ALGORITMS

The Design and Analysis of Algorithms

A sequence of unambiguous instructions for solving a problem for obtaing the required output. for any legitimate input in a finite amount of time

IMPORTANT PROBLEM TYPES *

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

Understand the problem
↓
Decide on computational means
Exact approximate solution
Data structures
Algorithm design tech
↓
Design an Algorithm
↓
Prove correctness
↓
Analyze the algorithm
↓
Code the algorithm

How objects would be represented
How the operations would be implemented

Efficiency ** time space
Simplicity
Generality range of inputs, special cases
Optimality
no other algo can do better

FUNDAMENTAL DATA STRUCTURES

Linear Data S.

- Array
- Linked List
- Stack
- Queue

Operations: search, delete
insert

Implementation: static
dynamic

Non Linear Data S.

- Graphs
- Trees: connected graph without cycles

Ordered trees
Rooted trees
Binary trees

Graph representation
adjacency lists
adjacency matrix

Tree representation
as graphs
binary nodes

Set: unordered collection of distinct elements
farklı

~~Bag:~~

Operations
membership
union
intersection

Representation
Bit string
Liner structure

Bag: unordered collection elements may be repeated

Dictionary: a bag with operations search, add, delete

Chapter 2: Fundamentals of the Analysis of Algorithm Efficiency : Analysis Framework

Issues	Approaches
· Correctness	· Theoretical analysis
· Time efficiency	· Empirical (analysis) analysis
· Space efficiency	
· Optimality	

Examples

Problems	Size of input
Find x in a array	The number of the elements in the array
multiply two matrices	The dimensions of the matrices
Sort an array	⊛
Traverse a binary tree	The numbers of nodes
Solve a system of linear equations	number of equations & number of unknowns

BASIC OPERATIONS ??

* Applied to all input items in order to carry out the algorithm.

* Contribute most towards the running time of the algorithm.

Problem	operation
Find x in an array	comparison of x with an entry in the array
matrix x matrix with real entries	multiplication of two real numbers
Sort an array of numbers	Comparison of 2 array entries plus moving elements in the array
Traverse a tree	Traverse an edge

Time Efficiency: the number of repetitions of the basic operations as a function of input size

input SIZE is influenced by

* data representation eg matrix

* operations of the algorithm eg spell-checker

* properties of the objects in the problem eg checking if a given number (int) is a prime number

Units for Measuring Running Time

- Using standard time units is not appropriate
- Counting all operations in an algorithm is not

++ The approach ++

Identify and count the basic operations in an algorithm.

WORK DONE BY AN ALGORITHM

$T(n)$: running time

C_{op} : execution time for basic operation

$C(n)$: number of times basic operation is executed

$$T(n) = c_{op} \cdot C(n)$$

Types of formulas for basic operation count

* Exact formula

eg $C(n) = n.(n-1)/2$

* Formula indicating order of growth with specific multiplicative constant.

eg $C(n) \approx 0.5 n^2$

* Formula indicating order of growth with unknown multiplicative constant.

eg $C(n) \approx cn^2$

Example

Let $C(n) = 3n(n-1) \approx 3n^2$

Suppose we double the input size.

How much longer the program will run?

Specifics of the input

WORST CASE ; $W(n)$ max over inputs of size n

BEST CASE ; $B(n)$ min over inputs of size n

AVG CASE ; "average" over input of size n
$A(n)$

Average case

Expected number of basic operations repetitions considered as a random var. under some assumption about the probability distribution of all possible inputs of size n

Asymptotic Notations and Basic Efficiency Classes.

- classifying functions by their asymptotic growth

Given a particular function $g(n)$, the set of all functions can be partitioned into three set :

* Little Oh : $o(g(n))$ the set of functions $f(n)$ that grow slower than $g(n)$

* Theta : $\Theta(g(n))$ the set of functions $f(n)$ that grow at same rate as $g(n)$

* Little omega : $\omega(g(n))$ the set of functions $f(n)$ that grow faster than $g(n)$

Formal Definition of Theta

$f(n)$ and $g(n)$ have the $\boxed{\text{same rate}}$ of growth, if

$$\lim_{n\to\infty} \left(f(n)/g(n) \right) = c \quad ; \; 0 < c < \infty$$

Notation: $\boxed{f(n) = \Theta(g(n))}$

There exist constant c_1 and c_2 and a nonnegative integer n_0 such that ;

$$c_2 g(n) \leq f(n) \leq c_1 g(n) \qquad n \geq n_0$$

Little Oh

f(n) grows $\boxed{\text{slower}}$ than g(n)

(or g(n) grows faster than f(n)

