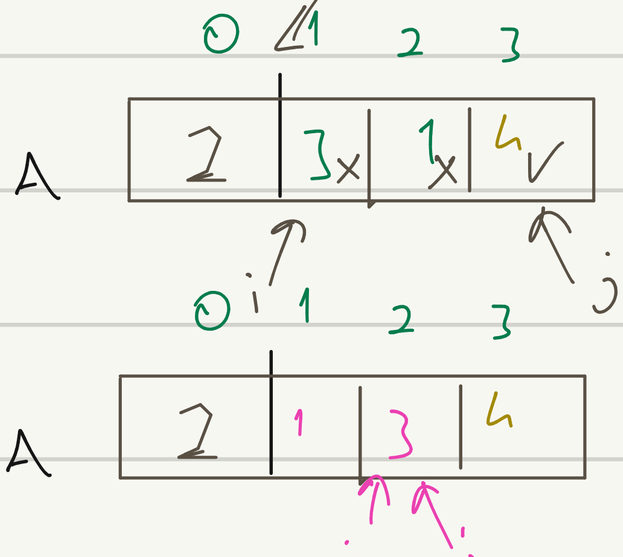
L=0 R=3

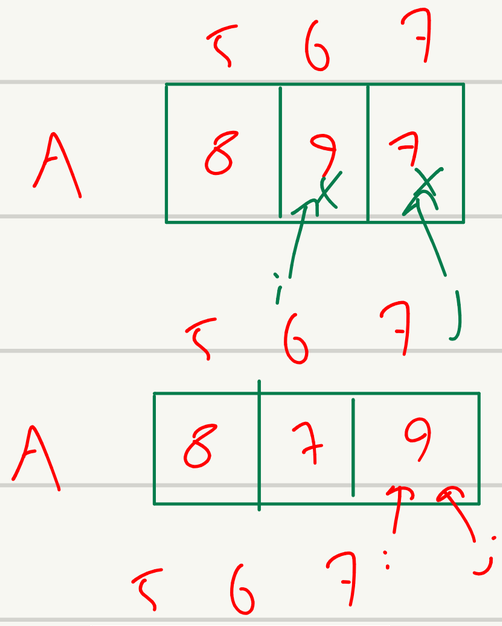
S=1 pivot=2

S=1

L=5 R=7

S=6 pivot=8





L=0 R=0

X

L=2 R=3

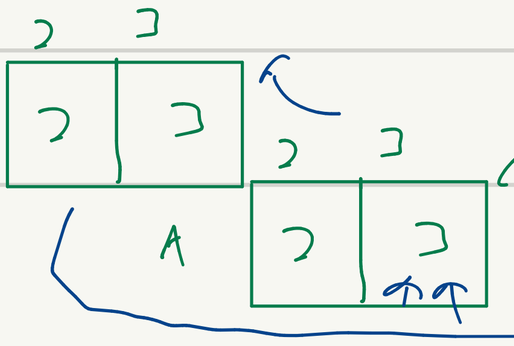
S=2 Pivot=2

L=2, R=1

X

L=3, R=3

Y



L=2 R=1

Y

L=3 R=3

Y

PARTITIONING

```
quicksort(n,A[l....r]){
      if(l<r){
            s=partition(A[l...r]); //Pivot yerine yerleşir küçükler
solunda büyükler sağında kalır ÖDEV
            quicksort(A[l...s-1]);
            quicksort(A[s+1.....r]);}}
```

Best Case $\rightarrow T(n) = 2T(n/2) + n^{\lceil \rceil \rightarrow 0}$ $T(1) = 0$

$a=2 \quad b=2 \quad d=1$

$2 = 2^1$ olduğu için $n^1 \log n \rightarrow n \log n$

Pivot ortadaşken best case hep aynı olduğundan

$\Theta(n \log n)$

Worst Case

---

birişiken büyük-büyükten küçüğe sıralı olunca

$T(n) = T(n-1) + n$ for $(n \geq 1)$ $\Theta(n^2)$

$T(0) = T(1) = 1$

Average

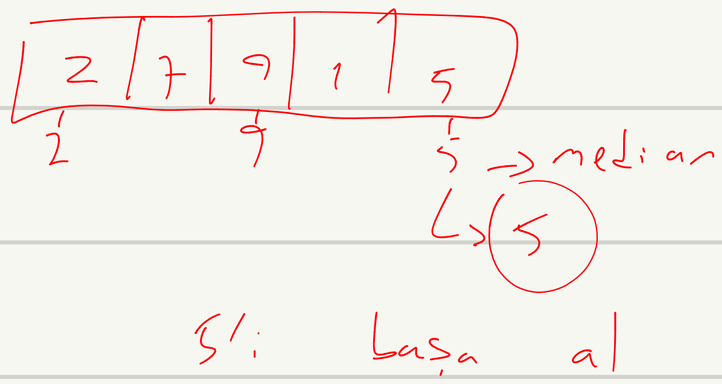
Position of pivot	left of pivot	r. of p.	$T(n_{left})$	$T(n_{right})$
0	0	$n-1$	$T(0)$	$T(n-1)$
1	1	$n-2$	$T(1)$	$T(n-2)$
2	2	$n-3$	$T(2)$	$T(n-3)$
⋮				
$n-1$	$n-1$	0	$T(n-1)$	$T(0)$

$T(n) = \frac{1}{N} \cdot \sum_{k=1}^{N} (T(k-1) + T(n-k) + n) \rightarrow T(n) = 1,39\, N \log_2(N) \in O(n \log_2 n)$

↳ QS merge s.'dan %39 fazla kıyaslama yapıyor ama eleman değiştirmesi uz ondan dolayı daha hızlı

worst case'den kurtulma yolları

(1) karıştır (2) median of samples → (3) random pick pivot



Decrase And Conquer

(1) Decrease by a constant fuctor

binary search

problem of size n

↓

" " " n/2

⋮

solution to the original problem

$T(n) = T(n/2) + c$

(2) Variable size decrease

$$GCD(a,b) = \begin{cases} a & \text{if } b = 0 \\ GCD(b, a \bmod b) \end{cases}$$

GCD(80, 44)

↳ GCD(44, 36)

↳ GCD(36, 8)

↳ GCD(8, 4)

↳ GCD(4, 0)

↳ 4

(3) Decrease by a constant topological sort

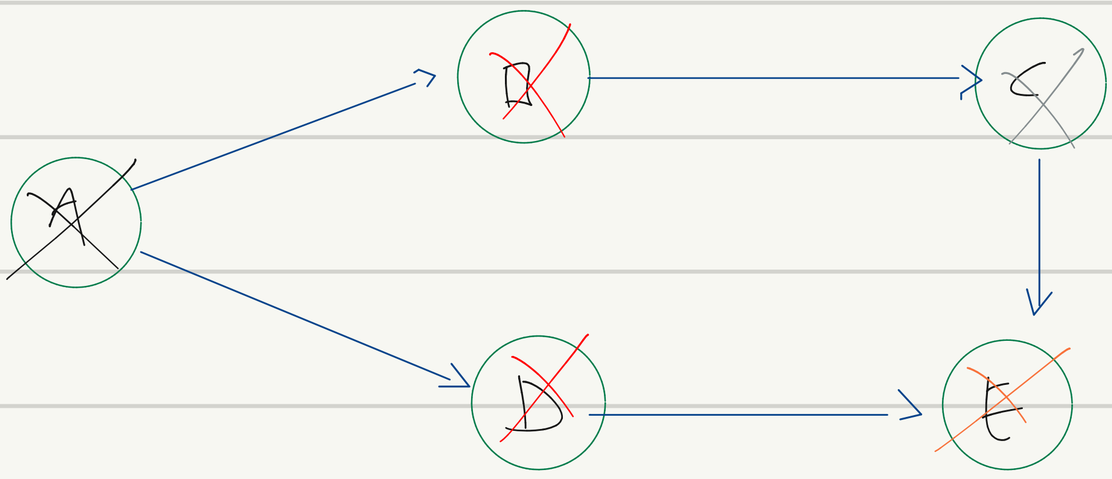
6. Hafta

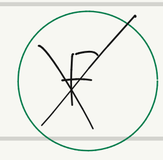
Decrease and conquer

Decrease by a constant

Topological sort → Grafla çözülür

Dairesel olmamalı ünlü olarak ön şartlar verildi





İlk final

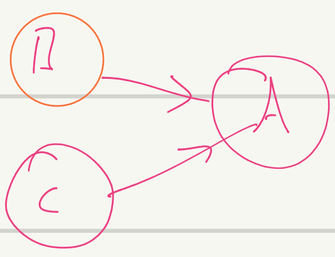
A F

B D

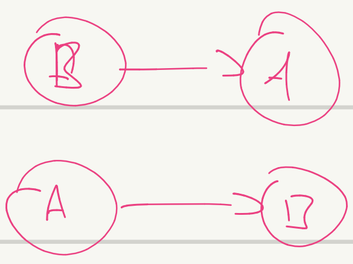
C

E

In_degree



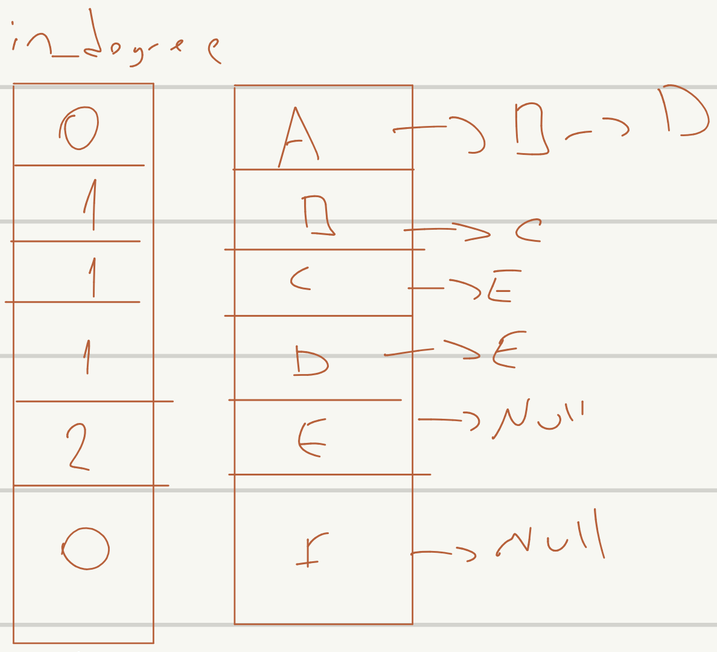
A'nın
In_degrees: 2



A'nın
In_degrees: 1

A'nın 0

Komşuluk Listesi

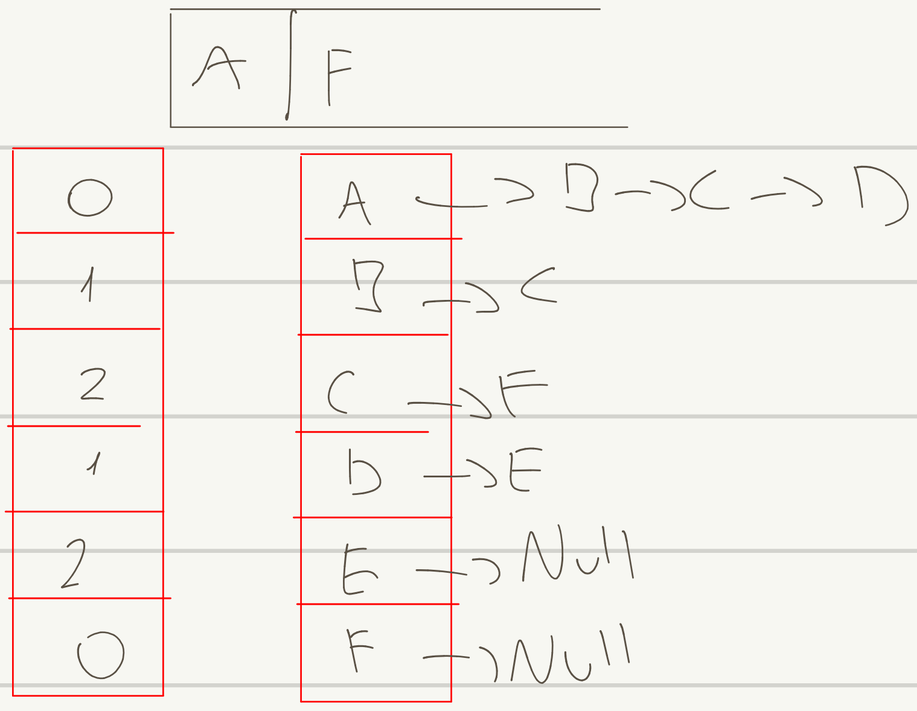


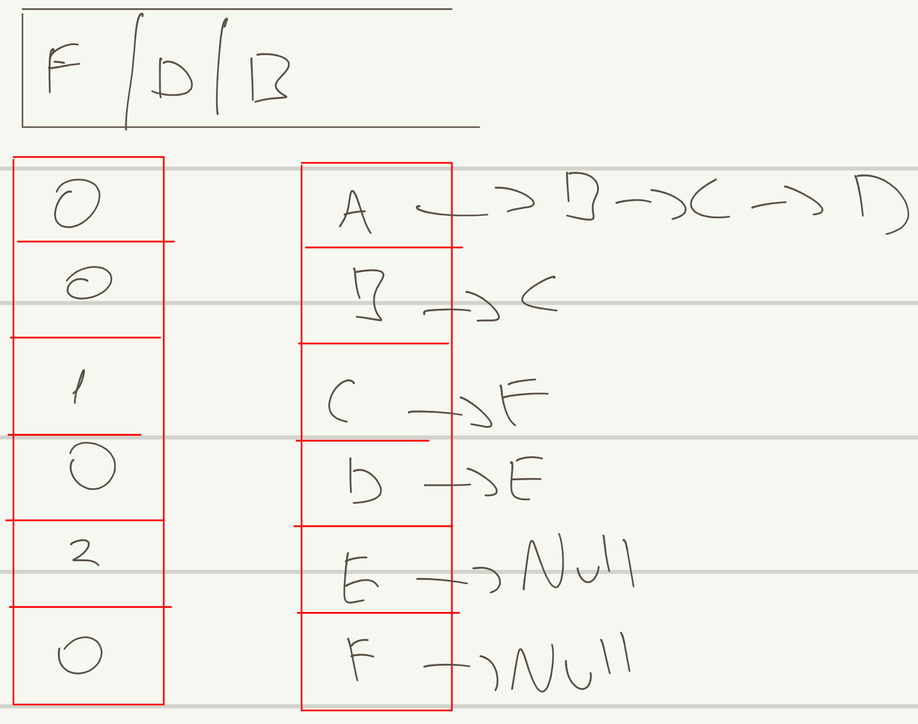
→ indegree Sayısı bulma

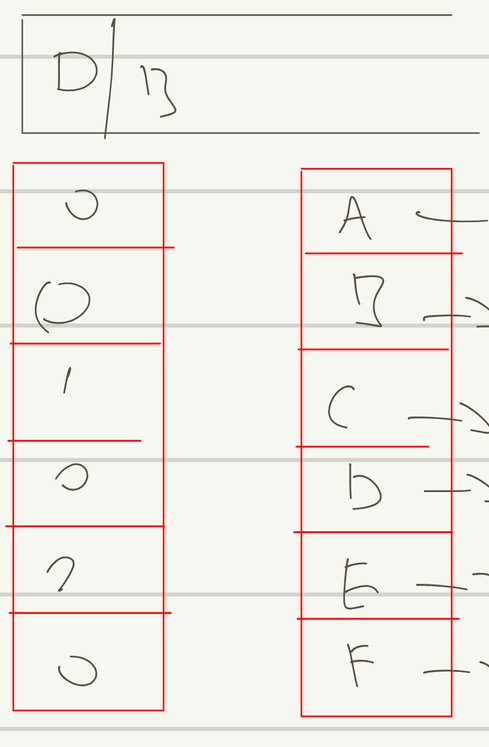
Listeyi geziyoruz, nextlediğimiz nesne eşitse onu artırıyoruz

