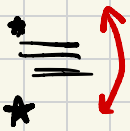
Algoritma yazılırken sağlaması gereken şartlar:

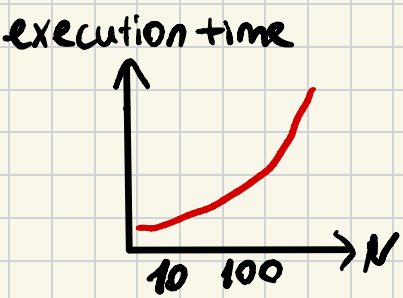
1. Sonlu adımda bitmeli
2. Doğru olmalı
3. Efektif olmalı → zaman, yer
4. Genel olmalı
5. Basit olmalı

Execution Time Factors

① 2 nokta arası zamanı ölçmek



② 10, 100, 100 eleman için çalıştırmak



Çalışma zamanını neler etkiler?

1. hardware
   - → CPU speed
   - → amount of memory
2. Compiler ex) assembly > C > Java
   - → efficiency of code generation
3. Data
   - → number of items
   - → initial order
4. Algorithm

System dependent effects

hardware : memory, cache ...

software : compiler, garbage collector...

system: OS, network, other applications

System independent effects:

algorithm

✱ Growth rate of some algorithm

N	logN	N	NlogN	N^2	2^N	N!
10	0,003μs	0,01μs	0,033μs	0,1μs	1ms	
100 milyon	0,027μs	0,1sec	466sec	115day		

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

operation	frequency
variable decleration	3
assignment	2
less than compare	n+1
equal to compare	n
array access	n
increment	n to 2n
+	
	4n+6 to 5n+6

ex) $4n^2 + 3n + 5$

1. ignore constant terms → $4n^2 + 3n$
2. ignore coefficients → $n^2 + 3n$
3. pick the most significant term → n^2

Asymptotic Analyse → Growth rate

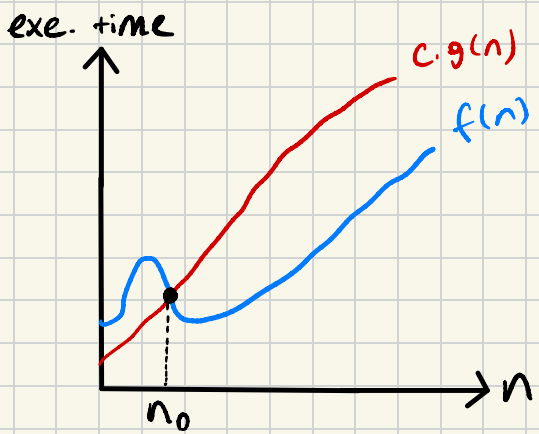
Asymptotic Notation → Big oh (O)
Big Omega (Ω)
Big Theta (Θ)

Big-Oh Notation : Asymptotic Upper Bound

$f(n) \in O(g(n))$ if $c < n$ constant, $n \geq n_0 \geq 1$

$O(g(n))$ upperbound on the growth of a function $f(n)$

$0 \leq f(n) \leq g(n)$



ex) $f(n) = 10n + 30$ $O(n)$ midir?

$g(n) = n$

$10n + 30 \leq c.n$

$30 \leq n(c - 10)$ $n_0 = 1 \Rightarrow c = 40$
$n_0 = 2 \Rightarrow c = 80$
$n_0 = 3 \Rightarrow c = 120$

✓

ex) $f(n) = 10n + 30$ $O(n^2)$ midir?

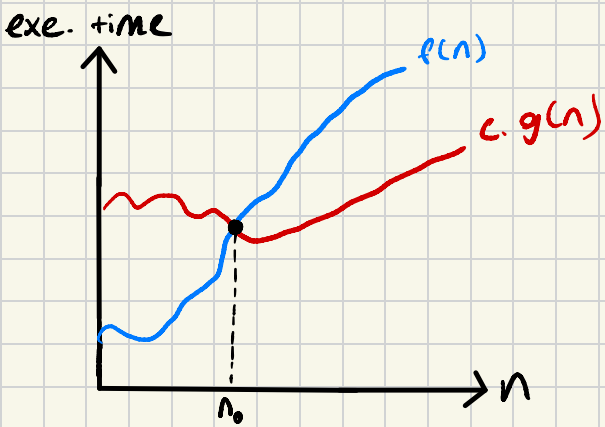
$10n + 30 \leq c.n^2$

$30 \leq n(cn - 10)$ $n_0 = 1 \Rightarrow c = 40$ ✓

ex) $f(n) = n^2 + n$ $g(n)$ için sağlar mı?

$n^2 + n \leq n$ ✗ hayır

Big-Omega Notation : Asymptotic Lower Bound



$f(n) \geq c.g(n)$ $c > 0$ constant

$n_0 \geq 1$, $n \geq n_0$

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

ex) $f(n) = 5n^2 \in \Omega(n^2)$ midir?

$5n^2 \geq cn^2$

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

$3 \geq 0$ ✓

ex) $f(n) = 5n^2 \in \Omega(n)$ midir?

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

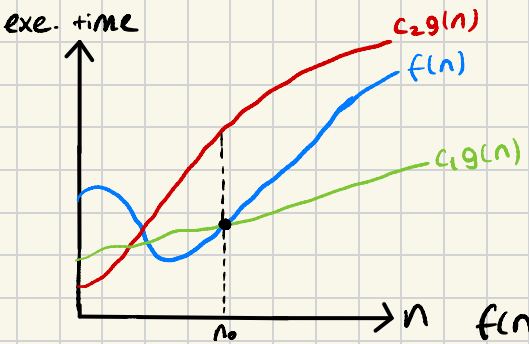
$n(5n-c) \geq 0$ $\quad n_0 = 1$ ve $c = 3$ iken

$\downarrow$

$5n^2 \geq 3n$ ✓ $\quad 2 \geq 0$ ✓

✓

Big-Theta Notation: Asymptotic Tight Bound



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

$c_1 > 0,\ c_2 > 0,\ c_2 \geq c_1,\ n \geq n_0 \geq 1$

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

$f(n)$ is asymptotically equal to $g(n)$

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

① $f(n) \in O(n)$?

$2n+5 \leq c \cdot n$

$5 \leq n(c-2)$ $\quad n_0 = 1$

$7 \leq c \Rightarrow c = 7$

c_2

② $f(n) \in \Omega(n)$?

$2n+5 \geq c \cdot n$

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

$7 \geq c \quad \Rightarrow c = 2$

c_1

$2n \leq 2n+5 \leq 7n$ ✓

ex) $f(n) = 4n \in \Omega(n^2)$ midir?

$4n \geq cn^2$ sağlamaz.

$f(n) = 4n^2 \in O(n)$ midir?

$4n^2 \leq cn$ sağlamaz

O halde:

$4n^2 \in \Theta(n)$ midir? Hayır. $O(n^2)$ sağlamaz.

$4n^2 \in \Theta(n^3)$ midir? Hayır. $\Omega(n^3)$ sağlamaz.

⇓ aynı derece olmalı haliyle

* Best case
Worst case
Average case

* Bazı toplam formülleri

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

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

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

→ finding max element in an array

best case : n

worst case: n

average case: n

} $\Theta(n)$

→ sequential search in an unordered array

```
i=0 ; // x: aranan sayı
while ( (i<n) && (a[i]!=x))
    i++ ;
```

$C_{best}(n) = 1$

$C_{worst}(n) = n$

$C_{average}(n) = ?$

↳ Çözüm: probability = p successful search : $0 \le p \le 1$

i^{th} position : $\frac{p}{n}$

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

1 · p/n : # of compare

n · p/n : # of compare

n · (1−p) : unsuccesful search

if element is in the array

$$= \frac{p}{n}\left[1+2+\cdots+n\right] + n(1-p)$$

① $p=1 \Rightarrow$ successful search

$$= \frac{1}{n} \cdot \frac{n(n+1)}{2} + 0 = \frac{n+1}{2}$$

② $p=0 \Rightarrow C_{worst}(n) = n(1-p) = n$

$O(n)$ ve $\Omega(1)$ olur.

Θ söylenemez.

→ insertion sort

best case : n → $\Theta(n)$ [her zaman n karşılaştırma]

worst case : n^2 → $\Theta(n^2)$

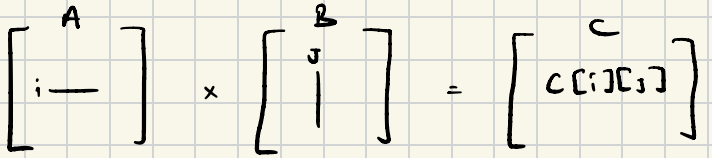
average case : $n^2\left(\frac{1}{4}n^2 + \frac{3}{n} - 1\right)$ → $\Theta(n^2)$

$O(n^2)$

description	order of growth	example
constant	1	a = lal
logarithmic	log n	binary search
linear	n	finding max
linearithmic	n log n	merge s, quick s, heap sort
quadratic	n^2	selection sort
cubic	n^3	matrix multiplication
exponential	2^n	exhaustive search (knapsack, travelling salesman)

2^n ↓ non deterministic polinomsal

Mathematical Analyses of Nonrecursive Algorithms



$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (n-1-0+1) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n$$

$$= \sum_{i=0}^{n-1} n.n = n^3$$

$t(n) = c_m \cdot M(n) = c_m \cdot n^3$

time ↓ constant

* Running time of the algorithm

$T(n) = c_m \cdot n^3 + c_A \cdot A(n)$ → $O(n^3)$

↓ n^3 $\Theta(n^3)$

Mathematical Analyses of Recursive Algorithms

→ factorial function

$n! = 1 \cdot 2 \cdot \ldots \cdot n = n \cdot (n-1)!$

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

initial condition (durma noktası)

basic operation

Recurance Relation

$$M(n) = \underbrace{M(n-1)}_{\text{factorial}(n-1)} + \underbrace{1}_{n * (n-1)}$$

$n = 0 \Rightarrow M(0) = 0$ çünkü no multiplication

Backward substition

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

$$= [m(n-2) + 1] + 1 = m(n-2) + 2$$

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

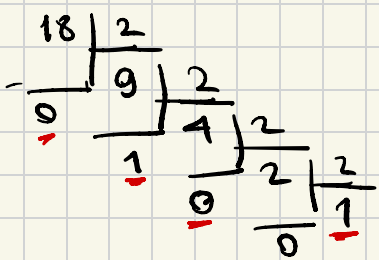
$$\vdots$$

$$= m(n-i) + i \qquad n = i \text{ için}$$

$$= m(0) + n$$

$$= 0 + n = n \in \Theta(n)$$

→ number of binary digits in a decimal integer



→ 10010
5 digit

```
int binary (int n){
   int count = 1;
   while (n>1) {
      count ++;
      n /= 2;
   }
   return count;
}
```

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

Recurance Relation: $A(n) = A(n/2) + 1$ for $n>1$

$A(1) = 0$ for $n=1$

Backward substition:

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

$$= [A(n/4) + 1] + 1 = A(n/4) + 2$$

$$= [A(n/8) + 1] + 2 = A(n/8) + 3$$

$$\vdots$$

$$= A(n/2^i) + i \qquad n = 2^i \Rightarrow \log_2 n = i$$

$$= A(1) + \log_2 n$$

$$= 0 + \log_2 n = \log_2 n$$

★ FORWARD SUBSTITION EXAMPLE

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

$n=1 \Rightarrow T(1) = 1$

$n=2 \Rightarrow T(2) = 2T(1) + 1 = 3$

$n=3 \Rightarrow T(3) = 2T(2) + 1 = 7$

$n=4 \Rightarrow T(4) = 2T(3) + 1 = 15$

$T(n) = 2^n - 1$ $n>1$ için $T(n) \in O(2^n)$

* Master Theorem

$$T(n) = a \cdot T(n/b) + f(n) \qquad T(1) = c$$

- a: # of subproblems
- n/b: size of each subproblems
- c: constant

$a \geq 1$, $b > 1$, $d \geq 0$

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

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

↳ bu ifadeler O ve Ω için de geçerli

ex) $T(n) = T(2n/3) + 1$

$a = 1$, $b = 3/2$, $d = 0$

$b^d = \left(\frac{3}{2}\right)^0 = 1 \Rightarrow a = b^d \Rightarrow \Theta(n^d \log n) = \Theta(n^0 \log n)$