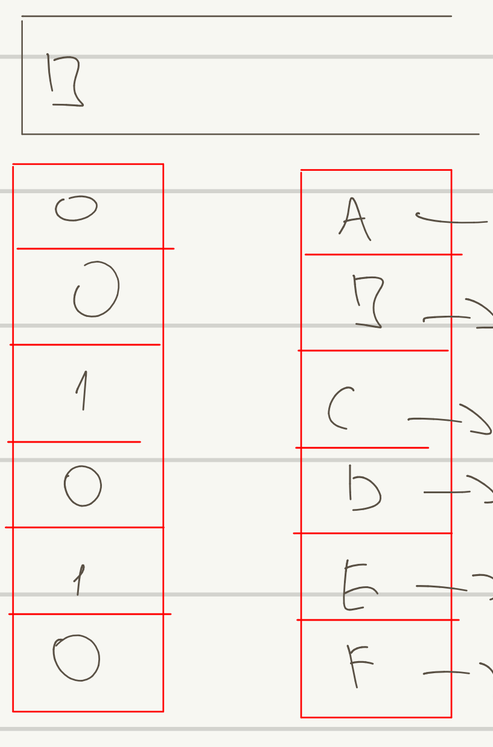
↳ Bu bir işin yapılabilmesi için başka işin bitmesi gerektiğini söyleyemez ~~E~~ E için önce A, sonra B, Sonra C'nin bitmesi lazım. Ama indegree[E]=2 ~~A~~ A bitince B ve C'yi azaltmak lazım. Indegree[]--

Hangi işten başlayacağımızı bulmak için queue'ye atıyız.

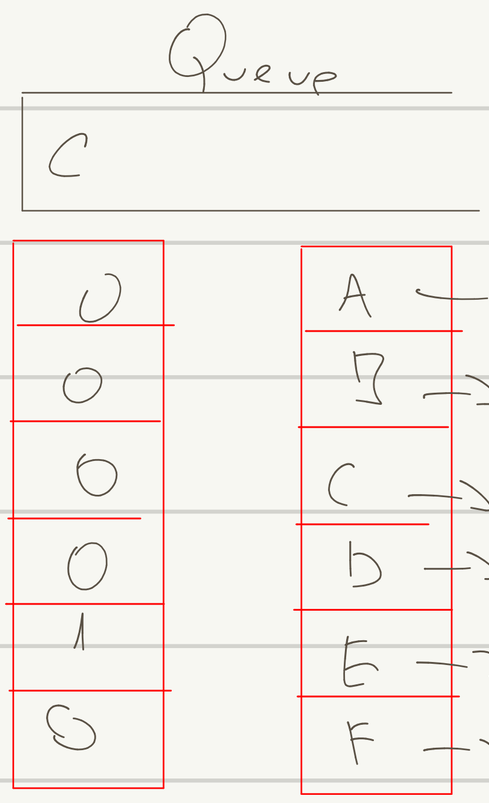


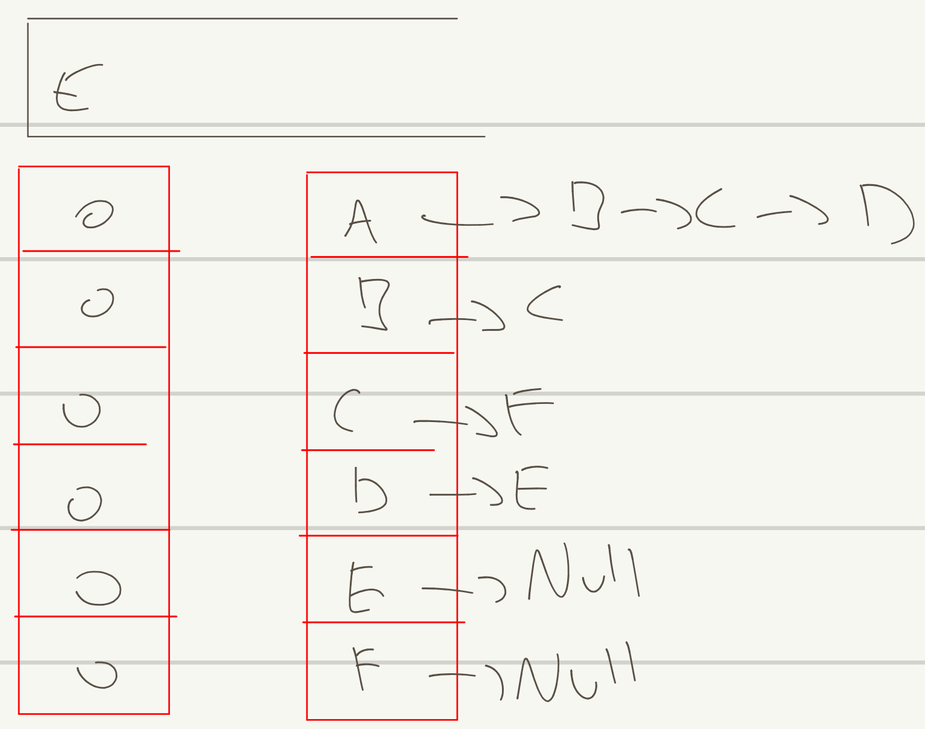






Queue





Queue -> $O(V^2+E)$

1. Store each vertex's in_degree in array $O(|E|)$
2. While there are vertices remainly:

   find a vertex with in degree zero $O(|V|^2)$

   reduce in degree of all vertices adjecent to it by 1 $O(E)$

   mark in vertex's in-degree -1

Queue -> $O(V+E)$

1. Store each vertex's in_degree in array $O(|E|)$
2. Initialized a queue with all in_degree zero vertices $O(V)$
3. While there are vertices remaining in the queue
   - Dequeue and output $O(V)$
   - Reduce in_degree of all vertices adjcent to it by 1 $O(E)$
   - Enqeue vertices with indegre zero $O(E)$

	Worst search	Worst insert	Worst delete	Average search	Average insert	Average delete
Sırasız dizi	n	1	n	n/2	1	n/2
Sıralı dizi	$\log_2 n$	n	n	$\log_2 n$	n/2	n/2
Dengesiz Tree	n	n	n	$\log_2 n$	$\log_2 n$	$\log_2 n$
Dengeli Tree	$\log_2 n$	$\log_2 n$	$\log_2 n$	$\log_2 n$	$\log_2 n$	$\log_2 n$

Hashing

	hash table
0	Null
1	Null
2	Null
3	Null
4	18 247
5	Null
6	Null
7	Null
8	Null

18247 → hash function → 4

insert → O(1) search O(1) ← 4 [hash funct.] ← 18247

① Hash table'ın uzunluğu ne olmalı? Aynı indise birden çok sayı atamamak için.

② Hash fonksiyonunu nasıl seçmeliyim?

③ Çakışma olunca ne olacak

Hash functions

① Division method

hash function (key) = key mod m

table size → 0 → m-1

asal sayı olsa daha iyi olur. çünkü 8 olsa 2'nin katları da ⓪ olabilir. Ama 7 olursa 7'nin katları 07

key = 25

hash(25) = 25 % 7 → 4

hash(7) = 7 % 7

0	7
1	
2	
3	
4	25
5	
6	

② Multiplication method

hash function (key) = ⌊ m * (key * A) mod 1 ⌋

(m ↑ size)

knuth ↓

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

m 10.000 key 123.456

hash function (123.456) = ⌊10.000 * [(123.456 * 0.618033) mod 1]⌋

↓

76300.0041151 mod 1

.0041151

10,000 * .0041151

⌊41.151⌋ = (41)

41 → 123.456

HAFTA 7

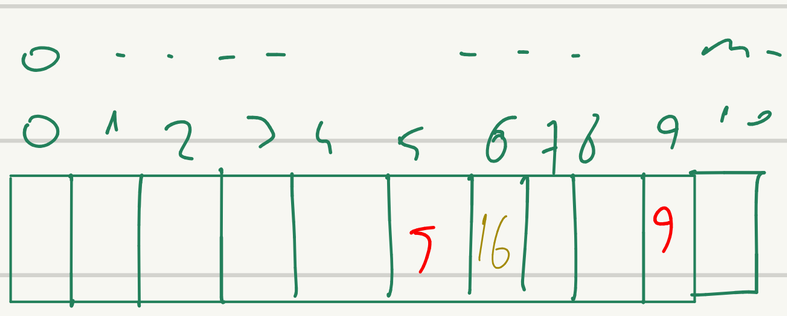
Hashing

1-) Division method $hash(key) = key \% m$ (m → table size)

2-) Mult. " $hash(key) = \lfloor m * ([key * A] \% 1) \rfloor$

table size PRIME olmalı.

Collision problemi (çakışma)



m = 11

① Open Adressing

key = 5
hash(key) = 5 % 11

key = 9
hash(key) = 9 % 11

key = 16
hash(key) = 16 % 11 = 5 ??!
(5 dolu)

1.1) Linear Probing → each check is a probe (→ Silmede flag kullanırız)

hash(key, i) = [hash(key) + i] % m (i. yüz boş)

key = 16 i = 0 → hash(16, 0) = 5

key = 16 i = 1 → hash(16, 1) = (5 + 1) % 11 = 6

insert

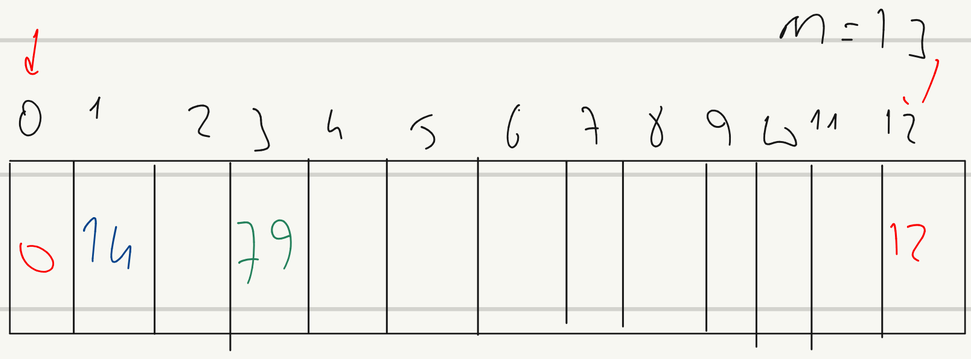
Arama işleminde Null görene kadar ya da başa dönene kadar ararız. Search(key, i)

→ while(hash[adr] != Null && i < m && hash[adr] != key) { }

(i < m kontrol: Başa dönmeyi kontrol: M ≠ N olur.)

Linear Probing'in Clustering problemi var → Mesela 5 5'e geldi 16 da 6'ya geldi. Bu önek için 6 asıl yerine gelemeyecek. Bunden dolayı

Burası hiçbir zaman dolmaz



hash1(key) = key % 13

hash2(key) = key % 11

→ 1.2-) Double Hashing

hash(k, i) = [hash1(k) + i * hash2(k)] % m (hash2(k) → jump size)

k = 14 hash(14) = 1

k = 79 hash(79) = 1 → hash(79, 1) = (1 + 1*2) % 13 → 3

Birthday paradox

2 kişinin doğum gününün farklı olma olasılığı

$\frac{365}{365} \times \frac{364}{365}$

k kişi → $\prod_{i=0}^{k-1} \frac{365-i}{365}$ → Tablo size m

$\frac{m}{m} \cdot \frac{m-1}{m}$ $k=2$ → $\prod_{i=0}^{k-1} \frac{m-i}{m}$

taylor expansion yaklaşık olarak $e^{\frac{-(k-1)*k}{2m}} = p$ load factor

collision problem $= 1 - e^{\frac{-(k-1)*k}{2m}}$

çakışma olasılığı

$\frac{N}{M}$ $\frac{10}{100}$ → %10

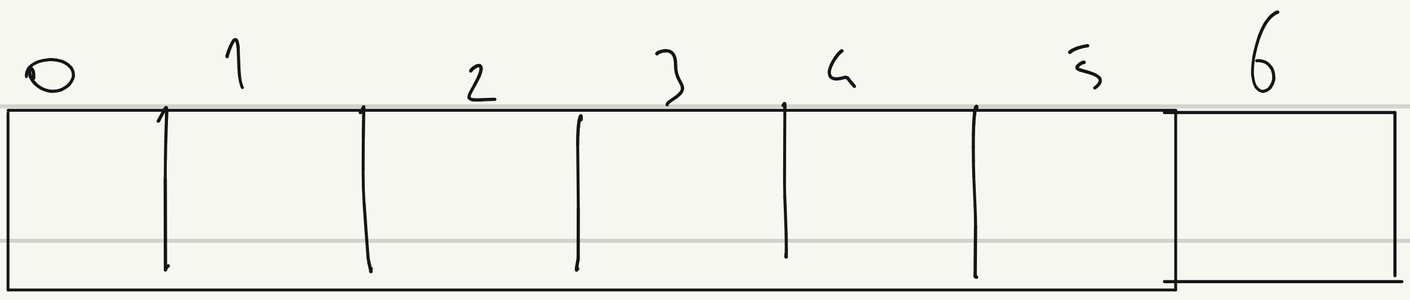
Open addressingde load factor → yaklaşık %50 → Aslı sayı $M \cong 2N$

---

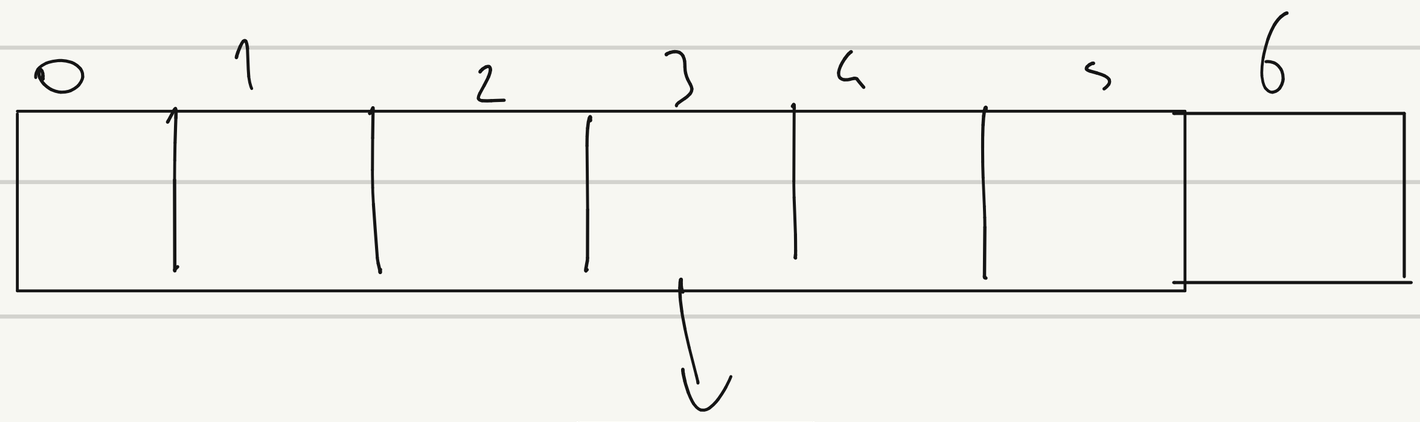
(2) Seperate Chaining

$M \cong N/5$

load factor $\cong$ %70

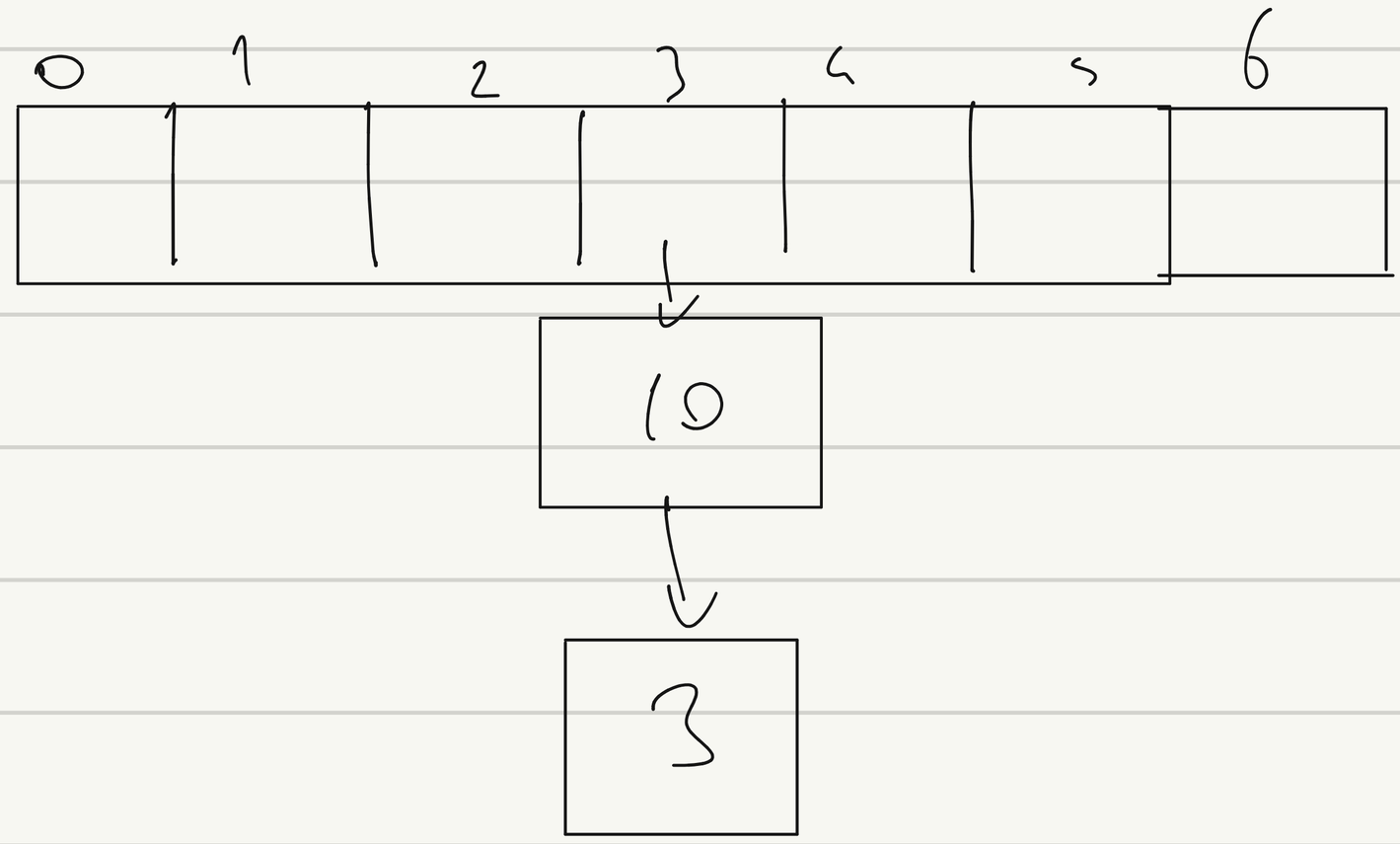


key=3 10%7=3



yeni elemanı ekleyiniz

key=10 10%7=3




```
struct node{
    int key;
    char name[50];
    struct node *next;
};
```

key unique olmalı

(1) key Stringse

"Kayra"   hash("kayra")

70 10 60 50 10 toplam 200 olur ama aryaktın da değeri 200 olur.

tercih edilmez

bunu

Lenght = L → $k^{L-1} * str[0] + k^{L-2} * str[1] \dots k^{0} * str[L-1]$ ile çözeriz.

k asal ve küçük olmalı..   Horner's method <u>k=31</u>

```
hashkey=0
length = strlen(s);
for(i=0;i<length;i++){
    tmp=s[i]-'A'+1;
    hashkey=k * hashkey+tmp;}
```

Universal hashing → saldıranlar fonksiyonunu bilerek collusion oluşturmaya çalışıyor.

Birden fazla hash fonksiyonu kullanılıyor ve rastgele seçiliyor, karşı taraf bunu anlayamadığı için saldırması zor

hash(key) = [(a * k + b) mod p] mod m

a, b ve p değişken

Hash tablosuna eleman eklenecek, eğer varsa eklemeyen - yoksa ekleyen algoritmayı tasarla

↓

```
i=0
adr=key%m
while(hash[adr]!=NULL && i<m && hash[adr]!=key){
        i++;
        adr=(adr+1)%m; }

if( i==m)
    printf("tablo dolu")
else if(hash[adr]==NULL)
        hash[adr]=key;
else
    printf("eleman zaten var");
```

LAB 2

(1) $\sum_{i=1}^{n}\sum_{j=1}^{n} ij \rightarrow \sum_{i=1}^{n} i\sum_{j=1}^{n} j \rightarrow \sum_{i=1}^{n} i\left(\frac{n*(n+1)}{2}\right) \rightarrow \left(\frac{n*(n+1)}{2}\right)^2$

(2) $\sum_{i=1}^{n} i^3 \rightarrow$

$T(N) = T(N-1) + 3$

$T(N-1) = T(N-2) + 3$

$T(N-2) = T(N-3) + 3$

$T(1) = 1$

$T(N) = T(n-i) + 3*i \quad N \geq 1$

$T(N) = 1 \quad i = 1$

$i = n-1$

$T(N) = T(1) + 3*(n-1) \rightarrow 3n-2$

Backward

forward

n	T(n)
1	1
2	4
3	7
4	10
:	:
N	T(n-1)+3

$3n-2$

(3) Q(n)

```
if(n=1)
    return 1;
return Q(n-1) + 2*n - 1
```

n^2 algo.

Backward

$\rightarrow T(n) = T(n-1) + 2$

$T(n) = T(n-2) + 2 + 2$

$T(n) = T(n-i) + 2*i$

$i = n-1$

$T(n) = T(1) + 2*(n-1)$

$= 1 + 2n - 2 = 2n-1$

$T(n) = T(n-1) + 2$

$T(n-1) = T(n-2) + 2$

$\vdots \rightarrow 2n-1$

$T(2) = T(1) + 2 \rightarrow$

$T(1) = 1 \rightarrow 1$

4) Selection sort rekürsif → $T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + n-1$

$T(n-2) = T(n-3) + n-2$ → $T(n) = T(n-i) + n-i+1 + n-i+2 + n-i+3 \dots n$

$T(2) = T(1) + 2$ $\quad i = n-1$

$T(1) = 1$ $\quad T(n) = T(1) + 2 + 3 \dots n \rightarrow \frac{n*(n+1)}{2}$

→ Backward

$T(n-1) = T(n-2) + n-1$

$T(n-2) = T(n-2) + n-2$ $\quad T(n) = 1+2+3 \dots n-1 = \sum_{i=1}^{n} i \rightarrow \frac{n*(n-1)}{2}$

$T(3) = 1+2+3 \dots$

$T(2) = 1+2$

$T(1) = 1$

5) inversion problem

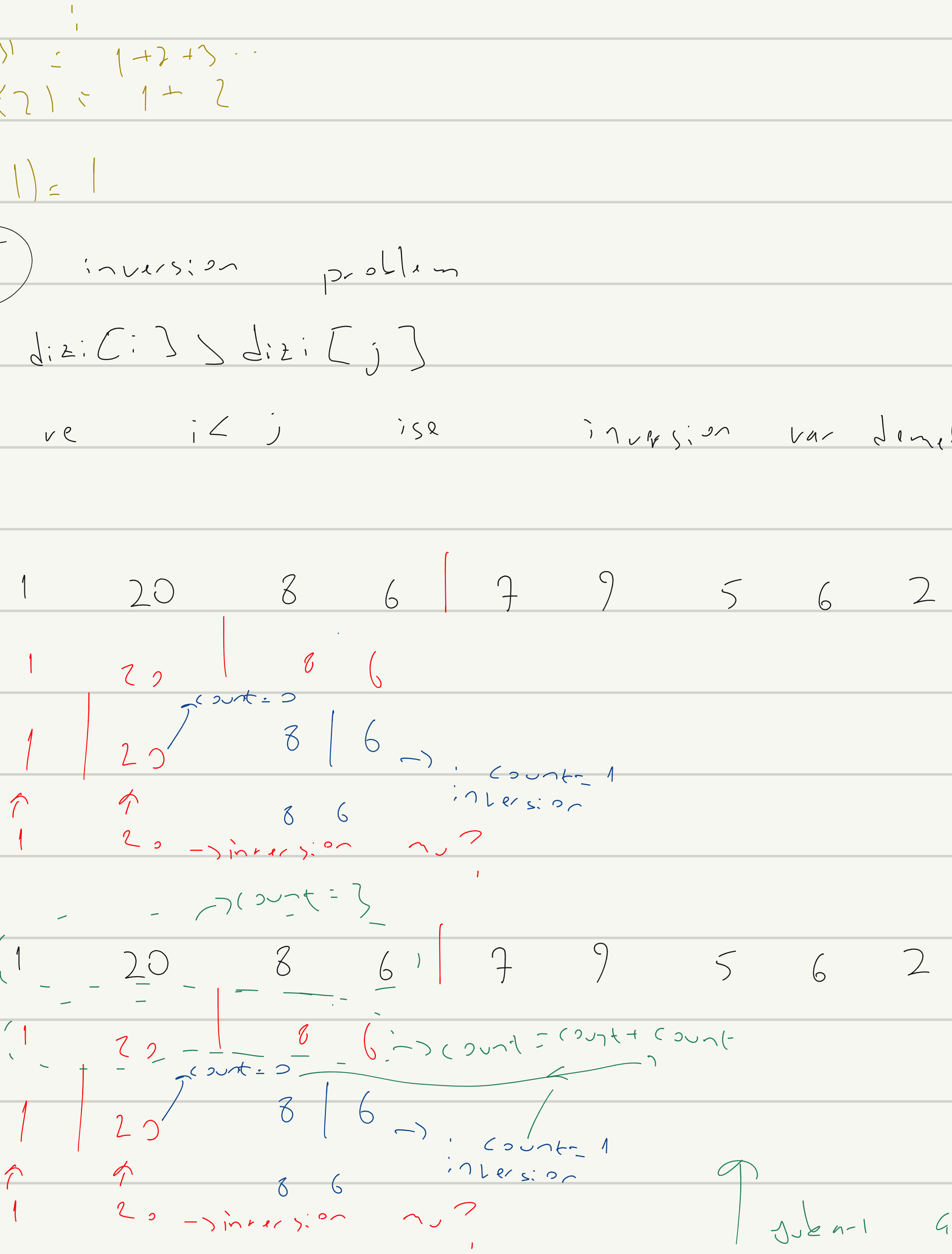
dizi[i] > dizi[j]

ve i < j ise inversion var demektir. Bunu nasıl bulabiliriz?