$= \Theta(\log n)$

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

$a = 2$, $b = 2$, $d = 1$

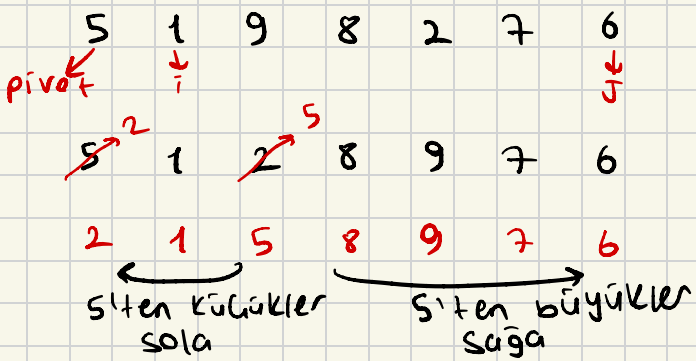
$b^d = 2^1 = 2 \Rightarrow a = b^d \Rightarrow \Theta(n^d \log n) = \Theta(n^1 \cdot \log n)$

$= \Theta(n \log n)$

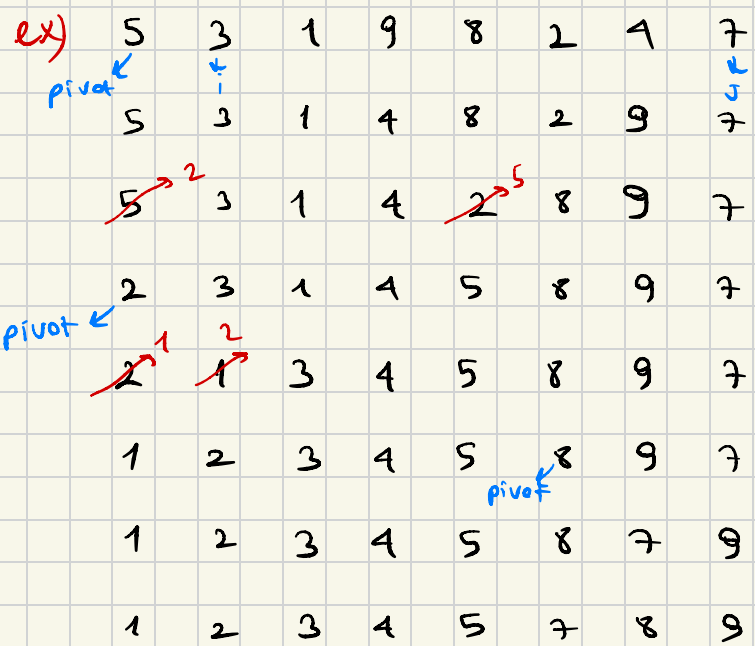
Selection sort
Bubble sort
Insertion sort
} worst case: n^2

Quick sort → average case: $n \log n$

→ Quick sort



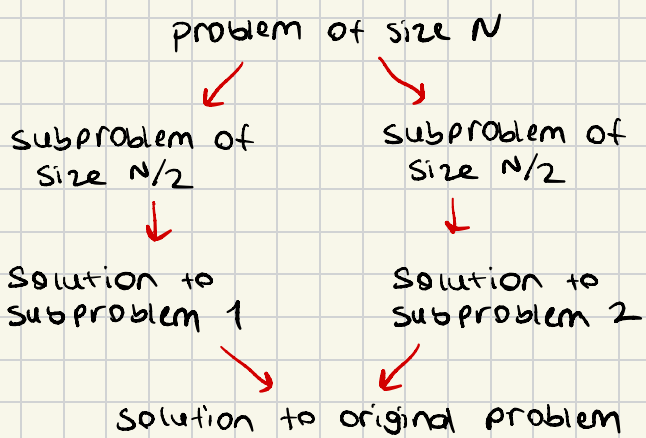
ex)

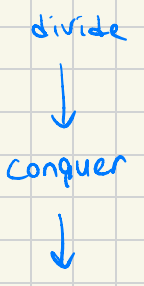


ex)

5	4	2	20	3	2	6	10
pivot							
~~5~~	4	2	2	~~3~~	20	6	10
3	4	2	2	5	20	6	10
pivot							
~~3~~	2	~~2~~	4	5	20	6	10
2	2	3	4	5	20	6	10
pivot							
2	2	3	4	5	20	6	10
2	2	3	4	5	20	6	10
2	2	3	4	5	20	6	10
					pivot		
2	2	3	4	5	~~20~~	6	~~10~~
2	2	3	4	5	10	6	20
					pivot		
2	2	3	4	5	6	10	20

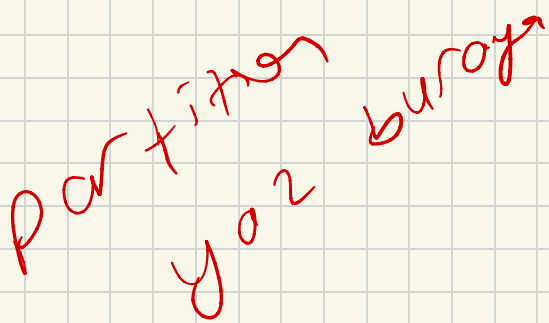
Divide & Conquer Algorithms



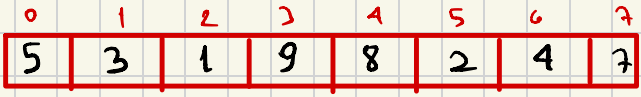


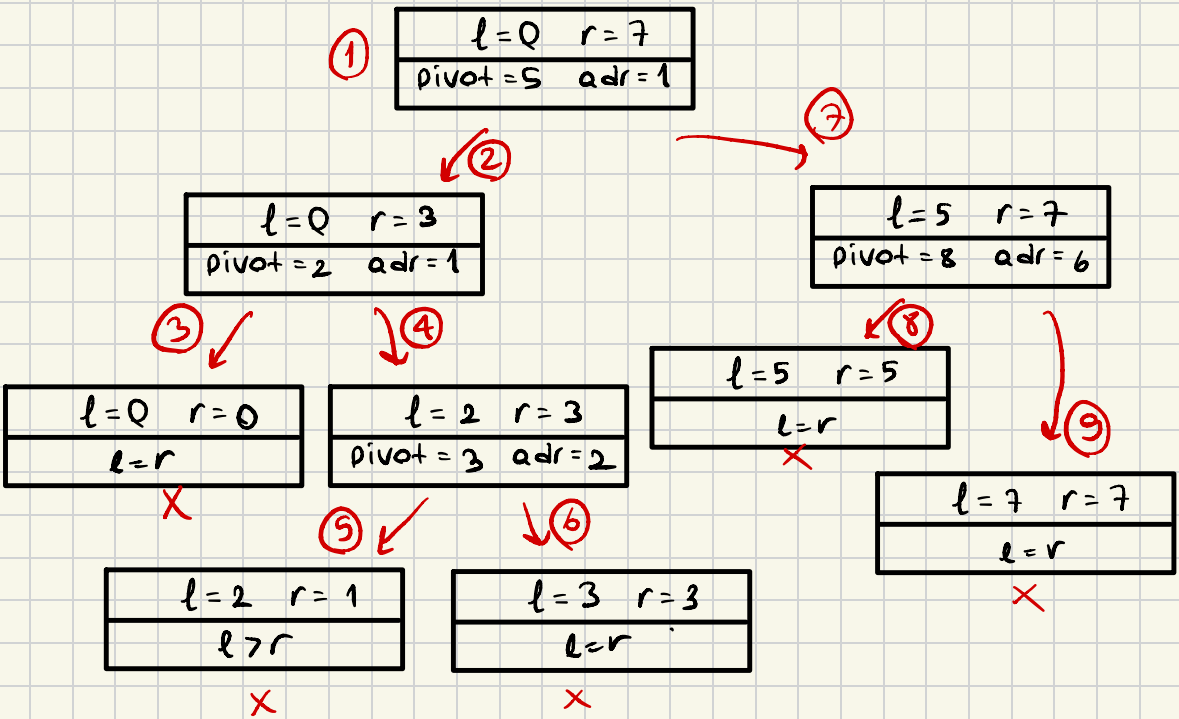
```
Quicksort (A[l...r]) {   // initial l→0  r→n-1
    if(l<r) {
        ↳durma şartı
        adr = partition (A[l...r]);
            ↳pivot elemanın yeri
        Quicksort(A[l... adr-1]);

        Quicksort(A[adr+1 ... r]);
    }
}
```

partition pivot elemanın yerini bulur, kendinden küçüklerin ve büyüklerin yerini ayarlayarak pivotu doğru yere yerleştirir.



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


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


1
l=0 r=7
pivot=5 adr=1
2
l=0 r=3
pivot=2 adr=1
7
l=5 r=7
pivot=8 adr=6
3
l=0 r=0
l=r
4
l=2 r=3
pivot=3 adr=2
8
l=5 r=5
l=r
9
l=7 r=7
l=r
5
l=2 r=1
l>r
6
l=3 r=3
l=r

Best case : Pivotun sağında ve solunda eşit eleman oluyordur.

$$C_{best}(n) = \begin{cases} 2 \cdot C(n/2) + n & , \ n > 1 \\ C(1) = 1 & , \ n = 1 \end{cases}$$

↳ partition

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

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

$$= 4[2C(n/8) + n/4] + 2n = 8C(n/8) + 3n$$

$$\vdots$$

$$= 2^i \cdot C(n/2^i) + i \cdot n \qquad n = 2^i \text{ için } i = \log_2 n$$

$$C(n) = n \cdot C(1) + n \cdot \log_2 n$$

$$= n + n \cdot \log_2 n$$

$$C_{best}(n) \in O(n \log_2 n)$$

* n eğer 2'nin kuvvetiyse hep ortadan böler, değilse orantısız olur iki taraf. Bunu kesin olarak bilemediğimiz için $\Theta(n \log n)$ diyemeyiz.