Burası kırmızı

Diğer sayfa

daha iyi



yukarı çıkarken sıralandığı için mesela 5 7 9 ve

2 6 var. O zaman 5 ile 2'yi kıyaslıyoruz ve sayıcı

direkt 3 artırıyoruz. Çünkü 5 sıkıntıysa 7 ve 9'da sıkıntı. Sonra

5-6'da yok. Sonra 6-7'yi kıyaslayıp 2 artırıyoruz. Çünkü 7 ile

varsa 9 ile de vardır.

1 20 8 6 | 7 9 5 6 2

1. adım

1 20 | 8 6

1 20

1 20 8 6 | 7 9 5 6 2

2. adım

1 20 | 8 6

count=0 | 8 6 -> count=1

not soldaki 6 kıyaslanır.

1 20 8 6 | 7 9 5 6 2

1 20 | 6 8 -> 126 ✓ 2056 count++

count=0 | count=1 2058 count++

1 6 8 20 | 7 9 | 5 6 2

count=3

7 | 9 5 6 2

count=0 | count=1 7 9

1 6 8 20 | 7 9 | 5 6 2

count=3

7 | 9 5 | 6 2

count=0 | count=1 count=0 count=0

1 6 8 20 | 7 9 | 5 6 2

count = 3

count = 0 | count = 1

7 9 | 5 | 6 | 2

count = 0 | count = 0 6 2 | count = 1

1 6 8 20 | 7 9 | 5 6 2

count = 3

count = 0 | count = 1

7 9 | 5 | 6 2

count = 0 | count = 1

1 6 8 20 | 7 9 | 5 6 2

count = 3

count = 0 | count = 1

7 9 | 5 | 6 2 -> count = 1

count = 0 | count = 1

5 - 6 ✓

5 - 2 count++

1 6 8 20 | 7 9 | 2 5 6

count = 3

count 0 | count = 2 -> 7 - 2 count = count + 2

7 - 5 count = count + 2

7 - 6 count = count + 2

1 6 8 20 | 2 5 6 7 9

count = 3

10 + 11 = 22

count = 7 -> 1 - 7 ✓

solution: 6 ✗

6 - 2 count = count + 3

6 - 5 count = count + 3

6 - 6 ✓ count = count + 2

8 - 6 " "

8 - 7

8 - 9 ✓ count++

20 - 9

6 Vida Somun

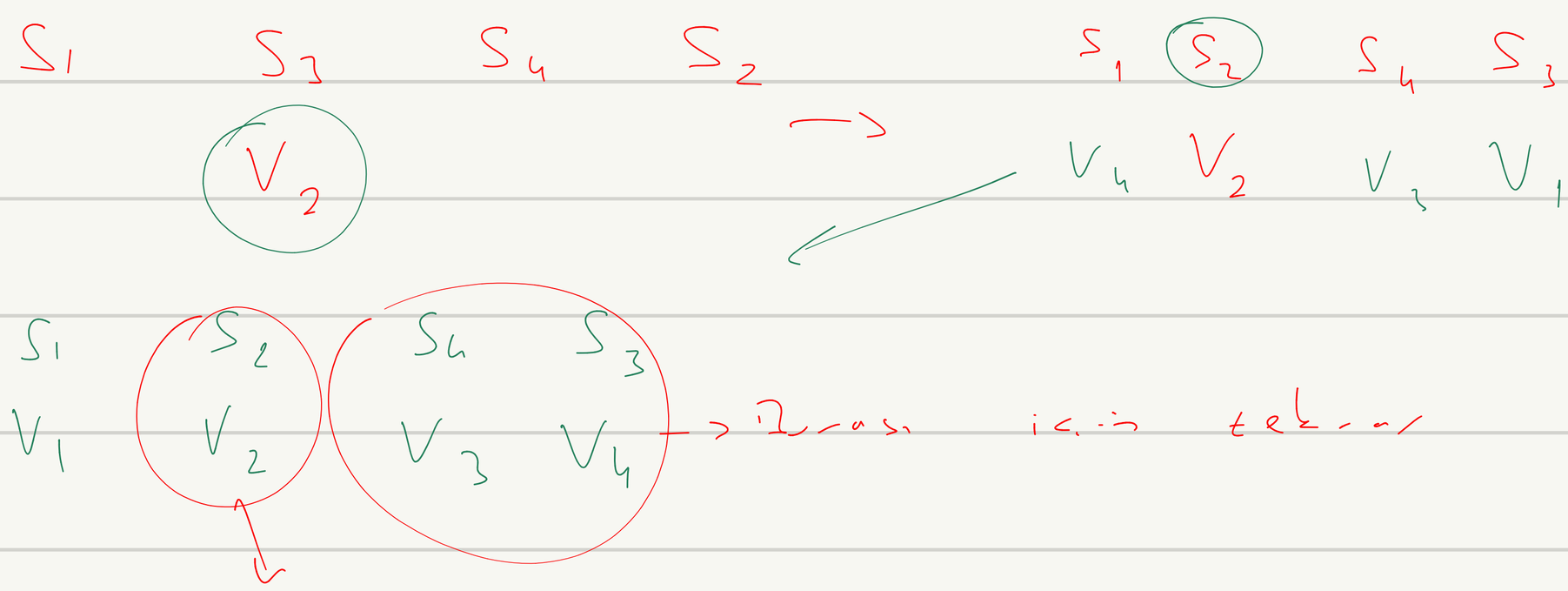
→ hangi vida hangi somuna

vidaları arasından kıyaslamak yok. somunları da

burada bir vidayı pivot seçeriz ve Quick sort gibi

yaparız. ↳ somunlar için vidaları vidalar için

somunları



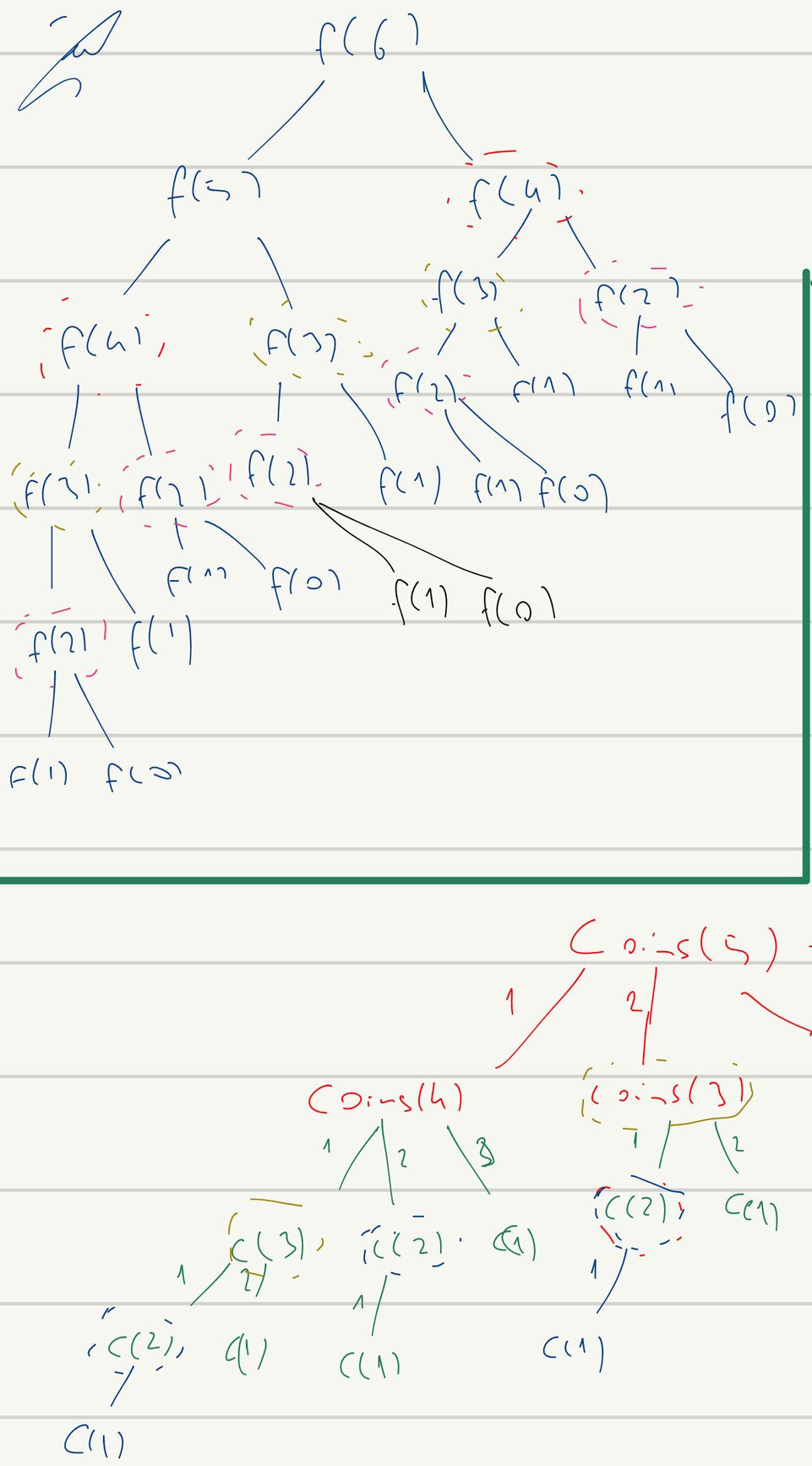
Arası Bitti

Hafta 9

Dynamic Programming

Fibonacci numbers

0 1 1 2 3 5 ...




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

Minimum coin change problem

coins = {1,2,5} value = 11 için Answer = 5+5+1

yol 1 → 1+1+1.....+1

1+1+1...+2

1+1+1...+2+2

⋮

5+5+1 → Exhaustive search → Exponantial algoritma

Bunun tersi de olabilir → P(11) = P(10)+1 gibi

→ O(n^value)

```
int minCoin(int* coins,int n,int value){
    int minCount = MAX;
    if(value == 0){
        return 0;}
     for(i=0;i<n;i++){
        if(coins[i]<=value){
            tmp = minCoin(coins,n,value-coins[i]);
            if(tmp<MAX && tmp+1<minCount)
                        minCount=tmp+1;}}
if(minCount == MAX)
       return -1;
else
    return minCount;}
```

Dynamic Programming

memorization

coins = {1,2,5}   value = 11   Answer = 5+5+1

0	1	2	3	4	5
0	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞
6	7	8	9	10	11

Top numbers (0–5) and bottom numbers (6–11) → Amount

→ minimum number of coins for each amount

↑ ilk durum

1TL

min(0+1, ∞)

0	1	2	3	4	5
0	1	2	3	4	5
6	7	8	9	10	11
6	7	8	9	10	11

→ min(10+1, ∞)

2TL

min(dizi[1]+1, dizi[2]+dizi[1], dizi[3]) → ?

0	1	2	3	4	5
0	1	1	2	2	3
3	4	4	5	5	6
6	7	8	9	10	11

→ min(dizi[3]+1, dizi[4]+dizi[1], dizi[5]) → ?

5TL

0	1	2	3	4	5
0	1	1	2	2	1
2	2	3	3	2	3
6	7	8	9	10	11

→ min(0+1, 3)

→ min(1+1, 3)

→ Burada aslında dizi[6] = min(dizi[1]+dizi[5], dizi[6])

→ dizi[7] = min(dizi[7], dizi[2]+1, dizi[6]+dizi[1]) → ?

C[P]: minimum number of coins (→ PTL)

x: first coin

C[P] = 1 + C[P−x]   (→ x için)

$$C[P] = \begin{cases} \min_{i:\, d_i < p} \{C[p-d_i]+1\} & \text{if } p>0 \\ 0 & \text{if } p=0 \end{cases}$$

C[8] = min(C[8−3], C[8−1], C[8−5])

```
C[0] = 0;
for p=1 to n do {          (n → Amount)
  min = ∞;
  for i=1 to k do {        (k → coin sayısı)
    if d[i] ≤ p then       (d → para dizisi)
      if (1+ C[p-d[i]] < min) then     (C → Adım dizisi)
        min = 1+ C[p-d[i]]; }
  C[p] = min;
}
```

$O(M * k)$

{1, 5, 2} → 3

0 - 1 Knapsack Problem

(hırsız sırt çantası problemi)

0 → sıfır → abandon (bırak)

1 → bir → select (al)

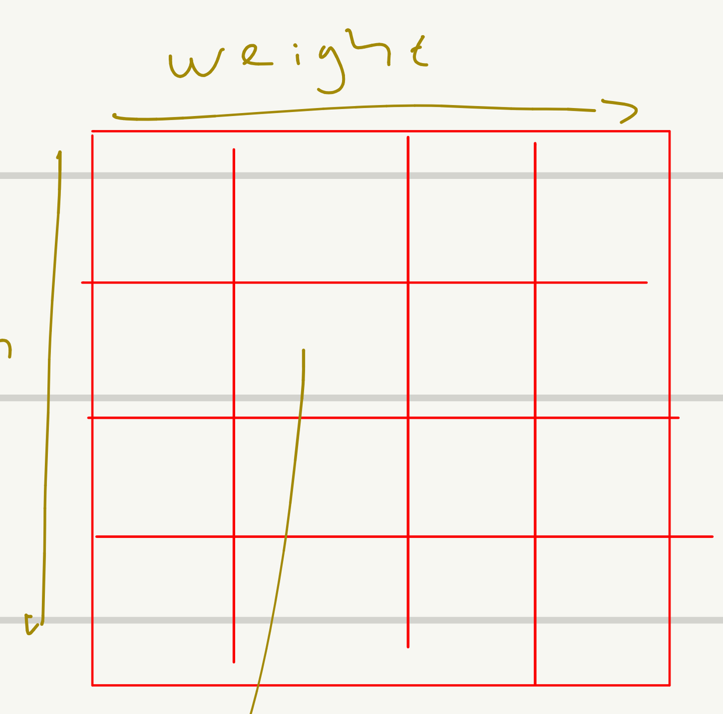
W = max weight

N = number of items

i^{th} item → value, weight

→ ~~Brute force~~ 2^n

maximize $\sum v_i$ subject to W



→ maximum profit from item 1 to item i, from weight 1 to weight j

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

$W = 5kg$

0 kiloya 2 kilo eklendiği için

12 mi 10 mu büyük

item \ weight	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

Case 1. thief takes item i

Case 2: thief doesn't take item i

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

(P(i, w) → not include item; P(i-1, w) → Before item i)

10. Hafta

0-1 Knapsack Problem

$W = 11$ item → value, → weight

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

weight →, item ↓, $\sum$ value → Profit

$P[i, w]$

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

Case 1: take item;

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

i → row, w → weight, $w - w_i$ → column, w_i → weight of item i

Case 2: not take item;

$P(i, w) = P(i-1, w)$ → $w_i > w$ $P[5,3] = P[4,3]$ → weight[5] > 3

→ Take $\max\{v_i + P[i-1, w-w_i], P[i-1, w]\}$

Hangilerini aldım?