Worst case: Dizi sıralı gelmişse tüm elemanlar sağda kalır.

$$C_{worst}(n) = \begin{cases} C(n-1) + n - 1 & , n > 1 \\ C(1) = 1 & , n = 1 \end{cases}$$

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

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

$$= \sum_{i=1}^{n} i = \frac{n(n-1)}{2} \in O(n^2) \text{ veya } \Theta(n^2)$$

Average Case:

$$C(n) = c \cdot n + \frac{1}{N} \sum_{1 \le k \le n} [C(k-1) + C(n-k)]$$

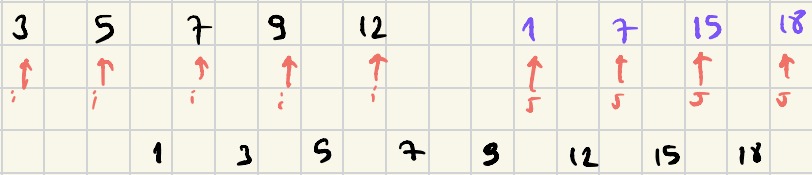
(partition)

$$\Rightarrow C(n) = 2n \cdot \ln n = 1.39\, N \log_2 N$$

Worst case olusmamasi için

1. Shuffle the array. → karmasikligi N
2. Pick the Pivot randomly.
3. Select median of three random elements

MERGE

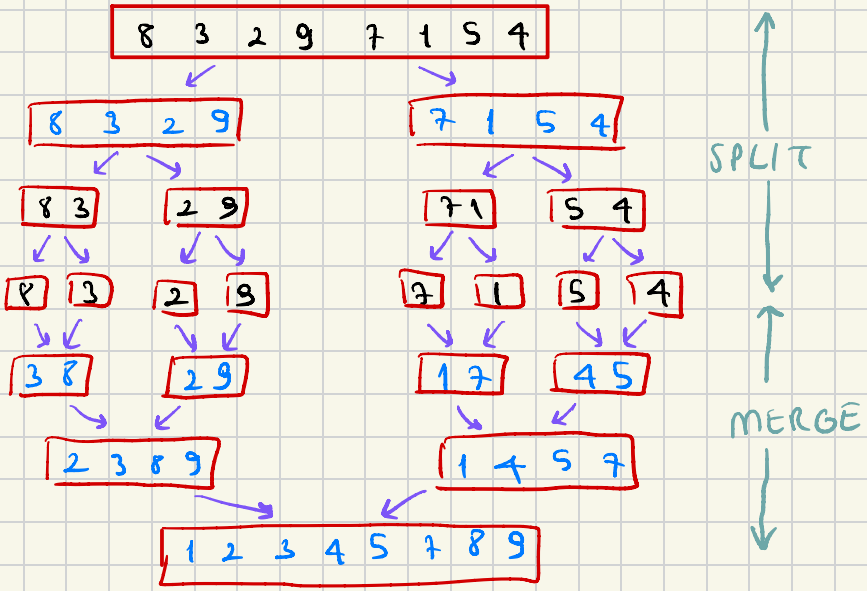


İki sıralı diziyi yeni bir diziye sıralı bir şekilde yerleştirmek istersek:

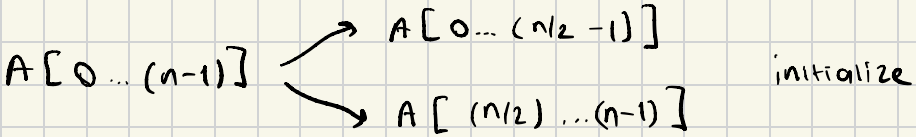
① İki dizi birleştirilip sıralanabilir. → $N \log N$

② merge ile yapılır. → $N+N \in O(N)$

MERGE SORT



Top down merge sort




```
void mergeSort (int A[], int p, int r) {
                                  // p → initial, 0
                                  // r → initial, A-1
    int q;
    if (p < r) {
        q = (p+r) / 2;
        mergeSort (A, p, q);          // sort: A[p] → A[q]
        mergeSort (A, q+1, r);        // sort: A[q+1] → A[r]
        merge (A, p, q, r);           // merge: A[p...q], A[q+1...r]
    }
}
```

A:

8	3	2	9	7	1	5	4

```
mS(A,0,7)  p=0  r=7  q=3
  mS(A,0,3)  p=0  r=3  q=1
    mS(A,0,1)  p=0  r=1  q=0
      mS(A,0,0)  p=0  r=0
      mS(A,1,1)  p=1  r=1
      merge(A,0,0,1)
    mS(A,2,3)  p=2  r=3  q=2
      mS(A,2,2)  p=2  r=2
      mS(A,3,3)  p=3  r=3
      merge(A,2,2,3)
    merge(A,0,1,3)
  mS(A,4,7)
    ⋮
  merge(A,0,1,3)
merge(A,4,5,7)
```

```
Void merge(int A, int p, int q, int r){

    B = (int*)malloc(...);
    while ((i<=q) && (j<=r)){
        if (A[i] < A[j])
            B[k++] = A[i++];
        else
            B[k++] = A[j++];
    }
```

Recurrence Relation

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

$2T(n/2)$: merge sort; $+n$: merge

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

Best case: $0,5 \cdot N \log N$

Worst case: $N \log N$

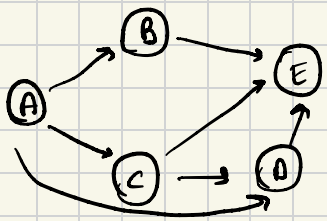
Average case: $0,7 \cdot N \log N$

Decrease And Conquer

1. Decrease by a constant :

insertion sort , topological sort

örnek : bağlı dersler



indegree	
0	A → B → C → D
1	B → E
1	C → D → E
2	D → E
3	E

2. Decrease by a constant factor :

binary search $(n \to n/2 \to n/4)$

3. Variable size decrease :

GCD

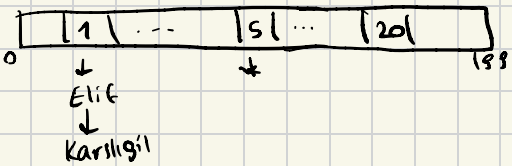
sorting	thousand	million	billion
inserting s.	instant	2.8 h	317 year
merge s.	instant	1 sec	18 min
quick s.	instant	0.6 sec	12 min

Küçük dizilerde insertion sort kullanılır.
merge → stabil quick → stabil sort değil.

HASHING

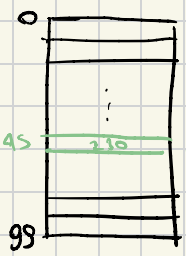
Hashing → karmaşıklığı O(1) constant

Numaraları 1 ile 200 arasında değişen öğrencilerin olduğu bir struct var. Öğrenci bilgilerine O(1) karmaşıklığında nasıl erişiriz?



Numaralar 200'e kadar değil de 1.000.000'a kadar olsaydı dizi kullanılamazdı. → Hashing bunun için var.

N → 50 kişi Numaralar → 1 ... 5000
M = 2N = 100 olsun.



230 numaralı kişiyi nereye yerleştireceğimizi bulmak için hash function kullanılır.

230 → HF → 45

170 → HF → 45 olabilir mi? ✗

① Division Method

hashfunction (key) = key % m
↳ hash table size

example) m = 100
key = 230
hashfunction(230) = 230 % 100 = 30 //

1. How to select table size?
   M > N → # of elements
2. table size is prime number.

② Multiplication Method

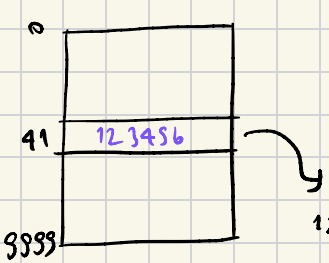
hashFunction (key) = ⌊m . [(key . A) % 1]⌋
↳ table size

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

example) m = 10000
hash (123456) = ⌊10000 . (123456 . 0,618033) % 1⌋

76300,0041151 | 1
0,0041151

= ⌊41,1511⌋ = 41



123456 $\xrightarrow{HF}$ [0 .. 9999]

* Key'ler unique olmali. → TCKN, student id, phone number

If the key is not an integer number?

① Floating point → convert to integer

35,780 → round (35,780 . 10000) % m

$0 \leq key \leq 1$ → hashfunction (round (key . m))

Java: Binary representation of the key.

② String → Java: Harver's method

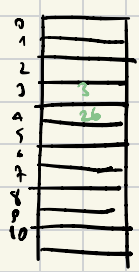
"Elif" = "E" + "l" + "i" + "f" olursa harflerin yeri degistiginde de ayni sonucu verir.

"Elif" = 'E' 'l' 'i' 'f'

R=31 $E \cdot 31^1 + l \cdot 31^2 + i \cdot 31^1 + f \cdot 31^0$

$hash = 31^{L-1} \cdot s[0] + 31 \cdot L^{-2} \cdot s[1] + \dots + 31^0 \cdot s[L-1]$

how to solve collusion?



key = 3 3 % 11 = 3

key = 14 14 % 11 = 3 ?

key = 26 26 % 11 = 4

Collusion resolution

① Open Addressing (closed hashing)

tablo uzunlugu eleman sayisindan her zaman fazladir. → genelde 2 kat.

1. Linear probing → each checking is a probe!

$h(k,i) = (h'(k) + i) \% m$ $\quad h'(k) = k \% m$

hash func, key, step

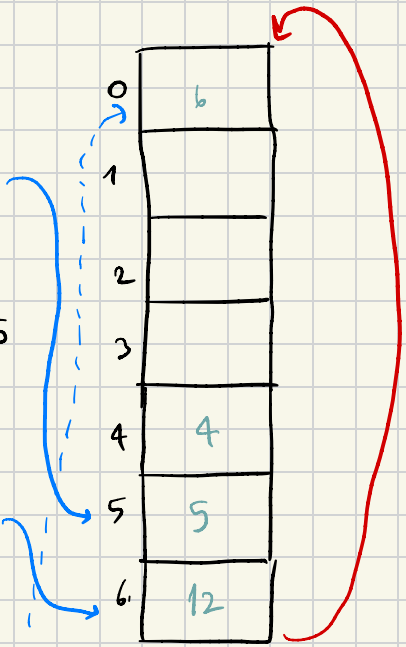
$K = 5$
$i = 0$
$h(K,0) = (5 \% 7 + 0) \% 7 = 5$

$K = 12$
$i = 0$
$h(K,0) = (12 \% 7 + 0) \% 7 = 5$

$K = 12$
$i = 1$
$h(K,1) = (12 \% 7 + 1) \% 7 = 6$

$K = 6$
$i = 0$
$h(K,0) = (6 \% 7 + 0) \% 7 = 6$

$K = 6$
$i = 1$
$h(K,1) = (6 \% 7 + 1) \% 7 = 0$



2. Quadratic Probing (non linear search)

$h(K,i) = (h'(K) + c_1 \cdot i + c_2 \cdot i^2) \% m$ → step size increase quadratically

$h(K,i) = (h'(K) + i^2) \% m$

$m = 5$ $K = 9$ $i = 0$ için:
$h(K,0) = (9 \% 5 + 0) \% 5 = 4$

$i = 4$ için
$h(K,4) = (9 \% 5 + 4) \% 5$

3. Double Hashing (use two hash functions)

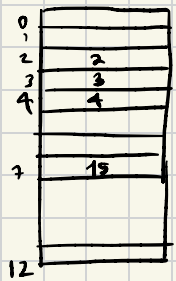
$$h(k,i) = \left(h_1(k) + i \cdot h_2(k) \right) \% m$$

$$h_1(k) = k \% m$$

$$h_2(k) = 1 + k \% mm$$

↳ $mm < m$

example



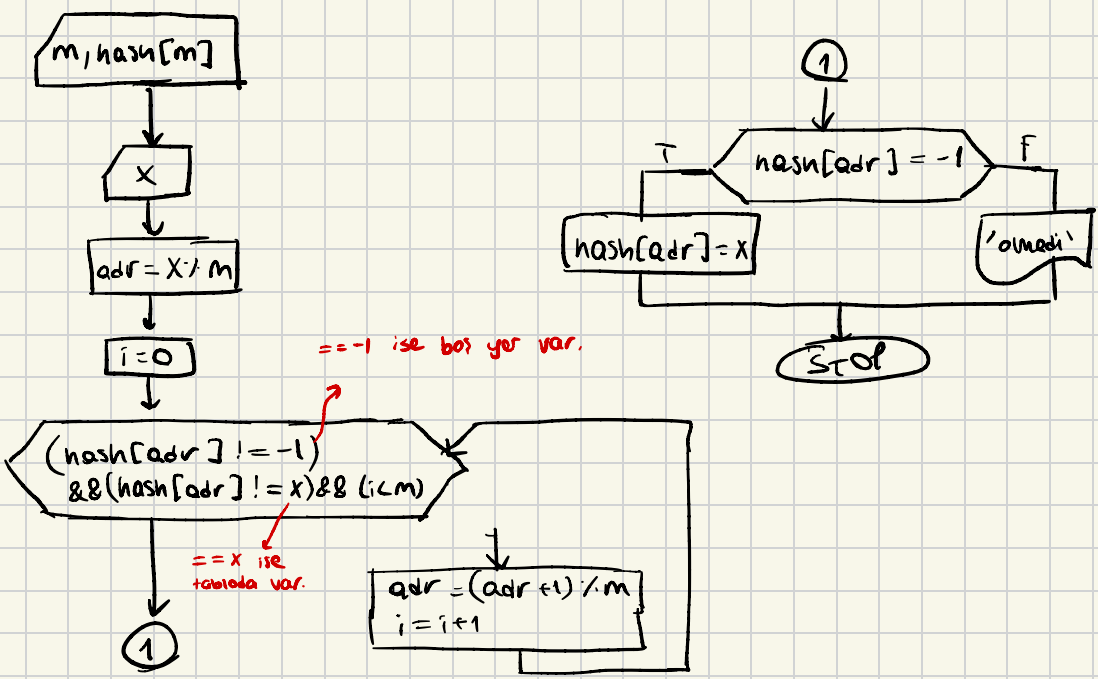
$m = 13$ $\quad h(k,0) = 15 \% 13 + 0 = 2$ dolu

$mm = 11$ $\quad h(k,1) = [15 \% 13 + \underbrace{1}_{\text{step size}} \cdot (15 \% 11 + 1)] \% 13 = 7$

$k = 15$

ex) Verilen bir x sayısını n uzunluklu bir hash tablosuna yerleştiren algoritma ?

boş indisler $\xrightarrow{-1}$ null şeklinde ifade edilebilir.

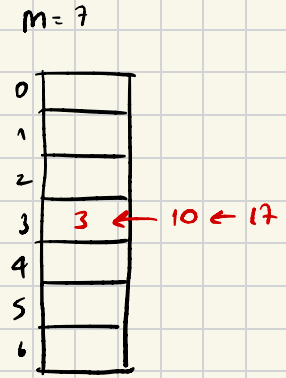


② Seperate Chaining (Open Hashing)

Linked list kullanılır.

division metot ile
$h(k) = k \% m$

$k = 3$
$h(3) = 3 \% 7 = 3$
$k = 10$
$h(10) = 10 \% 7 = 3$
$k = 17$
$h(17) = 17 \% 7 = 3$



1. insert
2. search
3. delete

```
Struct node {
   int key;
   char name[30];
   struct node *next;
}

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

Struct hash * hashTable = NULL;

int main() {
   =
   int key;
   char name[30];
   = } // Read işlemleri
```

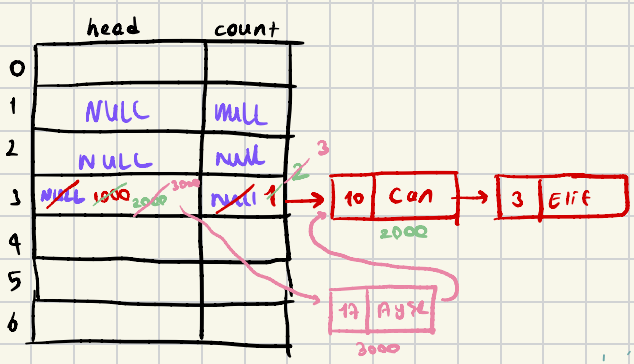
```
    hashTable = (struct hash*) calloc (m, sizeof (struct node));
    ≡} // calloc allocate edebilmiş mi kontrolü
    insertToHashTable (key, name);
    ≡
}

structure node * createNode (int key, char name[]){
    struct node * newNode = (struct node *) malloc (sizeof (struct node));
    ≡} //control
    newNode → key = key;
    strcpy (newNode → name, name);
    newNode → next = NULL;
    return newNode;
}

void insertToHash ( int key, char name[]){
    int hashIndex = key % m; → global variable
    struct node* newNode = createNode(key, name);
    if( !hashTable[hashIndex].head ){   // null mı?
        hashTable[hashIndex].head = newNode;
        hashTable[hashIndex].count = 1;
        return;
    }
    else {
        newNode → next = hashTable[hashIndex].head
        hashTable[hashIndex].head = newNode;
        hashTable[hashIndex].count ++;
        return;
    }
}
```

m = 7
key = 3
h(3) = 3

1000 | 3 | Elif

key = 10
h(10) = 3

2000 | 10 | can



Search yaparken, tabloya yerleştirilecek göz doluysa hemen aşağı bakma, o gözün içini kontrol et.

Ali
Veli
Can

Veli'yi silersek Can'a ulaşamayız.
↳ flag koyarız

	Flag
Ali	1
Veli	0 → silindi demek

linear probing için.

→ Seperate chaining'de silme problemsiz olur çünkü linked list.

Universal Hashing

$h_1()$
$h_2()$
$h_3()$
⋮
} randomly choose: $h_i()$

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

a, b, p → değişiyor.

h_3 için: $p = 17$ $m = 6$ $a = 3$ $b = 4$
↳ table size

$$h_3(key) = [(3 \cdot key + 4) \bmod 17] \bmod 6$$

Analysis of Hash Algorithms

1. Open Addressing (Linear Probing)

λ: load factor

$\lambda = \frac{N}{m}$ N: # of elements m: table size $\lambda = 0,5$

assume: data is uniformally distributed.

λ: probe. If the cell is occupied.
$1-\lambda$: not occupied

$P(\text{first probe is unoccupied}) = 1-\lambda$

$P(\text{first is occupied and second is unoccupied}) = \lambda.(1-\lambda)$

$P(\text{third is unoccupied}) = \lambda^2.(1-\lambda)$

$P(k^{th} \text{ is free}) = \lambda^{k-1}.(1-\lambda)$

the expected # of probes on insertion with linear probing $= \dfrac{1+\dfrac{1}{(1-\lambda)^2}}{2}$

$\lambda = 0,5$ ise $= 2,5$ adım

$\lambda = 0,8$ ise $= 13$ adım

λ	# of probes
0,1	1,11
0,2	1,28
0,3	1,52
0,4	1,89
0,5	2,5 → genelde bu seçilir. $M = 2N$
0,6	3,62 average case $= O(1)$
0,7	6,06
0,8	13
0,9	50,5

2. Seperate Chaining

load factor $= \dfrac{N}{M} = \lambda$

average case $= (1+\lambda) = \left(1+\dfrac{N}{M}\right)$

ideal de $M = \dfrac{N}{5}$ $\quad 2 = \dfrac{N}{M}$ da alınabilir.

* table size → prime olsa daha iyi.

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

M = table size = 8 olsa ⇒ 64, 128, 200, 400 % 8 = 0

100, 300, 500 % 8 = 4

M = table size = 11 olsa ⇒ 64, 128, 100, 500, 400, 300, 200

↓ 9 ↓ 7 ↓ 1 ↓ 5 ↓ 4 ↓ 3 ↓ 2

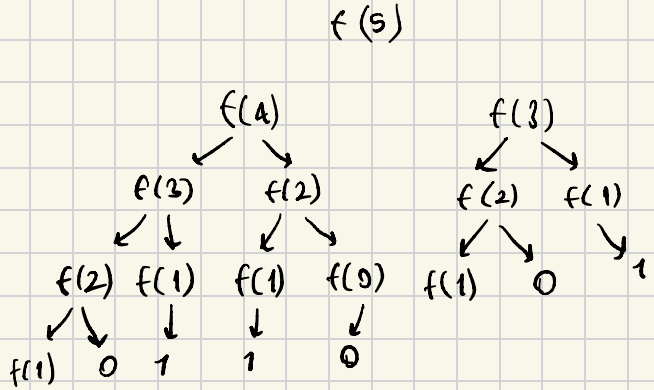
DYNAMIC PROGRAMMING

Factorial (n) = n . Factorial (n-1)
Factorial (n-1) = (n-1) . Factorial (n-2)

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

fibonacci (n) = fibonacci (n-1) + fibonacci (n-2)

0 1 1 2 3 5 8



Alt problemler bağımsız gerçekleşiyor: ⇒ dinamik Programlama

↳ select

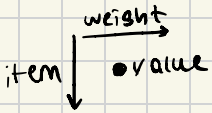
0-1 Knapsack Problem

↳ abondon

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

Brute force:

Exhaustive search → 2^n



W = 5 $\quad 12 + 10 + 15 = 37$

	0	1	2	3	4	5
0						
1						
2						
3						
4						

P[i, w] — w: weight, i: item, max profit

→ 3. item'a gelindiğinde 4 kilo için max kazanç ne olur?

kilo →

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

case 1: thief does not take item i:

$$P[i, W] = P[i-1, w]$$

case 2: thief takes item i: $\quad w_i \leq w - w_{allnon}$

$$P[i, w] = P[i-1, w - w_i] + v_i$$

$$P[i,w] = \begin{cases} P[0,j] = 0,\ P[i,0] = 0 & \text{if } i=0 \text{ OR } j=0 \quad \rightarrow \text{initial} \\ P[i-1,w] & \text{if } w_i > w \text{ not taken} \\ \max\{P[i-1, w-w_i] + v_i,\ P[i-1,w]\} & \end{cases}$$

$P[4,3] = \max\{\underbrace{P[3,1]+15}_{25},\ \underbrace{P[3,3]}_{22}\}$

$P[4,4] = \max\{\underbrace{P[3,2]+15}_{27},\ \underbrace{P[3,4]}_{30}\}$

1. define the subproblems
2. write the recurence relation
3. solve the base case
4. solve the solutions in a TABLE
   ↳ array, matrik, etc.

37'ye giden yol:

	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

0+12
12+10
12+10
12+10+15

1	2	3	4
✓	✓	X	✓

minimum coin change Problem

→ en az sayıda parayı kullanarak 11'i bul.

coin : {1, 2, 5} value = 11

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

* 4: max { 2+2 , 3+1 }
  (2+2 → 1+1 = 2 para, 3+1 → 1+1 = 2 para)
* 5: max { 4+1 , 3+2 , 5+0 }
* 6: max { 5+1 , 4+2 , 1+5 }
  (5+1 → 1+1 = 2 para, 4+2 → 2+1 = 3 para, 1+5 → 1+1 = 2 para)
* 11: (para sayıları)
  11 − 1 = 10 (2 + 1 = 3)
  11 − 2 = 9 (3 + 1 = 4)
  11 − 5 = 6 (2 + 1 = 3)
  ↙ 2 tane para ile elde edilir. ↓ STL'den gelen

$$C[money] = \begin{cases} \emptyset & \text{if } money = \emptyset \\ \min \{ C[money - d_i] + 1 \} & \text{if } money > 0 \end{cases}$$

$i : d_i \leq money$??

1. Longest Common Subsequence

Subsequence :

yakamoz

yak

az

koz

koz

⋮

yaka

~~ay~~

$X[m] = 2^m$ → anlamlı alt sıra çıkıyor mu?

yakamoz yaramaz

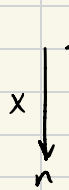
↘ yaz ↙

X = ALGORITHM Y = LOGARITH

LCS (x,y) = LORITHM

Worst case : $O(2^m * n)!$ → brute force

Dinamik Programlama ile çözülmeli.



LCS[i][j] → x[i] ve y[j]'ye gelindiğinde en uzun ortak sequence'in uzunluğu.