$P[5,11] == P[4,11]$ almadım

else aldım $P[4, 11-k_5] == P[3, 11-k_4]$

$O(n * w)$ — n → item sayısı, w → top. ağırlık

Recurrance Relation

$$P[i,w] = \begin{cases} P[i,w] = 0 & \text{if } i = 0 \text{ or } w = 0 \\ P[i-1, w] & \text{if } w_i > w \text{ not taken} \\ \max(v_i + P[i-1, w-w_i], P[i-1, w]) \end{cases}$$

i: satır
w: sütun
kazanç

$v_i + P[i-1, w-w_i]$ → alırsam
$P[i-1, w]$ → almazsam

Edit Distance

① graffe

↓

graph, graft, grail, giraffe

② Computational Biology

AC G T G G A

A G T A T G A

I N T E # N T I O N

E X E C U T I O N

↓ ↓ ↓ ↓ ↓

Sil değiş d ekle değiş

toplam distance -> 5

ⓐ Substitute → same / change

change(x, i, a) → $x_1 \dots x_{i-1}\ a\ x_{i+1} \dots n$

$x_i = y_j \rightarrow ED[i,j] = ED[i,i]$

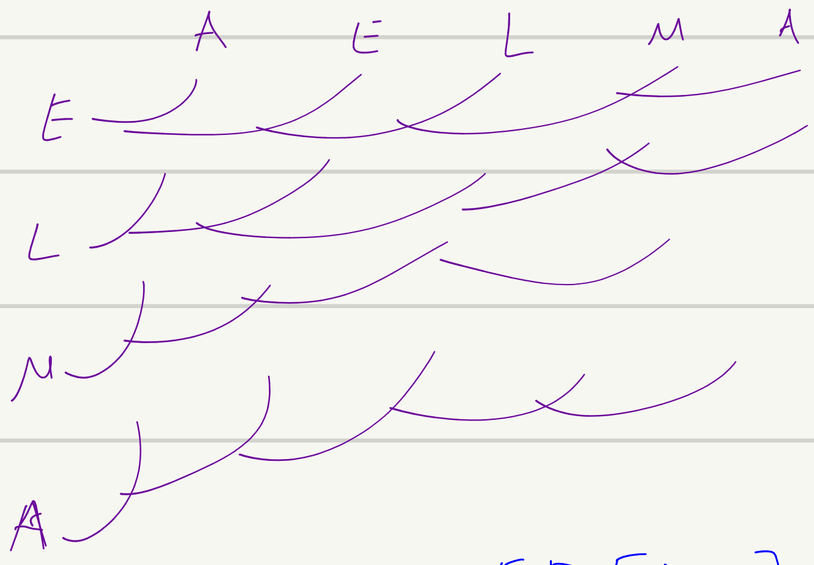
$x_i \neq y_j \rightarrow ED[i,j] = ED[i-1, j-1]$

ⓑ Delete: $x_1, x_2 \dots x_{i-1}, x_{i+1} \dots x_n$

$x_i \neq y_j \quad ED[i,j] = ED[i-1, j] + 1$

ⓒ Insert: $x_1, x_2 \dots x_i, a, x_{i+1} \dots x_n$

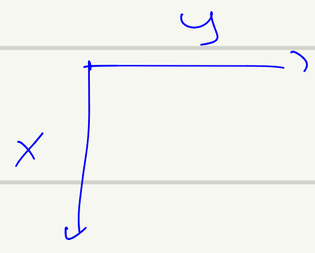
$ED[i,j] = ED[i, j-1] + 1$



hops: hepsinde

kayıp.

ED[i,j]



Returns Bağıntısı

$ED[i,j] = \min(ED[i,i-1] + 1,\ ED[i-1, i] + 1,\ ED[i-1, j-1] + 1)$

↳ eğer farklıysa

min { $ED[i-1][j] + 1$ or ↑ deletion

$ED[i][j-1] + 1$ or ← insertion

$ED[i-1][j-1] + cost \rightarrow x_i = y_j \quad cost = 0$

else $cost = 1$ ↖ Substitution

x = ARTS y = MATHS

		A	R	T	S
	0	1	2	3	4
M	1	1	2	3	4
A	2	1	2	3	4
T	3	2	2	2	3
H	4	3	3	3	3
S	5	4	4	4	3 → min

$O(N*M) \approx O(N^2)$

A R T * S
M A T H S eşsiz gibi...

Longest Common Subsequence

subsequence of a word

yabamoz
↓
yab
yaz
.
.

common subsequence

yabamoz yozgaz
yoz
yaz

Brute force → $O(n*2^m)$

X: ATCTGAT
Y: TGCATA
} LCS(X,Y) = TCTA or TGAT

LCS[i,j] x[i] ve y[j]'ye kadar longest common subsequence

Case 1: $x_i = y_j$ LCS[i,j] = LCS[i-1, j-1] + 1

X: ACGT... y: GC → LCS[i,j] = 1
↓ x_i ↓ y_j

Case 2: $x_i \neq y_j$ → x_i is not part of CS LCS[i-1, j]
→ y_j " " " " " LCS[i, j-1]

$$LCS[i,j] = \begin{cases} 0 & \text{if } i=0, j=0 \\ LCS[i-1, j-1]+1 & \text{if } i>0, j>0 \text{ and } x_i = y_j \\ \max(LCS[i-1,j], LCS[i,j-1]) & i>0, j>0 \quad x_i \neq y_j \end{cases}$$

X = "BACDB" Y = "BDCB"

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

diyagonelleri alıyorum B C B

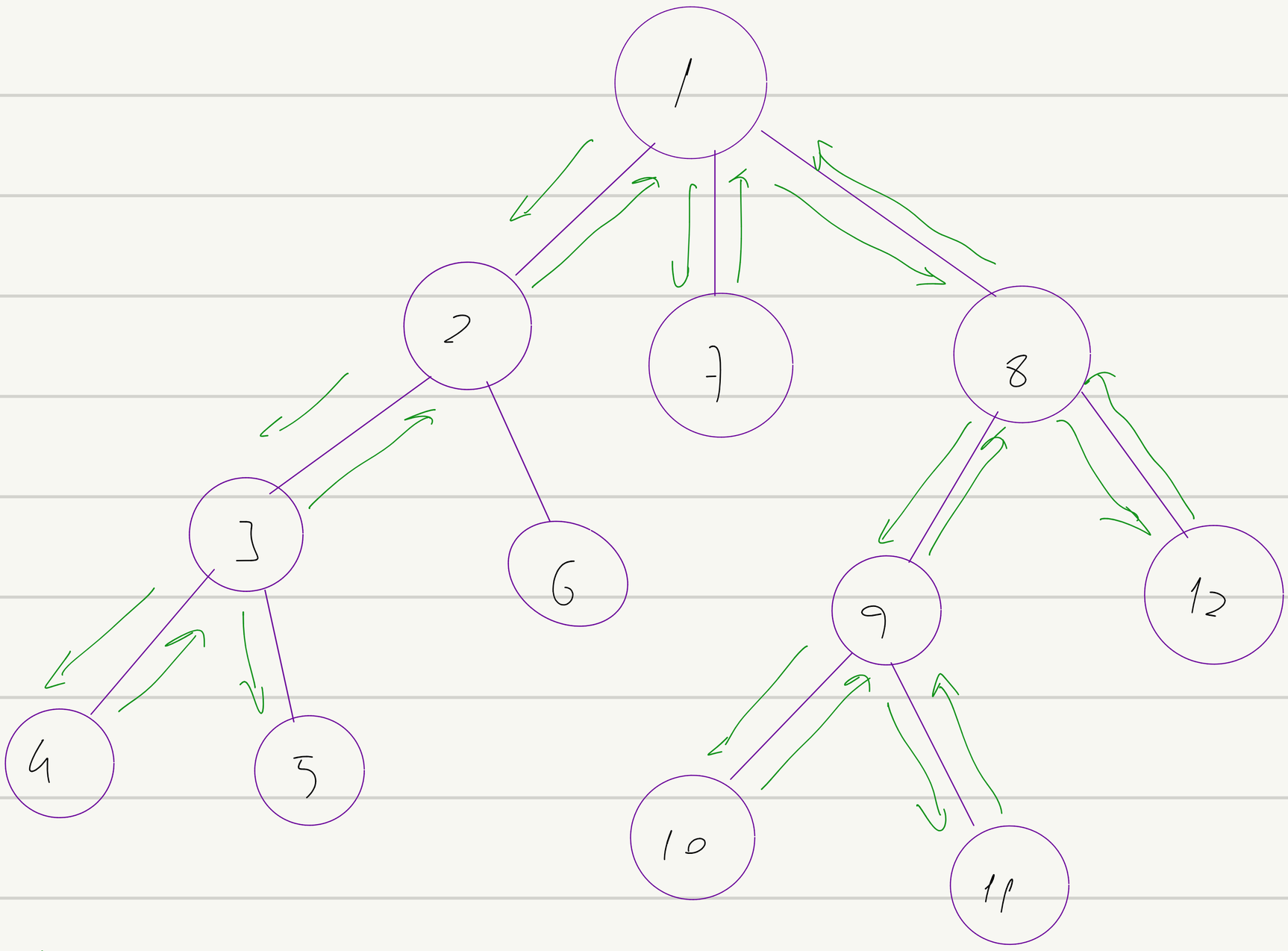
11. HAFTA

Graph traversal

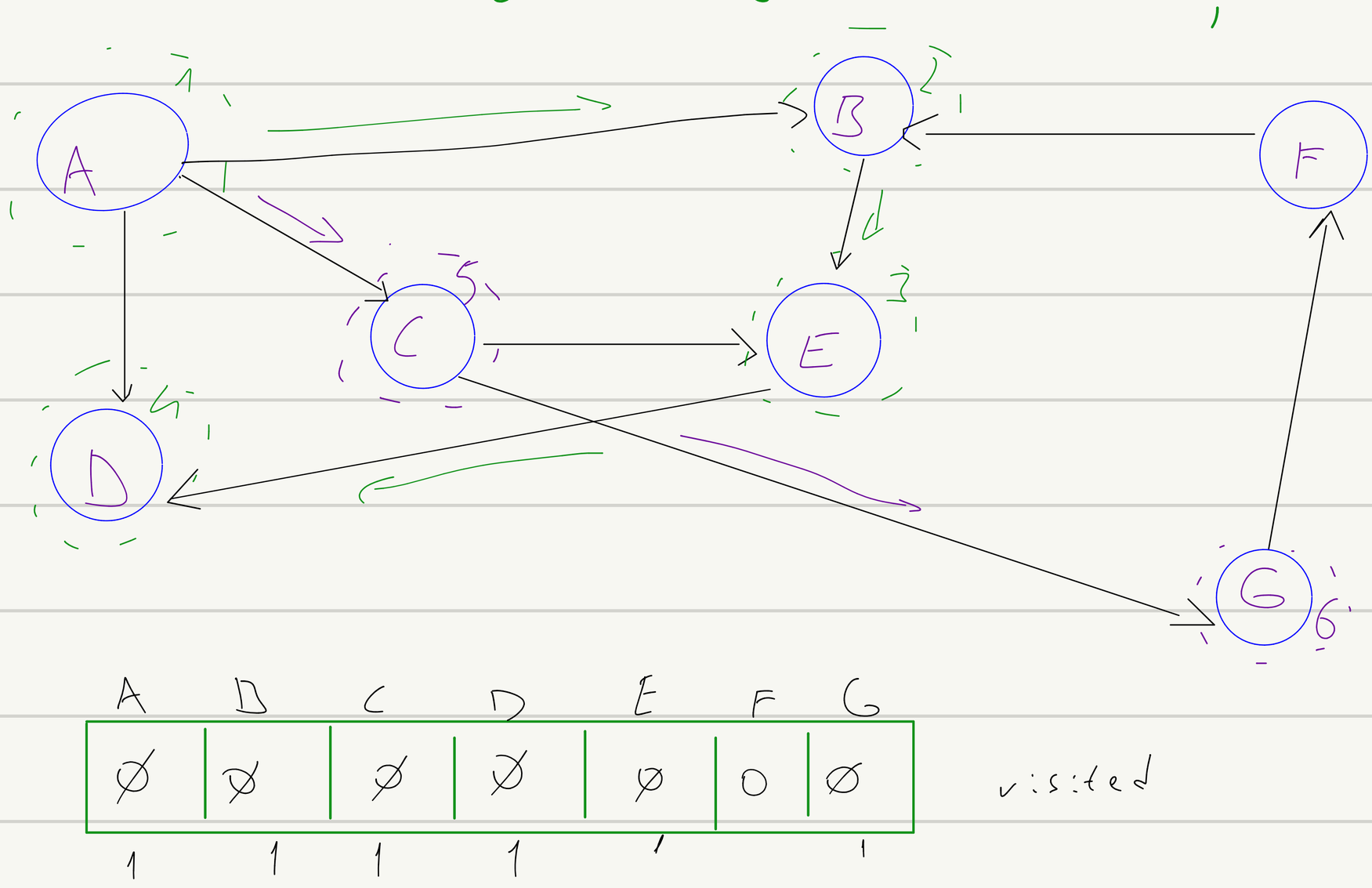
-> komşuluk matris: -> BFS -> Queue

-> komşuluk listesi: -> DFS

DFS -> STACK



A'dan G'ye yol var mı?