LCS[n][m] → LCS'in uzunluğu

Case 1: $x_i = y_j$ x = ACGT y = GTC

x_i = G, y_j = G

$LCS[i][j] = LCS[i-1][j-1] + 1$

x: <BACDB> y: <BDCB>

		0	1 B	2 D	3 C	4 B
0		0	0	0	0	0
1	B	0	1	1	1	1
2	A	0	1	1	1	1
3	C	0	1	1	2	2
4	D	0	1	2	2	2
5	B	0	1	2	2	3

B A C D B
↓↘
B D C B

(1) (B, BD) = 1
(2) (B, BDC) = 1
(B, BDCB) = 1
(BA, B) = 1

(3) (BA, BD) = 1
(4) (BA, BDC) = 1
(5) (BAC, BDCB) = 2
(6) (BACD, BD) = 2

case 2: $x_i \neq y_j$

ya da x_i is not part of CS LCS[i-1][j]
ya da y_j is not part of CS LCS[i][j-1]

initialize:

$LCS[i][0] = \emptyset$, $LCS[0][j] = \emptyset$

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

if $i,j > 0$

Karmaşıklık: $O(m \times n)$ hem yer hem zaman

		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

geri dönerken ne yaparız?

```
LCS( x[1-n], y[1-m] ) {
  for (i = 0 to n) {
    c[i][0] = 0, label[i][0] = SKIPX;
  }
  for (j = 0 to m) {
    c[0][j] = 0, label[0][j] = SKIPY;
  }
  for (i = 0 to n) {
    for (j = 0 to m) {
      if (x[i] == y[j]) {
        c[i][j] = c[i-1][j-1] + 1;
        label[i][j] = ADDXY
      }
      else {
        if( c[i-1][j] > c[i][j-1]) {
          c[i][j] = c[i-1][j];
          label[i][j] = SKIPX;
        }
        else {
          c[i][j] = c[i][j-1];
          label[i][j] = SKIPY;
        }
      }
    }
  }
}
```

→ geri dönme.

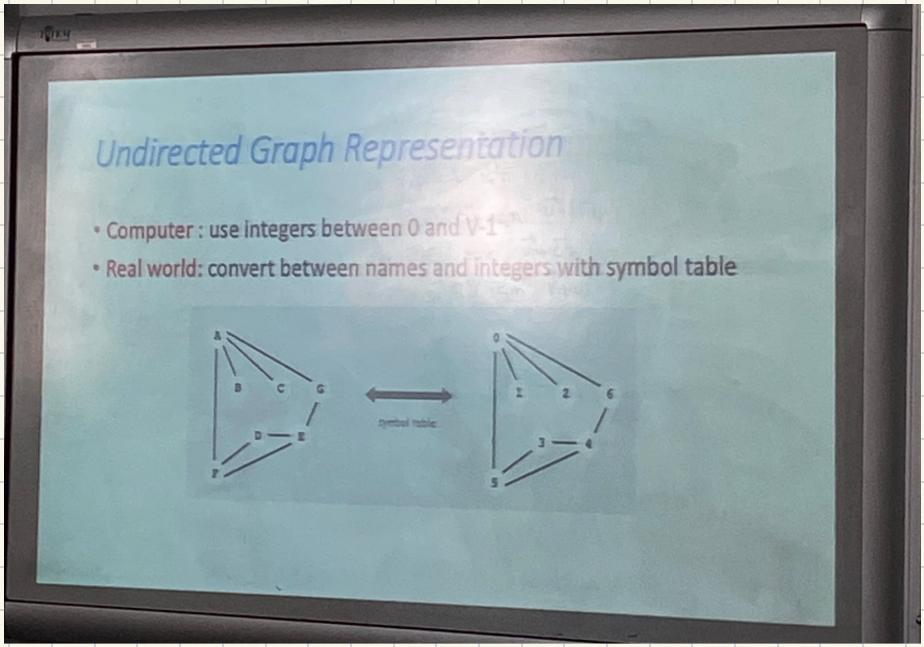
```
LCS( x[1...n], y[1...m], label[0...n][0...m]) {
  i = n; j = m;
  while (i != ∅ and j != ∅) {
    switch (label[i][j]) {
      case ADDXY:
        add x[i] to the LCS
        i--; j--;
        break;
      case SKIPX:
        i--;
        break;
      case SKIPY:
        j--;
        break.
```

Kelime Karga iken Kaga ya da Kayga yazıldığında nasıl 'Karga' önerilir?

1. $x_i = y_j$ ✓
2. $x_i \neq y_j$ → harf eklenmiş ya da silinmiş olabilir.
3. harf yanlış harfle değiştirilmiş olabilir.

Bir x kelimesinin, bir y kelimesine uzaklığı nasıl bulunur?
EDIT DISTANCE

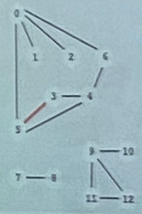
GRAPH TRAVERSAL



Undirected Graph - Adjacency Matrix

Maintain a two-dimensional V × V boolean array.

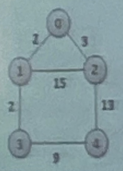
For each edge v–w in graph: adj[v][w] = adj[w][v] = true;



two entries for each edge

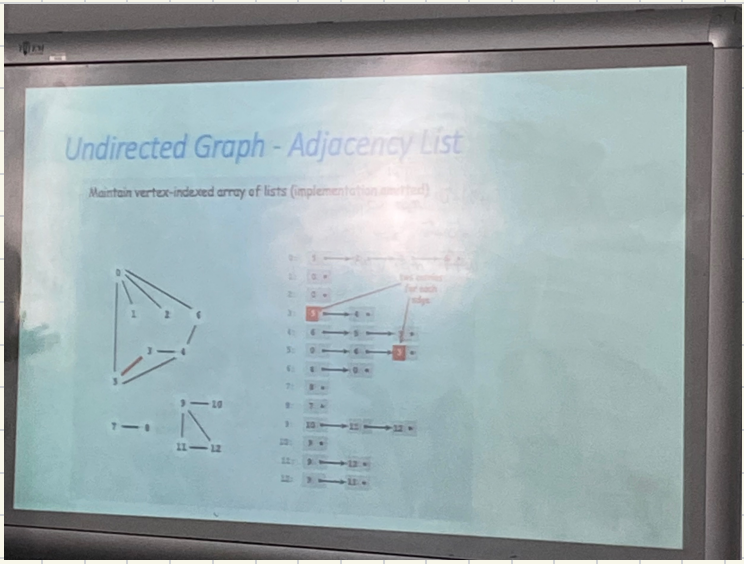
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

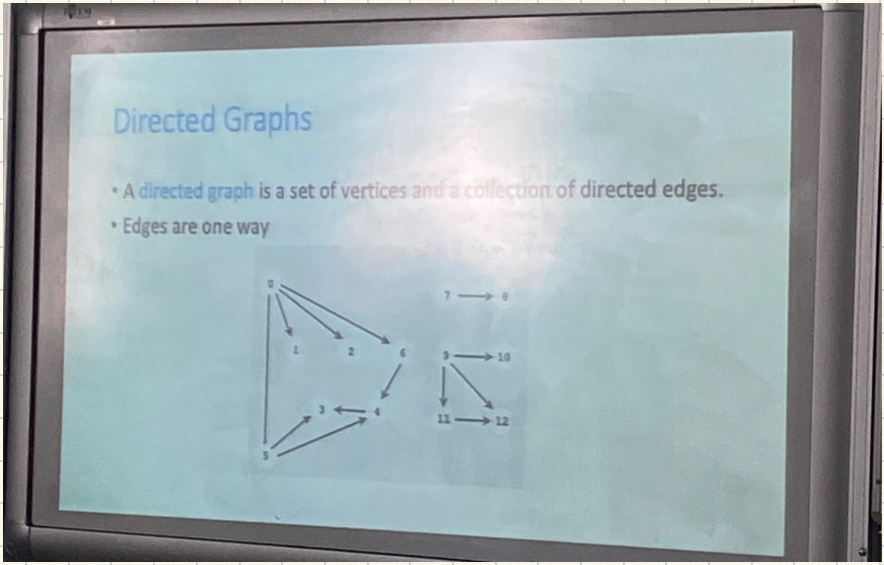
Undirected Weighted Graph - Adjacency Matrix

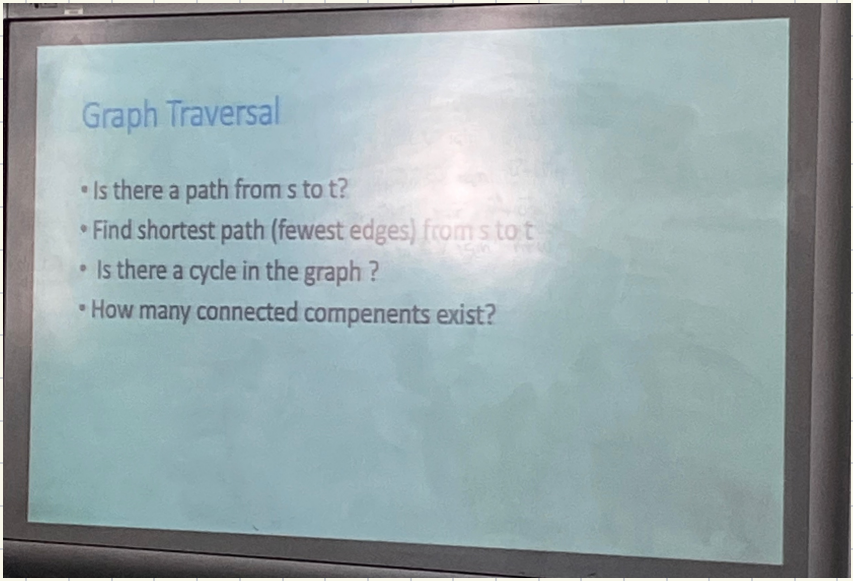


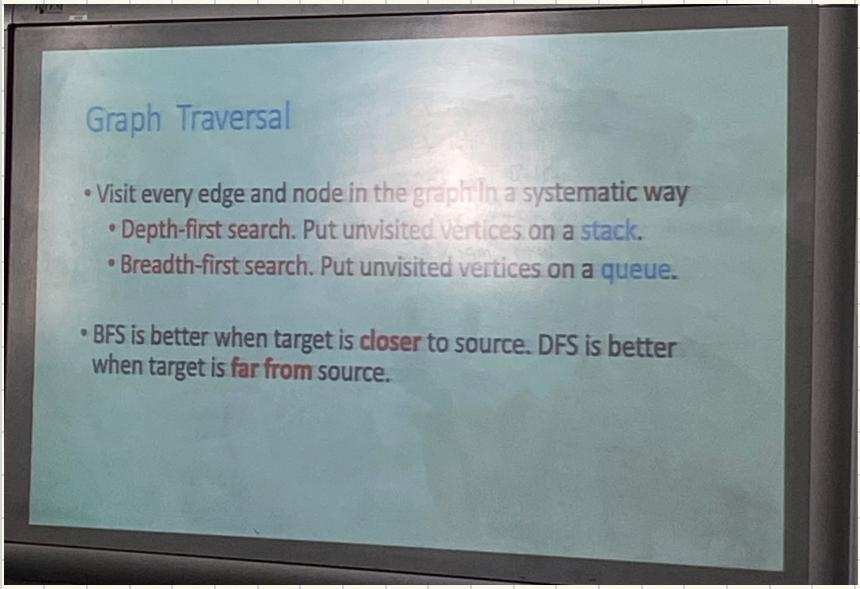
	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

Adjacency Matrix Representation of Weighted Graph


Undirected Graph - Adjacency List
Maintain vertex-indexed array of lists (implementation omitted)

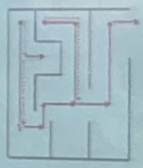

Directed Graphs
• A directed graph is a set of vertices and a collection of directed edges.
• Edges are one way


Graph Traversal
• Is there a path from s to t?
• Find shortest path (fewest edges) from s to t
• Is there a cycle in the graph ?
• How many connected compenents exist?


Graph Traversal
• Visit every edge and node in the graph in a systematic way
• Depth-first search. Put unvisited vertices on a stack.
• Breadth-first search. Put unvisited vertices on a queue.
• BFS is better when target is closer to source. DFS is better when target is far from source.