A	B	C	D	E	F	G
∅	∅	∅	∅	∅	0	∅
1	1	1	1	1		1

visited

Recursive DFS

```
DFS(G,S){
    mark s as visited
    for all neighbours w of s in graph G
        if w is not visited
            DFS(G,w) }
```

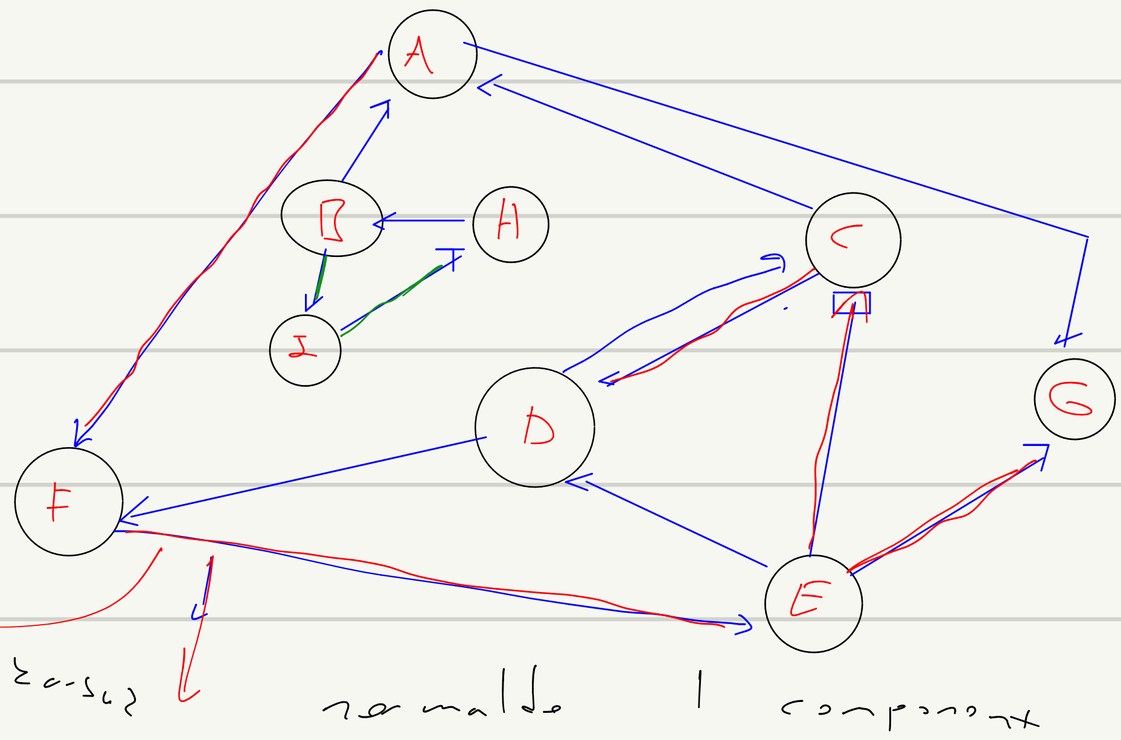
(G: graph, S: inicial point)

Iterative DFS

```
DFS(G,s)
    push(s)
    mark s as visited
    while stack is not empty
        v = pop()
        for all neighbours w of v in graph G
            if w is not visited
                push w
                mark w as visited
```

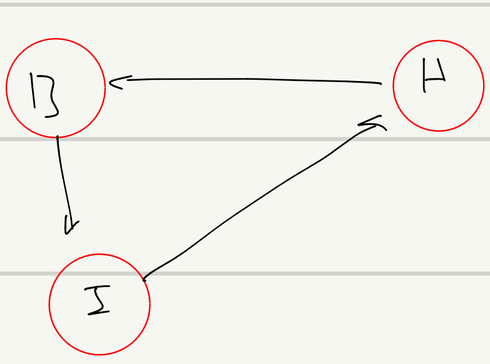
Connected Component

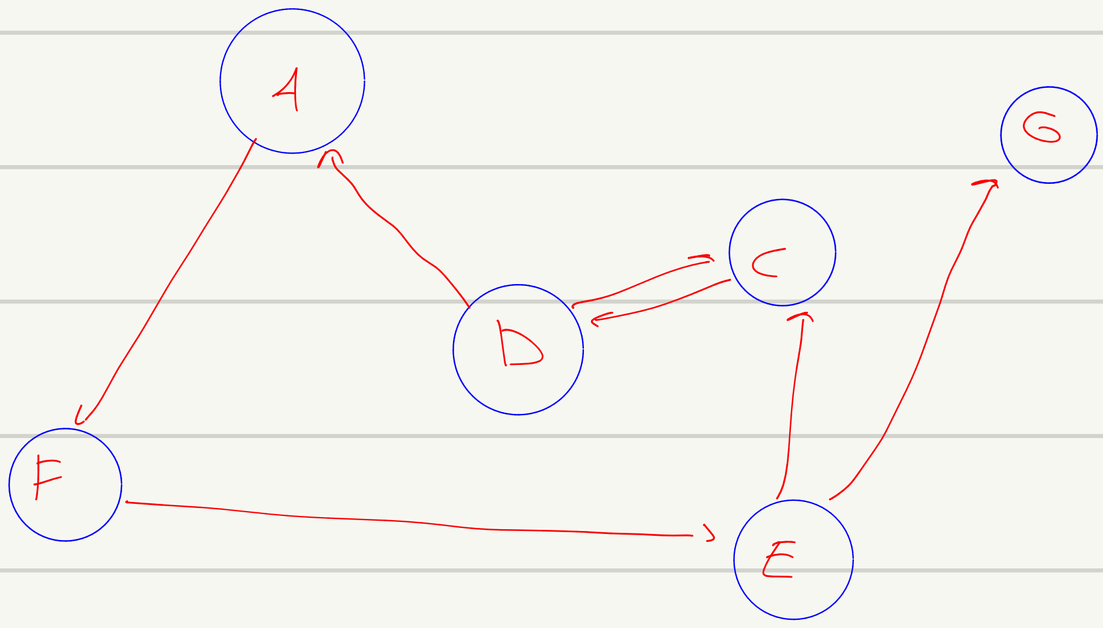
Gonlo Graph



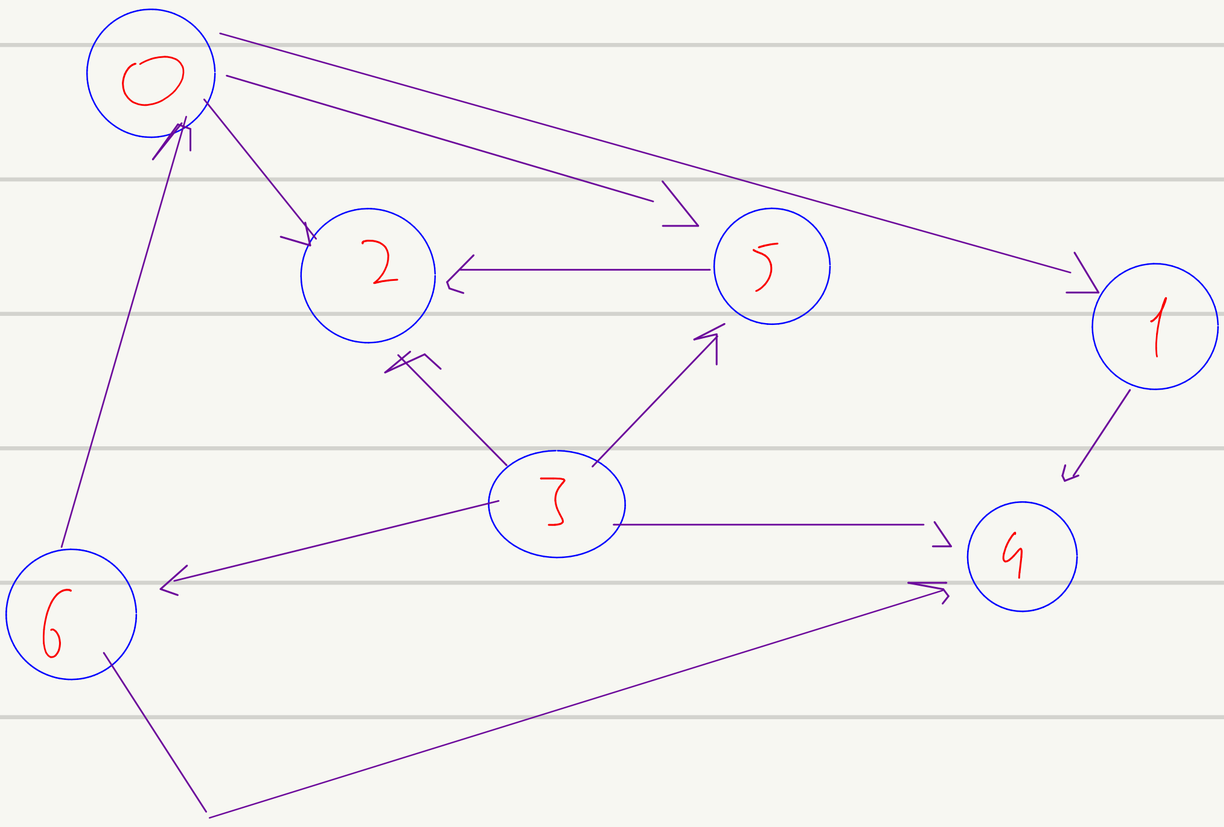
Not A can visit

normally 1 component





Topological Sort DFS → Acyclic ie Directed graph



Adjacency list

0 → 1 → 2 → 5

1 → 4

2 → x

3 → 2 → 4 → 5 → 6

4 → x

5 → 2

6 → 0 → 4

indegree → 0 in 1 (6's on)

outdegree → 0 "] (1,2,5)

dfs(0)

dfs(1)

dfs(4)

4 done

1 done

dfs(2)

2 done

dfs(5)

check 2

5 done

0 done

dfs(3)

c 2

c 4

c 5

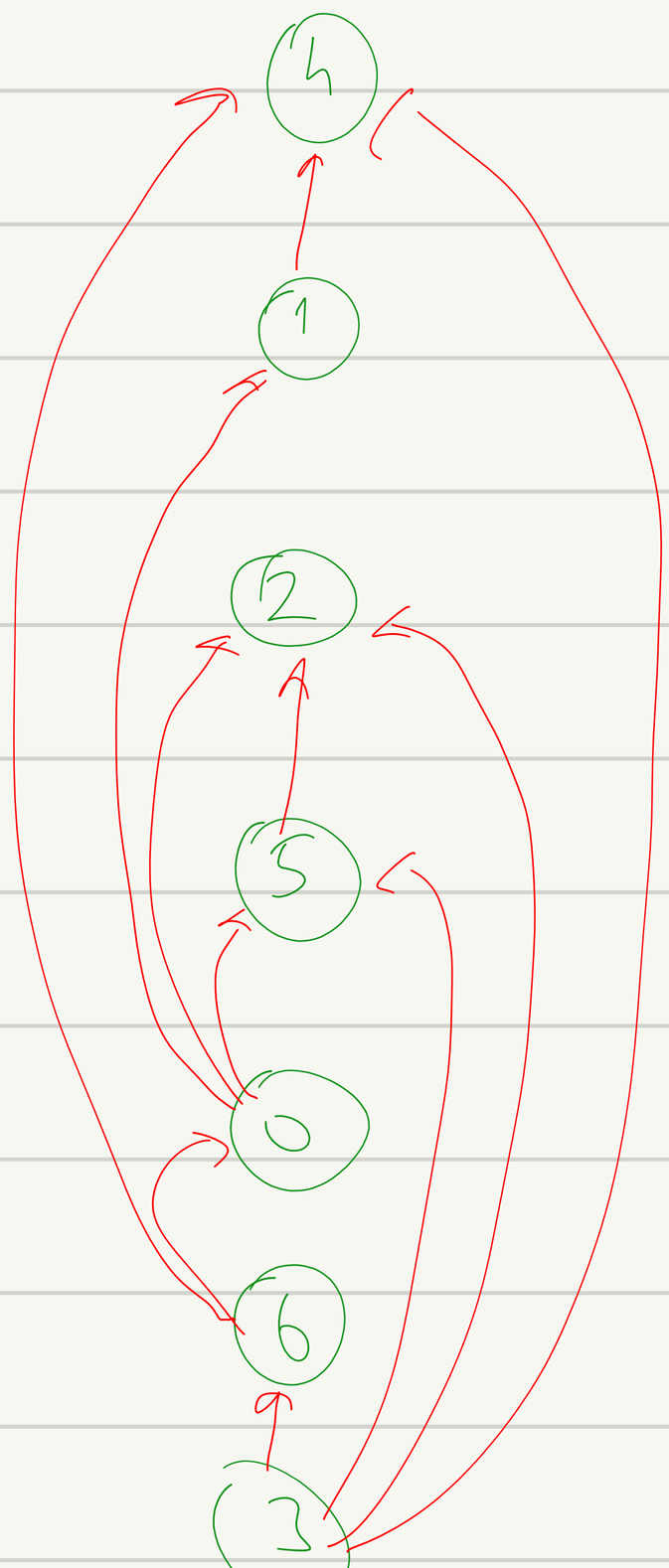
dfs(6)

c 0

c 4

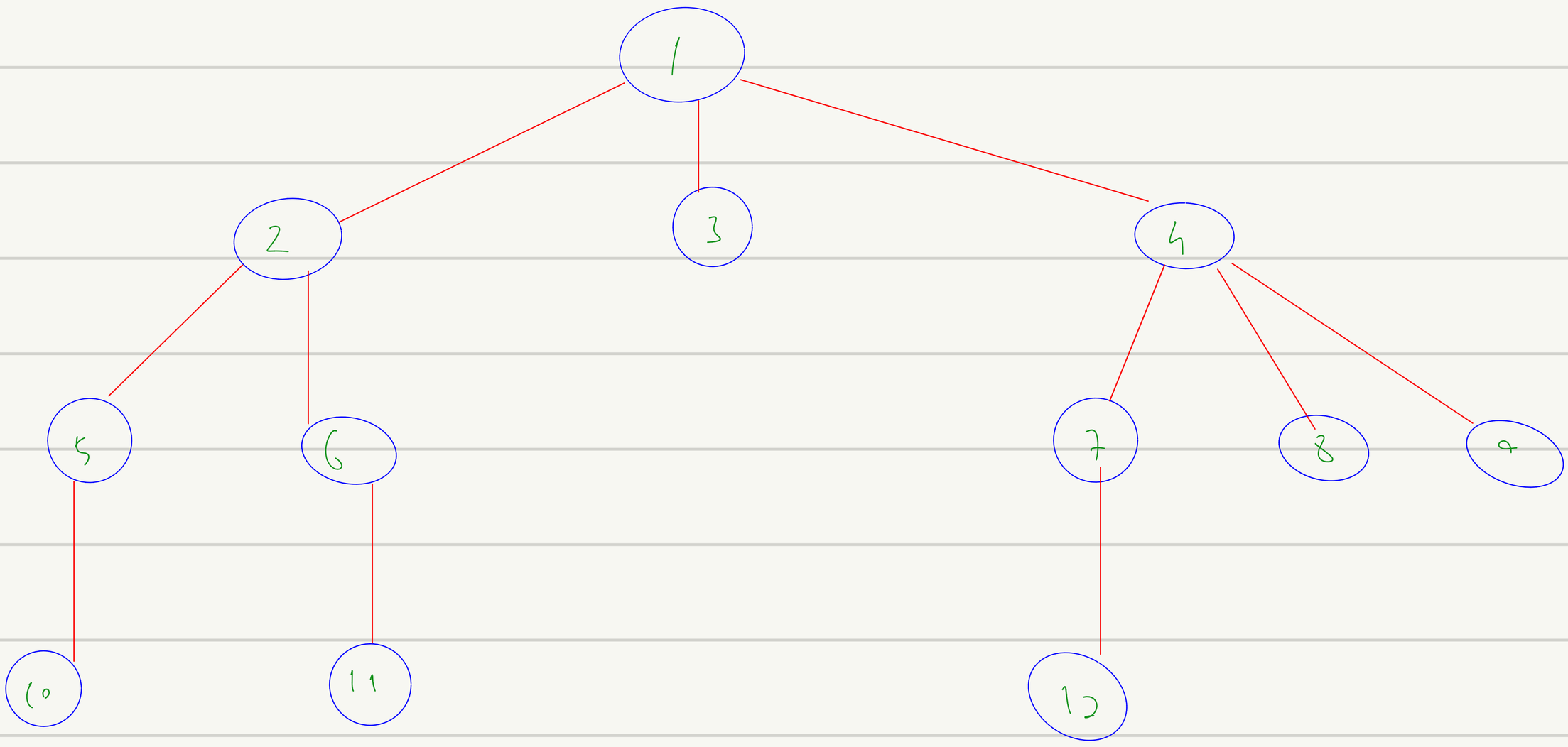
6 done

3 done

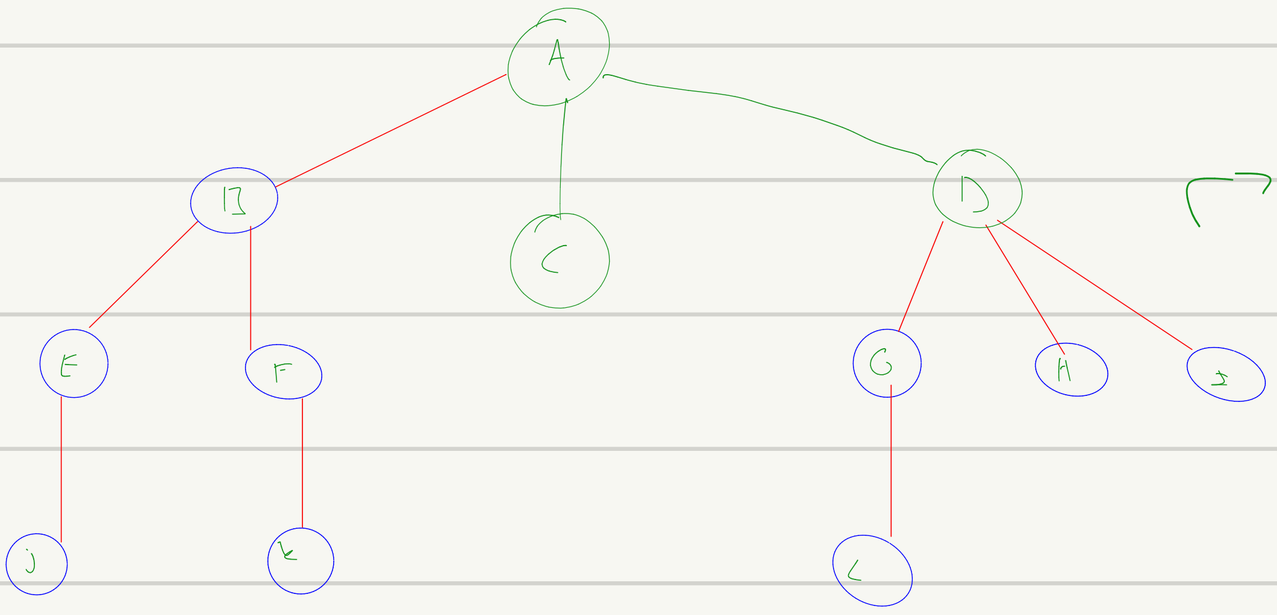


flood fill, cycle detection, topological sort gibi problemler DFS ile çözülür, bağlı connected component de

BFS → Queue




```
BFS(G,s){
    enqueue(s)
    mark as s visited
    while(Queue is not empty)
        v = dequeue(s)
        for all neighbors w of v in G
            if w isn't visited
                enqueue w
                mark as w visited
}
```



→ A'dan başlayıp en kısa yol

DFS

A	B	C	D	E	F	G	H	I	J	K	L	
1	1	1	1	1	1	1	1	1	1	1	0	

→ Visited

A	B	C	D	E	F	G	H	I	J	K	L	
0	1	1	1	2	2	2	2	2	3	3	0	

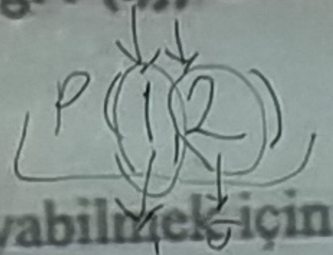
→ Path

Google haritalar, P2P network gibi uygulamalarda BFS kullanılır.

LAB

Dinamik Programlama Örnek:

A ve B takımları, içlerinden biri **n** galibiyet alana kadar maç yapacaklardır. A takımının bir maçı kazanma olasılığı her maç için **p**, kaybetme olasılığı ise **1-p**'dir. Dolayısıyla beraberlik ihtimali mevcut değildir. A takımının seriyi kazanmak için **i** tane daha maç kazanması, B takımının da seriyi kazanmak için **j** tane maç kazanması gereken durumda A'nin seriyi kazanma olasılığı **P(i,j)**'dir.



- P(i,j)'yi kullanarak **dinamik programlama** ile algoritma tasarlayabilmek için **rekürans bağıntısını** yazınız.

- A takımının bir maçı kazanma olasılığının 0.4 olduğu durumda 7 maçlık seride (4 alan kazanır) **A takımının kazanma ihtimalini** hesaplayınız. (Dinamik programlama yaklaşımını kullanınız.)

$$P(i,j) = p * P(i-1, j) + (1-p) * P(i, j-1) \qquad i, j > 0$$

$$P(0,j) = 1 \qquad i = 0$$

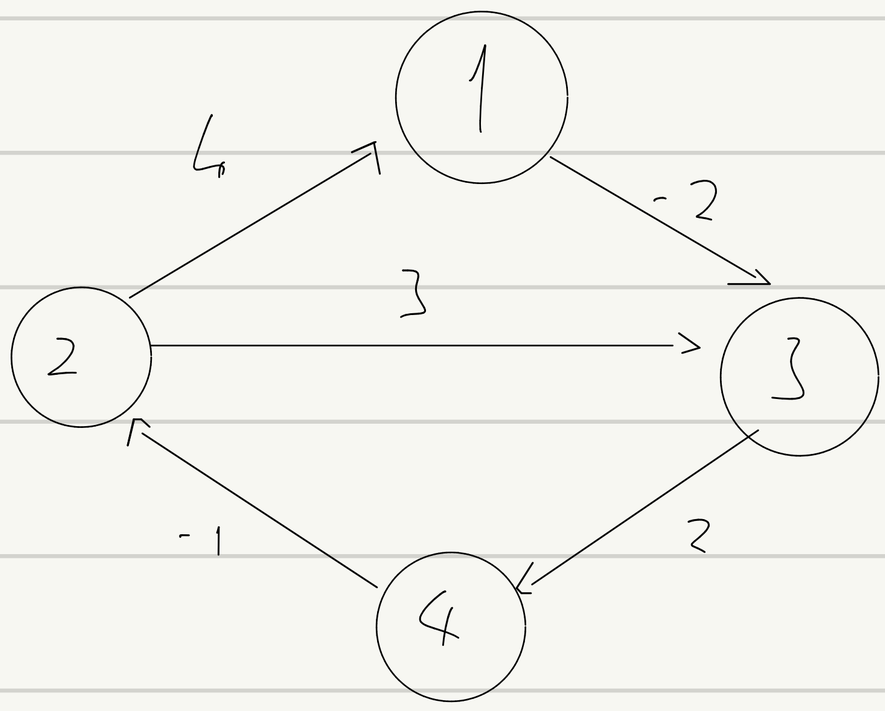
$$P(1,0) = 0 \qquad j = 0$$

i \ j	0	1	2	3	4
0	0	1	1	1	1
1	0	0.4	0.64	0,784	0,8704
2	0	0.16	0.352	0,5248	0,66304
3	0	0,064	0,1792	0,31744	0,45568
4	0	0,0256	0,08704	0,1792	0,289792

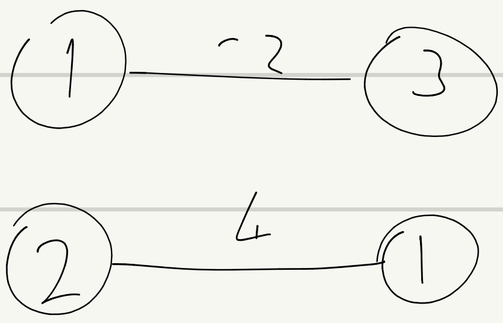
P(p)

```
for (i=0; i<n; i++)
    diz[i][0] = 0;
    diz[0][i] = 1;

for(i=1;i<n;i++)
    for(j=1;j<n;j++)
        diz[i][j] = diz[i-1][j] * p + diz[i][j-1] * (1-p)
```

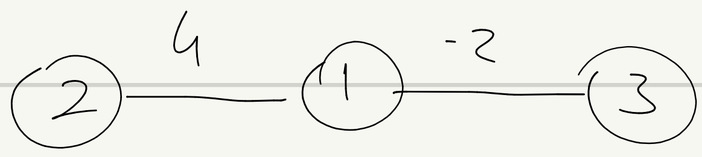
Graph - Floyd Warshall



k=0



k=1



min(distance[i][j], dist(i,k)+dist(k,j))

dist[2,3] = 5
dist[2,1]+dist[1,3]

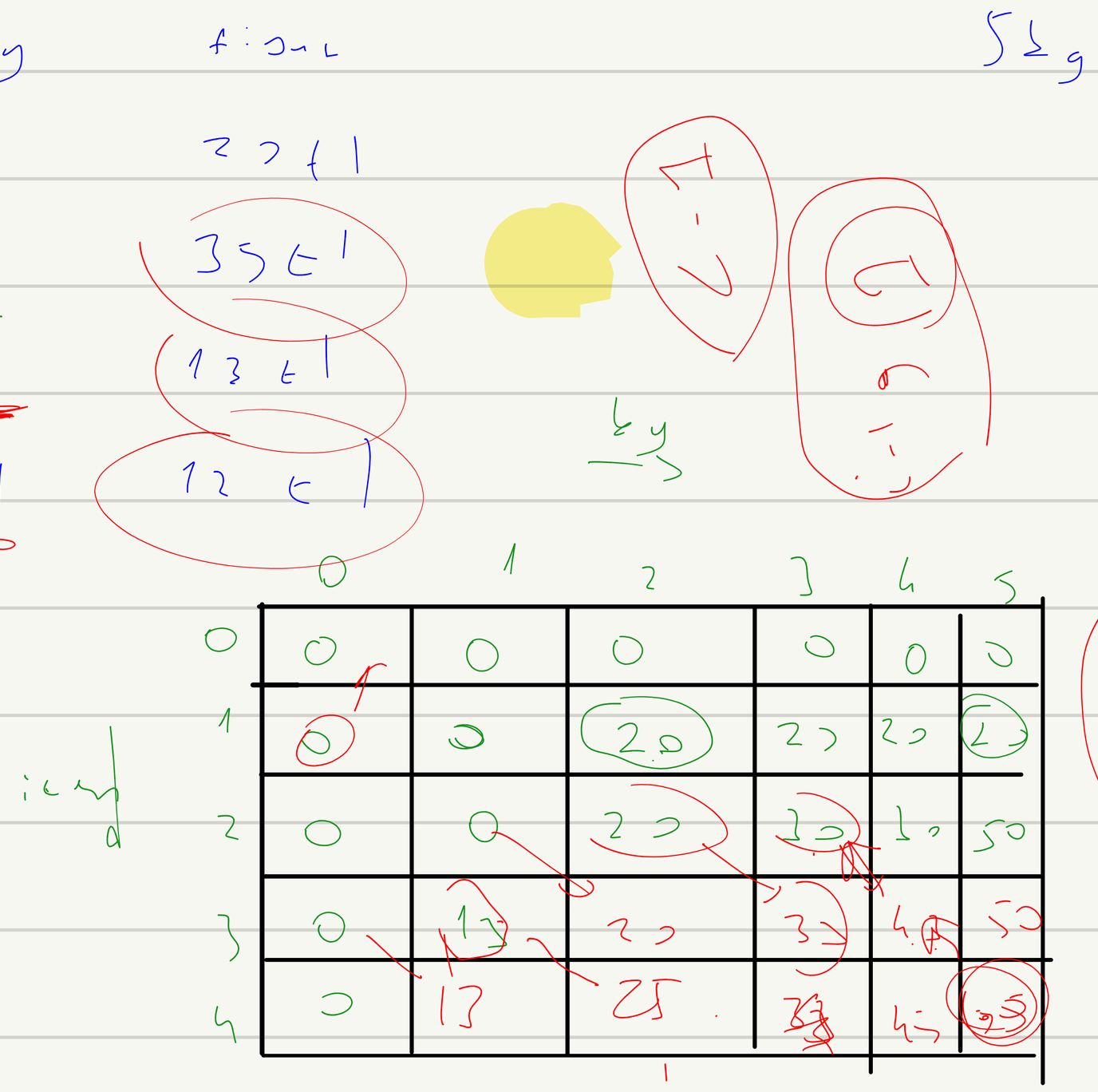
k=0	1	2	3	4
1	0	∞	-2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	-1	∞	0

k=1	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	∞	-1	∞	0

k=2	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	3	-1	1	0

W (graph, V, E)