Graph Traversal – Depth First Seach (DFS)

- Like exploring a maze
- From current vertex, move to another
- Until you get stuck
- Then backtrack till you find a new place to explore

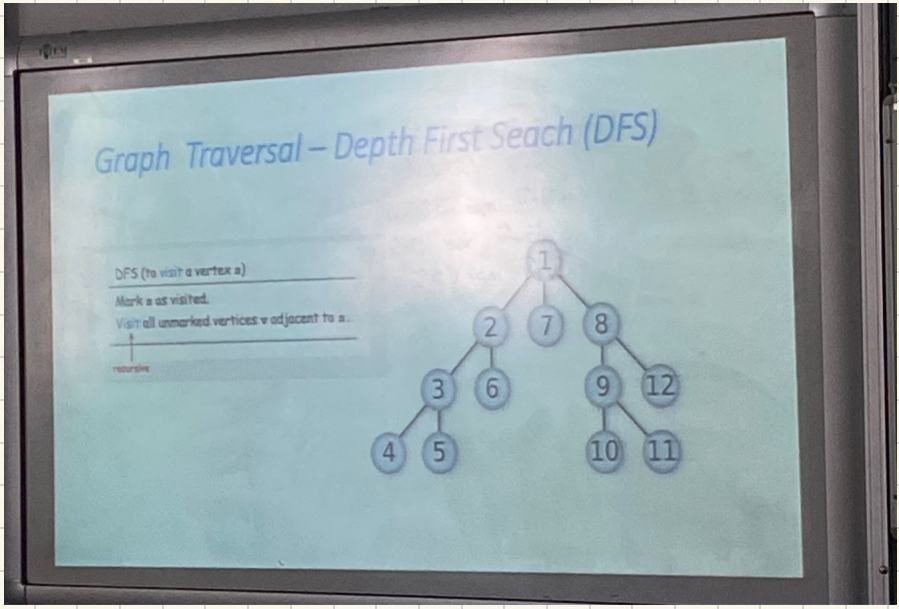


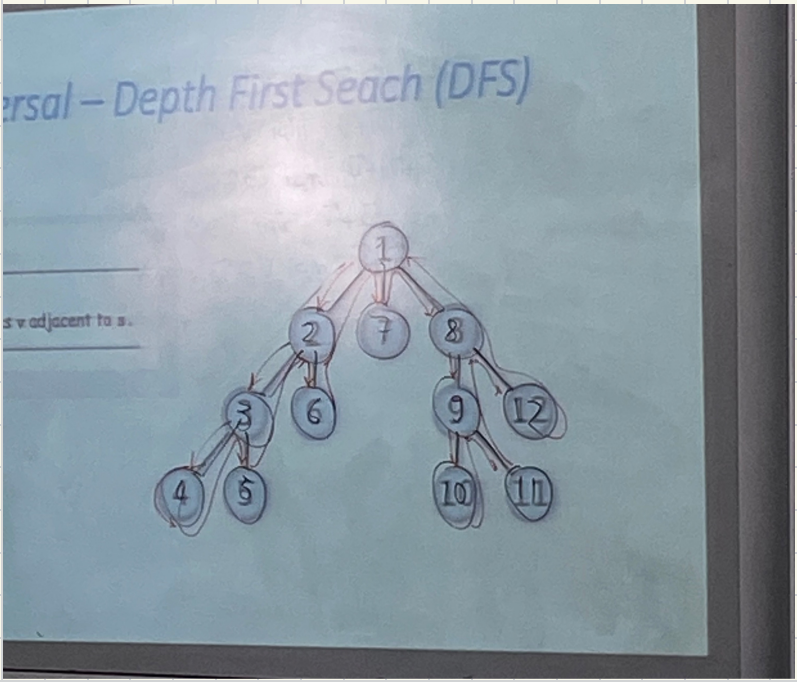
- e.g "left-hand" rule

Trémaux Maze Exploration

- Unroll a ball of string behind you.
- Mark each visited intersection by turning on a light.
- Mark each visited passage by opening a do

First use? Theseus entered labyrinth to kill the monstrous Minotaur;
Ariadne held ball of string.


Graph Traversal – Depth First Seach (DFS)
DFS (to visit a vertex a)
Mark a as visited.
Visit all unmarked vertices v adjacent to a.
recursive
1
2
7
8
3
6
9
12
4
5
10
11


ersal – Depth First Seach (DFS)
s v adjacent to a.
1
2
7
8
3
6
9
12
4
5
10
11

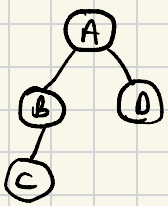
DFS: Stack Kullanır.

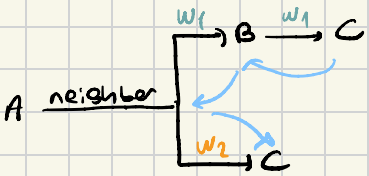
1. Recursive

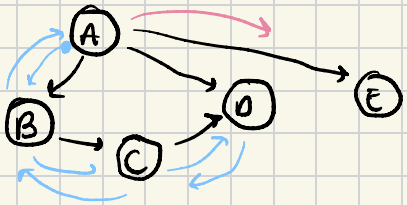
graph, source node

```
DFS( g, s ) {
    mark s as visited
    for all neighbor w of s in
    Graph g
        if w is not visited
            DFS (g, w)
}
```

	A	B	C	D
visited	0	0	0	0
	1	1	1	1





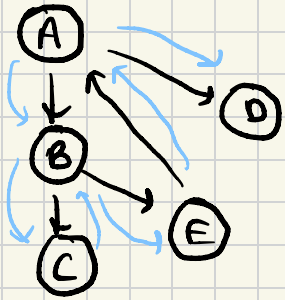


ABCDE

D'ye C'den gelir A'dan değil.

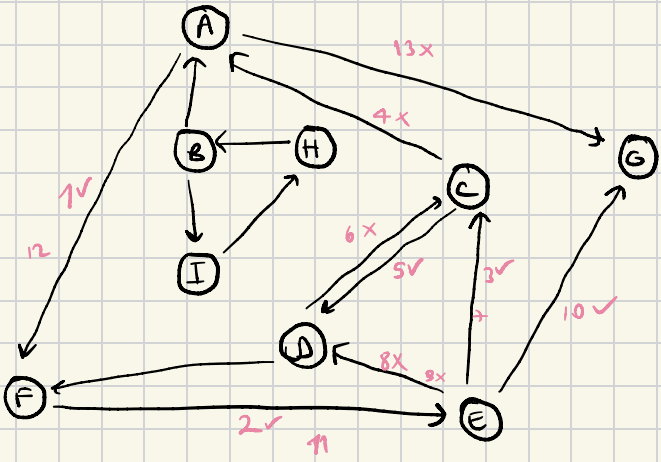
2. Non-recursive (using stack)

```
DFS( g, s) {
    stack. push(s)
    mark s as visited
    while (stack is not empty) {
        v = stack. pop( )
        for all neighbors w of v in graph g {
            if w is not visited {
                stack . push (w)
                mark w is visited
            }
        }
    }
}
```



ABCED

D'ye A'dan gelindi. E'den A'ya gittiğinde A visited olmuştu. O yüzden E'den döndü geri.

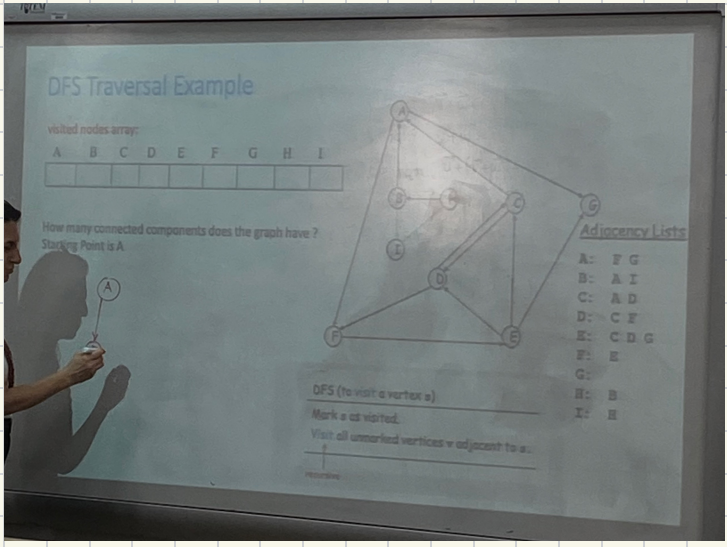


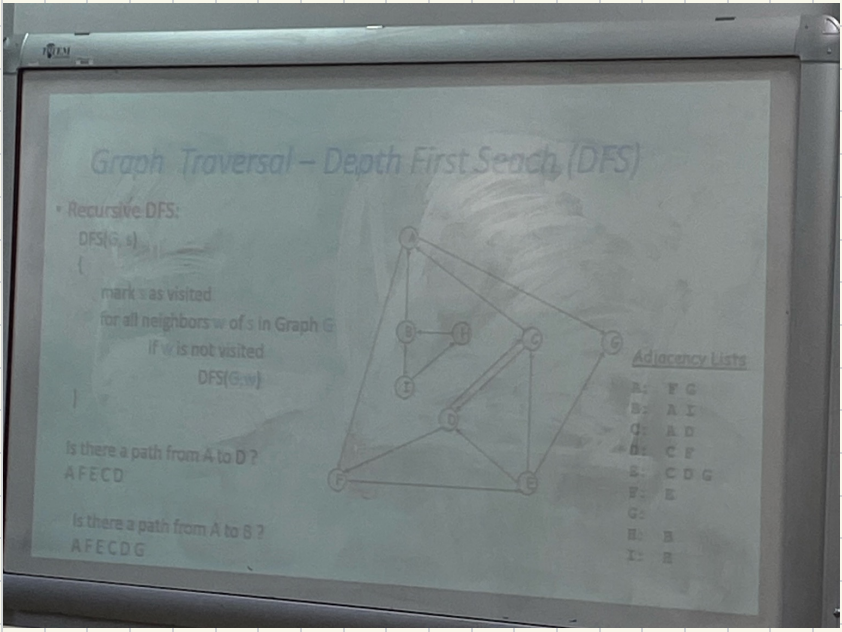
A	B	C	D	E	F	G	H	I
1	0	1	1	1	1	1	0	0

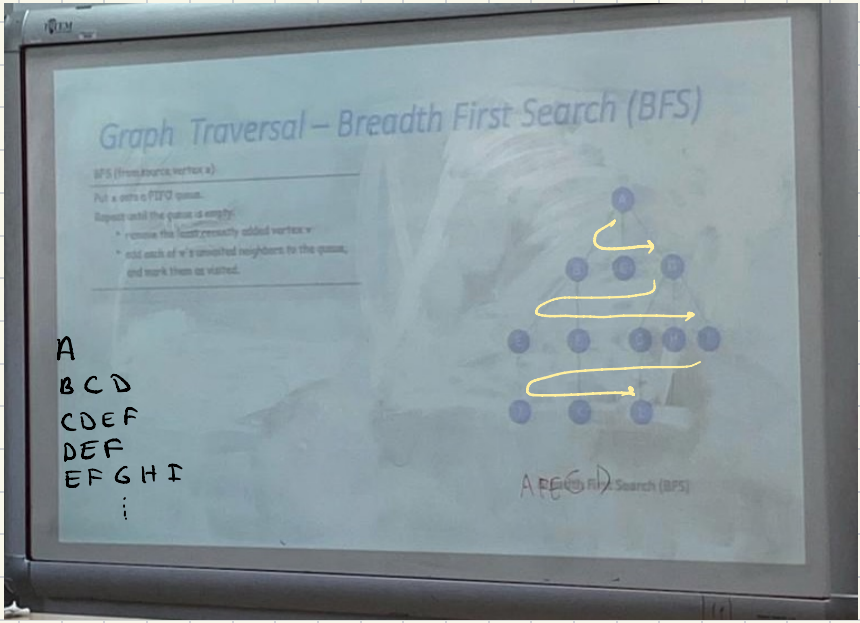
→

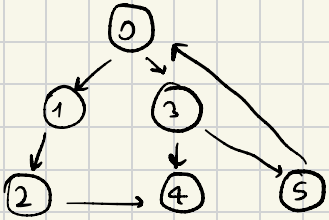
Ø olanları tekrardan geceriz

AFECDBG




Graph Traversal – Depth First Search (DFS)
Recursive DFS:
DFS(G, s)
{
mark s as visited
for all neighbors w of s in Graph G
if w is not visited
DFS(G, w)
}
Is there a path from A to D ?
A F E C D
Is there a path from A to B ?
A F E C D G
Adjacency Lists
A: F G
B: A I
C: A D
D: C E
E: C D G
F: E
G:
H: B
I: B