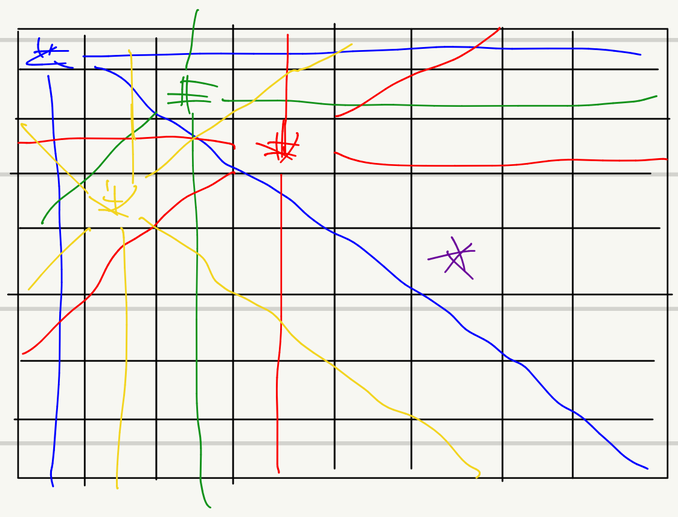
```
for i=0; i<V; i++
    for j=0; j<V; j++
        matrix[i][j]=∞

for each edge (u,v)
        dist[u][v] = weight[u][v]

for i=0; i<V; i++
        dist[v][v]=0

for k=0; k<V; k++
    for j=0; j<V; j++
        for i=0; i<V; i++
            if( dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] = dist[i][k] + dist[k][j]
```


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

12. Hafta

Back tracking

N-Queen Algorithm



Asıl mantık 8 tane vezir yerleştirirken, yerleşmezse geri dönmek gerekiyor.

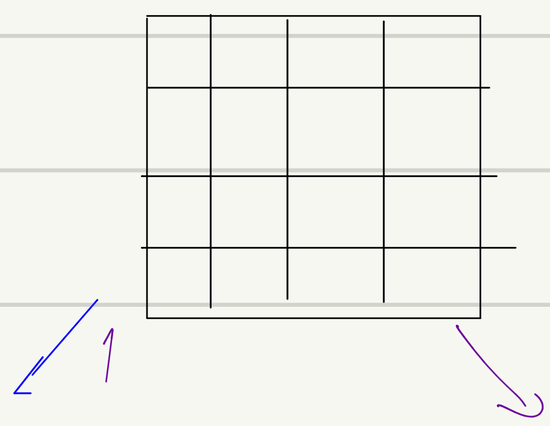
① Brute force'da $\binom{64}{8}$ → 4 milyar deneme

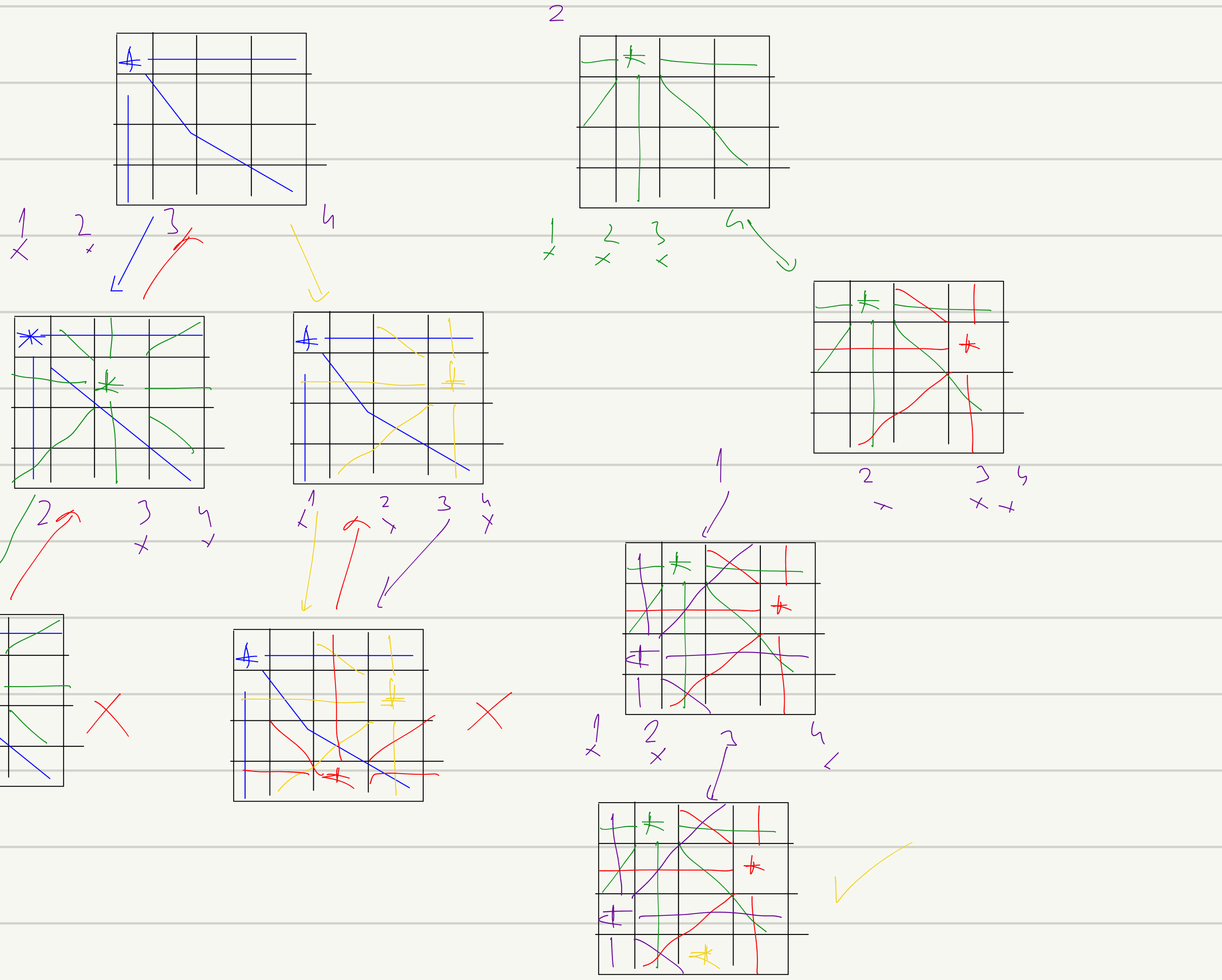
② Satır kontrolü yaparsak $n^n \rightarrow 8^8 \rightarrow$ 16 milyon

③ Satır ve sütun kontrolü yaparsak → n! → 40 bin

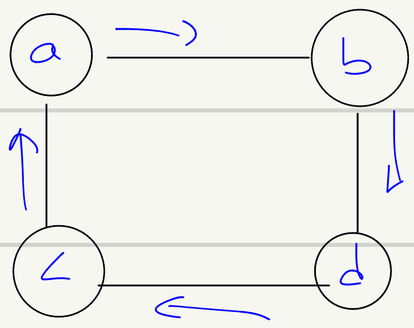
④ Yerleştirmediğinde bir önceki duruma dönme işlemi → 2 bin
(→ back tracking)

State space tree





Hamiltonian Circuit → DFS gibi;



→ başladığın noktaya her noktadan bi kere değerek dönüş

State space tree

Branch and Bound → BFS

4 kişi var ve 4 işe yerleştirilecek

State Space Tree

	Job 1	Job 2	Job 3	Job 4	
C =	9	2	7	8	Ahmet
	6	4	3	7	Mehmet
↓	5	8	1	8	Cevdet
Para	7	6	9	4	Serbest

Knapsack Problem

W = 10 kg

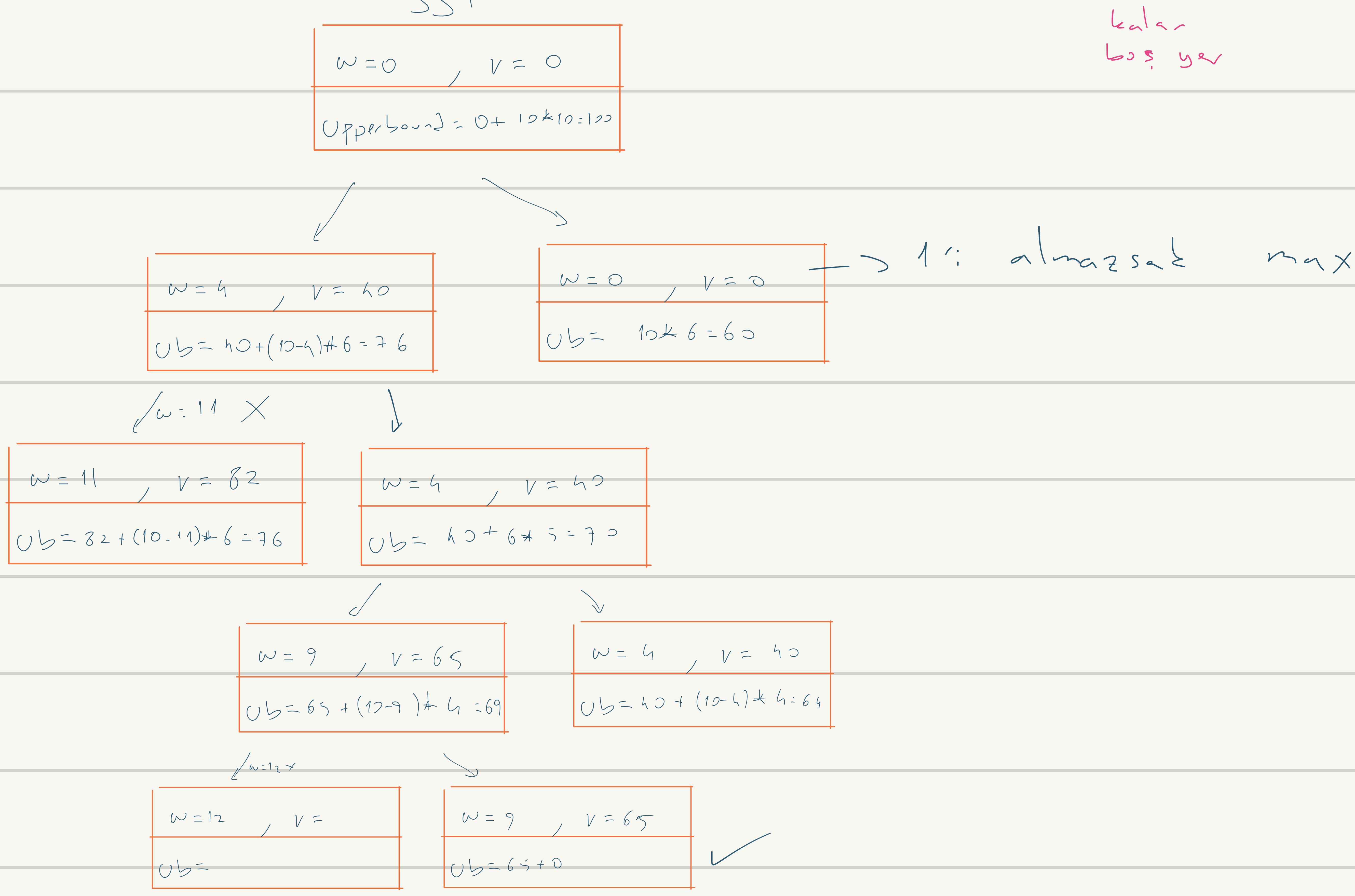
item	weight	value	vi/wi
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

$v_1/w_1 >> v_2/w_2 >> v_3/w_3 >> \dots$ için

upper bound = $v + (W - w)(v_{i+1}/w_{i+1})$

- v → anlık kazanç
- (W - w) → geriye kalan boş yer
- (v_{i+1}/w_{i+1}) → en yüksek oran

SST



13. HAFTA SON

Polinomsal -> $O(n^{1 milyar})$ -> efektif değil

efektifler -> $O(n^2)$, $O(n^3)$, $O(n \log n)$

efektif olmayanlar -> $O(2^n)$, $O(n^n)$, $O(n!)$, $O(n^{1 milyar})$

$n^{10 milyar}$ -> tractable gözüküyor ama gerçekte dünyada değil!

$n^{\log n}$ -> in " " " " " tractable

NP -> Non det. polinomial

Traveling Salesman -> Başladığın yerden her yere en fazla 1 defa uğrayarak, her yeri gezmek.

Hamiltanian circuit -> Kenar ağırlık. traveling Salesman her durum denemek lazım

- Dynamic -> $2^n \cdot n$
- Brute force -> $n!$

Problems

- Decision Problems
- Optimization Problems

Deterministic -> çıktısı belli

↳ max elemanı

Nondeterministic -> çıktısı belirsiz

↳ en yakın yol

```
NSort(A,B)
for(i=0;i<N;i++)
B[i]=0;

for(i=0;i<N;i++)
j=rand(1..N)
if(B[j]!=0) then failure
B[j]=A[j]
end for
for(i=0;i<n-1;i++)
if(B[i]>B[i+1]) then failure;
end for
print(B)
success
O(1)
```

```
NSearch(A,x)
j=choice(1..n)      -> (1) guessing
if(A[j]=x)          -> (2) Verification
print(j)
success             -> (3) Output
endif
print(0)
failure
```

Comlexity O(1)

↑

NP

$P \subseteq NP$ NP → polinomsal zamanda dogrulanabilen algoritmalar.