Graph Traversal – Breadth First Search (BFS)
BFS (from source vertex s):
Put s onto a FIFO queue.
Repeat until the queue is empty:
remove the least recently added vertex v
add each of v's unvisited neighbors to the queue,
and mark them as visited.
A
B C D
C D E F
D E F
E F G H I
⋮

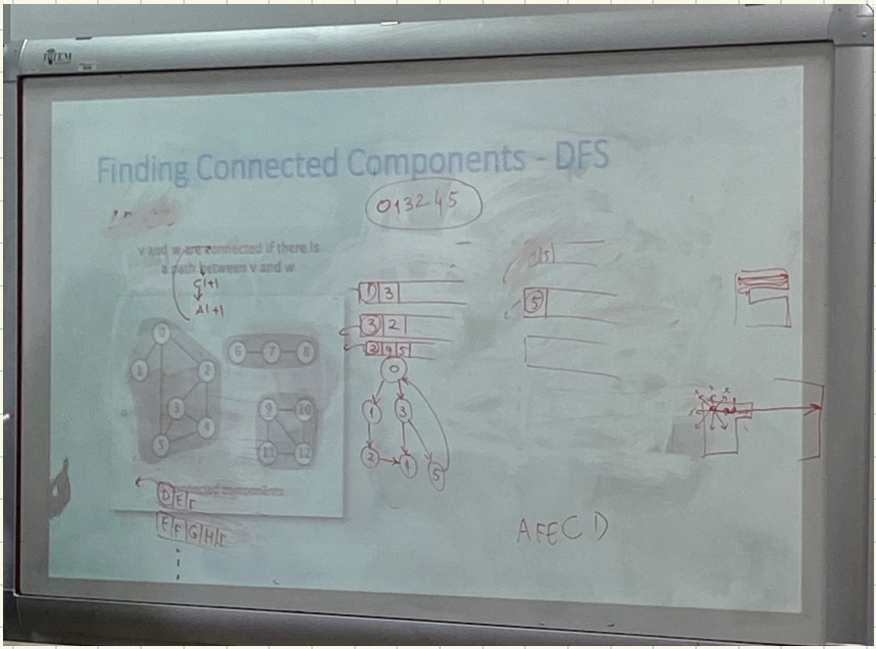


0	
1	3
3	2
2	4 5
4	5
5	

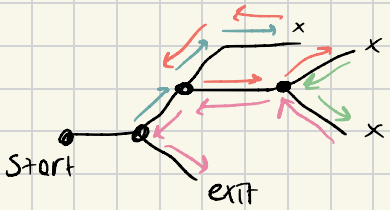
en bastakini çıkar komşularını ekle.

0-1-3-2-4-5

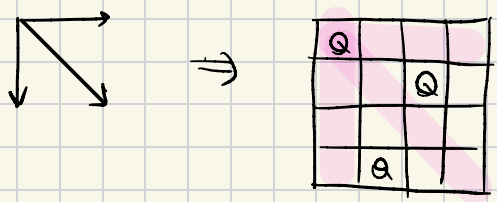
```
BFS ( g, s){
   Queue. enque(s)
   mark s as visited
   while (Queue is not empty){
      v = Queue. dequeue ( )
      for all neighbors w of v in graph g
      if w is not visited {
         Queue. enqueue( w)
         mark w as visited
      }
   }
}
```



BACKTRACKING



8 - Queen Problem

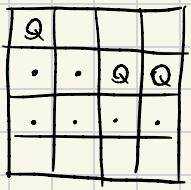


1. Brute force : $\binom{64}{8} = \frac{64!}{(64-8)!\,8!} = 4,126,165,368$ positions

2. One Per Row = $n^n = 8^8 = 16,776,216$ positions

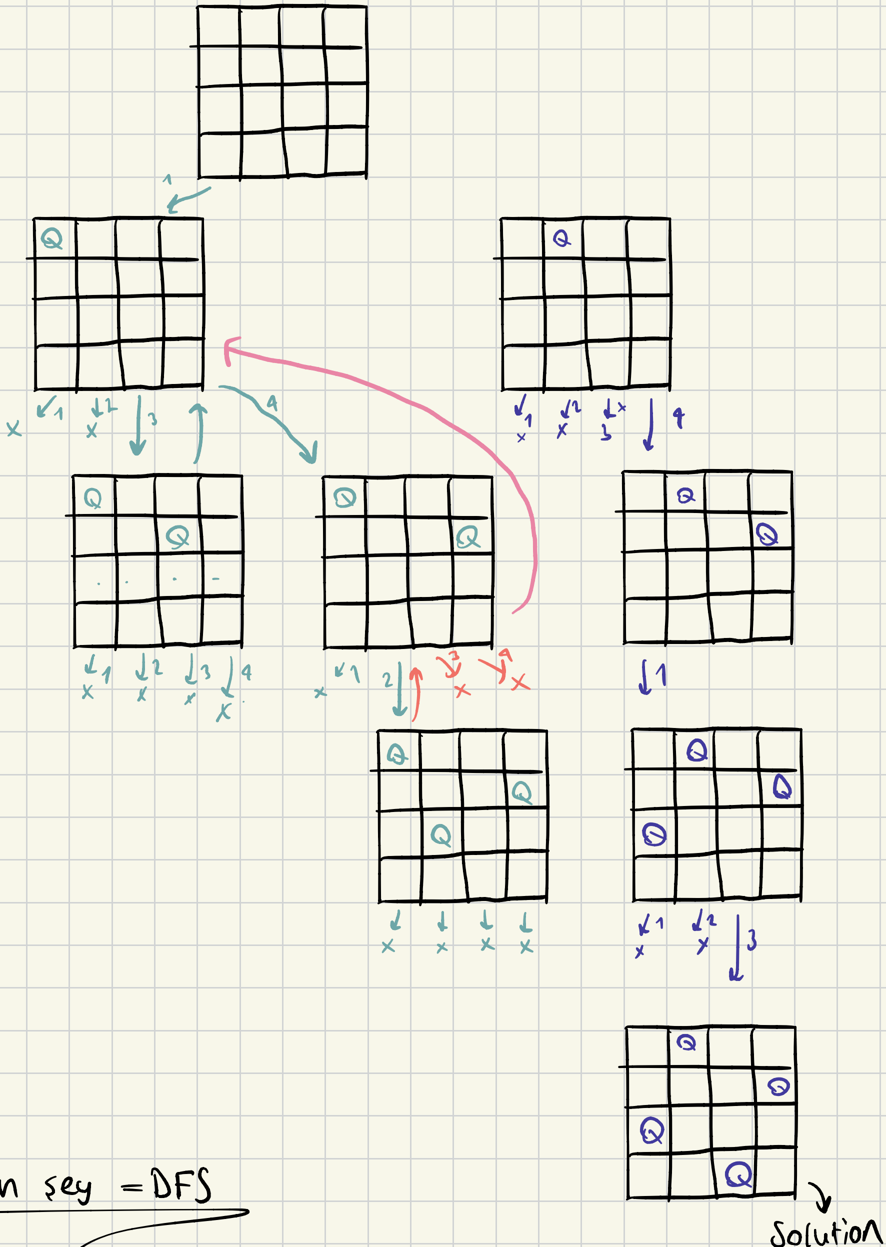
3. One per row, one per column $\Big\}$ $n! = 8! = 40,320$ positions

4. Backtracking :



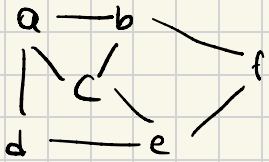
2057 positions

State Space Tree



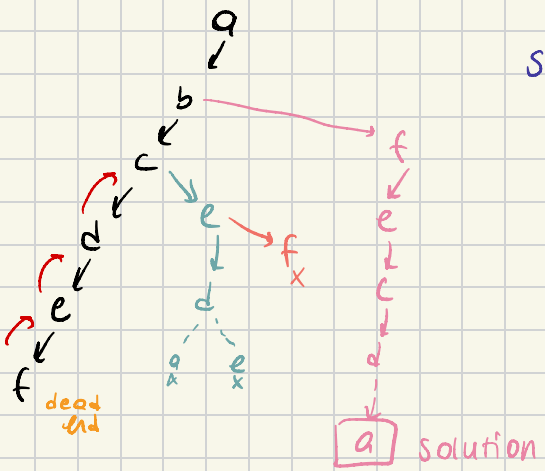
Yapılan şey = DFS

Hamilton Circuit Problem



a noktasından çıkınca her bir düğüme 1 kere uğrayarak dönülebiliyorsa Hamilton circuit

Graph (V, E) = visit every node exactly one.



State space tree for hamilton circuit

Branch And Bound

1) Assignment Problem

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

(state space tree)

Start → (çakışmalar önemsiz)

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

a → 1
$LB = 9+3+1+4 = 17$

a → 2
$LB = 2+3+1+4 = 10$

a → 3
$LB = 7+4+5+4 = 20$

a → 4
$LB = 8+3+1+6 = 18$

b → 1
$LB = 2+6+1+4 = 13$

b → 3
$LB = 14$

b → 4
$LB = 17$

c → 3
d → 4
cost = 13

c → 4
d → 3
cost = 25

a → 2
b → 1
c → 3
d → 4

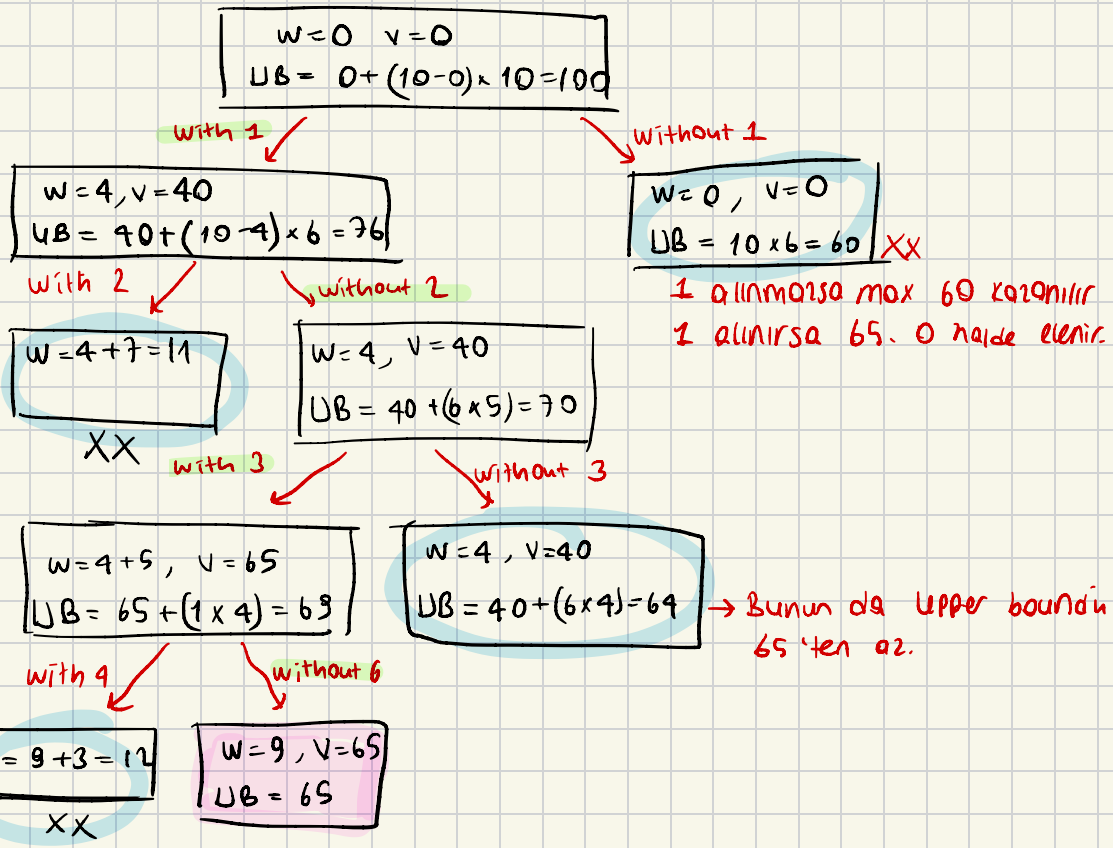
Knapsack Problem

$W = 18$

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

$v_1/w_1 \geq v_2/w_2 \geq \dots \geq v_n/w_n$

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


W=0 V=0
UB = 0+(10-0)x10=100
With 1
Without 1
W=4, V=40
UB = 40+(10-4)x6=76
W=0, V=0
UB = 10x6=60
XX
1 alınmazsa max 60 kazanılır
1 alınırsa 65. O halde elenir.
With 2
Without 2
W=4+7=11
XX
W=4, V=40
UB = 40+(6x5)=70
With 3
Without 3
W=4+5, V=65
UB = 65+(1x4)=69
W=4, V=40
UB = 40+(6x4)=64
→ Bunun da upper bound'u 65'ten az.
With 4
Without 6
W=9+3=12
XX
W=9, V=65
UB = 65

P/NP

P → polynomial NP → non deterministic polynomial

$O(n)$, $O(n^2)$, $O(n^3)$, $O(n \log n)$ → polynomial (efektif)

merge sort, huffman coding, binary search

↙

efektif algoritmalar

Inefficient algorithms → $O(n!)$, $O(2^n)$, $O(3^n)$

traveling salesman, hamiltonian circuit

	10	20	50
n	0.00001 s	0.00002 s	0.00005 s
n^2	0.001 s	0.004 s	0.0025 s
n^3	0.001 s	0,008 s	0.125 s
2^n	0,001 s	1.0 s	35.7 years

Polinomsal zamanda çözülebiliyorsa tractable

" " değilse intractable, unsolvable örn. traveling salesmon $O(n^2 \cdot 2^n)$

$n^{10^{9999}}$ → intractable

intractable problems → NP

What is an efficient algorithm?

- Algorithm Efficiency:
  An algorithm is **efficient** iff it runs in **polynomial time** on a serial computer.
- Runtimes of **efficient** algorithms:
  $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$, $O(n^{10,000,000})$
- Runtimes of **inefficient** algorithms:
  $O(2^n)$, $O(n^n)$, $O(n!)$

M. Elif Karsligil, Yildiz Technical University

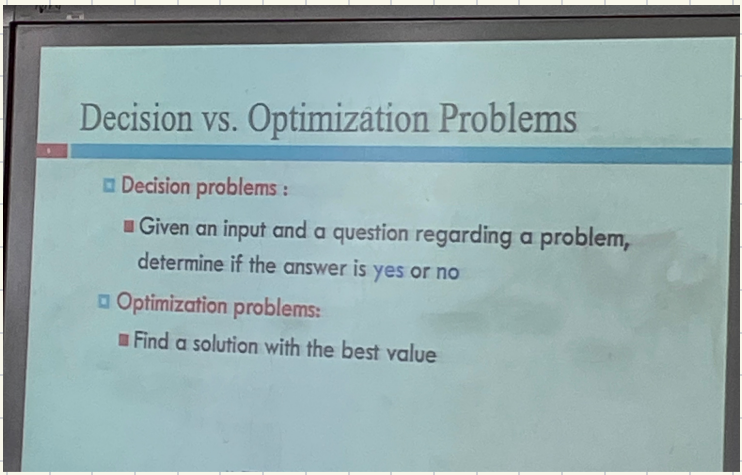
Tractable vs. Intractable Problems

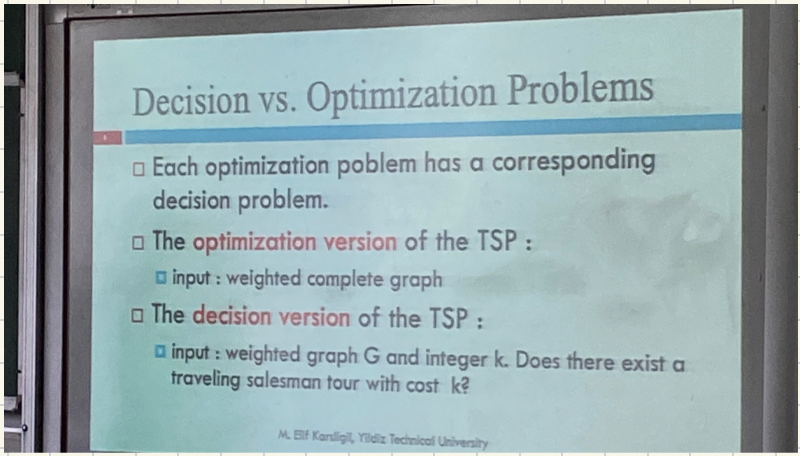
- Problems in P are called **tractable :**
  - Tractable algorithms run in polynomial time
  - Most searching and sorting algorithms $O(n^2)$, $O(n^*\lg n)$
- Problems **not** in P are **intractable or unsolvable**
  - Traveling salesperson ($O(n^2 2^n)$), knapsack ($O(2^{n/2})$)
- Are non-polynomial algorithms always worst than polynomial algorithms?
  - $n^{1,000,000}$ is *technically* tractable, but really impossible
  - $n^{\lg \lg n}$ is *technically* intractable, but easy

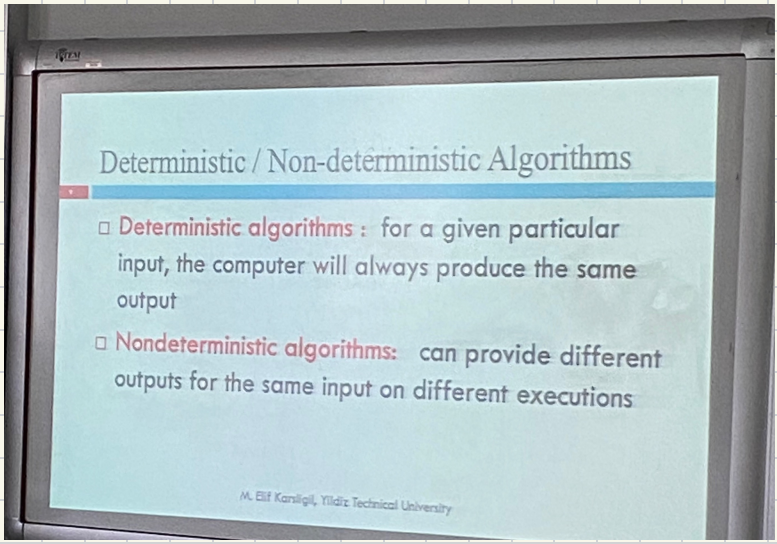
4

Intractable Problems

- Can be classified in various categories based on their degree of difficulty, e.g.,
  - NP(Non deterministic Polynomial)
  - NP-complete
  - NP-hard


Decision vs. Optimization Problems
Decision problems :
Given an input and a question regarding a problem, determine if the answer is yes or no
Optimization problems:
Find a solution with the best value


Decision vs. Optimization Problems
Each optimization poblem has a corresponding decision problem.
The optimization version of the TSP :
input : weighted complete graph
The decision version of the TSP :
input : weighted graph G and integer k. Does there exist a traveling salesman tour with cost k?
M. Elif Karsligil, Yildiz Technical University


Deterministic / Non-deterministic Algorithms
Deterministic algorithms : for a given particular input, the computer will always produce the same output
Nondeterministic algorithms: can provide different outputs for the same input on different executions
M. Elif Karsligil, Yildiz Technical University

A Searching Example

// the problem is to search for an element X in unordered array A[1..n]//
//output : j such that A[j] = x; or j = 0 if x is not in A//

```
NSearch (A,x)
j = choice (1, ..., n)        //chooses one of the element from A
if A[j] = x then
        print(j)
        success
endif
print('0')
failure
```

Complexity Ω(1) : Since A is not ordered, every deterministic search algorithm is of complexity Ω(n), whereas the non-deterministic algorithm has the complexity as O(1)

Nondeterministic decision algorithms generate a zero or 1 as their output.

Nondeterministic Algorithms

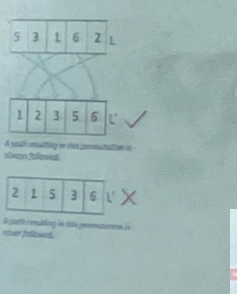
- The nondeterministic algorithm uses three basic procedures;
  - **CHOICE(1,n) or CHOICE(s)** : chooses and returns an arbitrary element, from the closed interval [1,n] or from the set s.
  - **SUCCESS** : declares a successful completion.
  - **FAILURE** : an unsuccessful termination.

Elif Karsligil, Yildiz Technical University

A Sorting Example

// sort the array A[1..n] into array B[1..n]//

```
NSort(A, B)
for i:=1 to n do
        B[i] = 0
for i:=1 to n do
        j:=choice(1... n)
        if B[j] != 0 then failure
        B[j]:=A[i]
endfor
for i:=1 to n-1 do
        if B[i]>B[i+1]then failure
endfor
print B
Success
Complexity θ(n)
```



1- guessing stage (non deter.)
2- verification stage (deter.)
3- output = failure or success

NP algoritmalar : verification aşaması polinomsal zaman olan algoritmalardır.

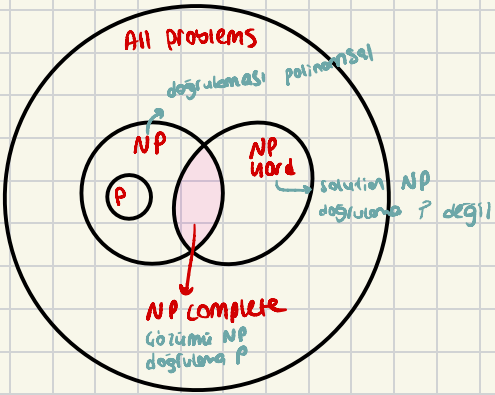
Nondeterministic Algorithms

- NP algorithms verifiable in polynomial time
- Warning: NP does not mean "non-polynomial"

1) Optimization Problems → NP
   ↳ en az kaç renge boyanır?
2) Decision Problems
   ↳ 2 renge boyanabilir mi?
   ↳ Verification ⇒ polinomsal

Class P and Class NP

- What do we mean when we say a problem is in P?
  - A solution can be found in polynomial time
- What do we mean when we say a problem is in NP?
  - A solution can be verified in polynomial time
- What is the relation between P and NP?
  - P ⊆ NP, but no one knows whether P = NP
  - A deterministic algorithm is just a special case of a nondeterministic one : P ⊆NP



	Solvable in polynomial time	Solution verifiable in polynomial time
P	✓	✓
NP		✓
NP Comp.		✓
NP hard		

NP hard: traveling salesman

NP(Nondeterministic Polynomial) Algorithms

- NP: decision problems that are verifiable in polynomial time
- Does a directed graph has a Hamiltonian Cycle : not a polynomial time
  - O(N * N!) visit all the permutations of the vertices N! iterations, in each iteration see if adjacent vertices are connected or not
  - Backtracking time complexity O(N!)
  - Dynamic Programming time complexity O(2^N)
- For a given sequence of vertices does a sequence forms a Hamiltonian Cycle : can be done in polynomial time
- Therefore Hamiltonian Cycles are in NP

P is a subset of NP

- Since it takes polynomial time to run the program, just run the program and get a solution
- But is NP a subset of P?
- No one knows if P = NP or not
- Solve for a million dollars!
  - http://www.claymath.org/millennium-problems