$\lim(f(n) / g(n)) = 0 , n \to \infty$

Notation : $\boxed{f(n) = o(g(n))}$

There exist a constant c and a nonnegative integer n_0 such that

$f(n) < c \cdot g(n)$ for all $n > n_0$

Little Omega

f(n) grows $\boxed{\text{faster}}$ than g(n)

$\lim (f(n) / g(n)) = \infty \quad n \to \infty$

Notation : $f(n) = \omega(g(n))$

There exist a constant c and a nonnegative integer n_0 such that

$c \cdot g(n) < f(n)$ for all $n \geq n_0$

Big Oh and Big Omega

$$O(g(n)) = o(g(n) \cup \Theta g(n))$$

$$\Omega(g(n)) = \omega(g(n)) \cup \Theta(g(n))$$

$f(n) = \omega\, g(n)$ f(n) faster
$f(n) = \Theta\, g(n)$ same rate
$f(n) = o\, g(n)$ f(n) slower

$f(n) = \omega\, g(n)$, $f(n) = \Theta\, g(n) \to \boxed{f(n) = \Omega\, g(n)}$

$f(n) = \Theta\, g(n)$, $f(n) = o\, g(n) \to \boxed{f(n) = O\, g(n)}$

Little Oh fn grows slower $\lim (f(n)/g(n)) = 0 \quad n \to \infty$ $f(n) = o(g(n))$ $f(n) < c.g(n) \quad n \geq n_0$
Theta fn gn grows same rate $\lim (f(n) / g(n)) = c \quad 0 < c < \infty$ $n \to \infty$ $f(n) = \Theta(g(n))$ $c_2 g(n) \leq f(n) \leq c_1 g(n) \quad n \geq n_0$
Little Omega fn grows faster $\lim (f(n) / gn) = \infty \quad n \to \infty$ $f(n) = \omega(g(n))$ $c.g(n) < f(n) \quad n \geq n_0$

Rules of manipulate Big-OH Expressions

Let $T1(N) = O(f(N))$
$T2(N) = O(g(N))$

$T_1(N) + T_2(N) = \max(O(f(N)), O(g(N)))$

where $\max(O f(N), O g(N)) =$

$f(N)$ if $g(N) = O f(N)$

$g(N)$ if $f(N) = O g(N)$

$T_1(N) * T_2(N) = O(f(N) * g(N))$

If $T(N)$ is a polynomial of degree k,

$T(N) = \Theta(N^k) = O(N^k)$

$\log^k N = O(N)$ for any constant k.

NON RECURSIVE & RECURSIVE ALGORITHMS

Steps of math analysis of re or non re functions

1. input size
2. Basic operation
3. Worst - average - best case
4. Set up summation
5. Simplify summation

Example 1
Selection sort 1

Input: An array A[0 .. n-1]

Output: Array A[0 .. n-1] sorted in ascending order (artan sırada)

```
for i ← 0 to n-2 do
  min ← i
for j = i+1 to n-1 do
  if A[j] < A[min]
  min ← j
  Swap A[i] and A[min]
```

Cpp: (array size=10)

```
for(i=0; i<=10; i++){
    for(j=0; j<=10-i; j++){
        if(a[j]>a[j+1])
        { temp=a[j];
          a[j]=a[j+1];
          a[j+1]=temp; }
    }
}
```

Selection sort 2

Inner loop $S(i) = \sum_{j=i+1}^{n-1} 1 = (n-1)-(i+1)+1 = n-1-i$

Outher loop: $C(n) = \sum_{i=0}^{n-2} S(i) = \sum_{i=0}^{n-2} (n-1-i) = \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$

Basic formula $= \sum_{i=0}^{n} i = n(n+1)/2$

$C(n) = (n-1)(n-1) - (n-2)(n-1)/2 =$

$(n-1)[2(n-1)-(n-2)]/2 =$

$(n-1)n/2 = O(n^2)$

Important Recurrence Types

* One constant operation reduces problem size by one.

  $T(n) = T(n-1)+c$ $T(1)=d$ linear

  Solution: $T(n) = (n-1)c+d$

* A pass through input reduces problem size by one.

  $T(n) = T(n-1)+cn$ $T(1)=d$ quadratic

  Solution: $T(n) = [n(n+1)/2 - 1]c + d$

* One constant operation reduces problem size by half.

  $T(n) = T(n/2)+c$ $T(1)=d$ logarithmic

  Solution: $T(n) = c\log n + d$

* A pass through input reduces problem size by half. n logn

  $T(n) = 2T(n/2)+cn$ $T(1)=d$

  Solution: $T(n) = cn\log n + dn$

Example 1 : Factorial

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

$0! = 1$

Recurrence relation:

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

$T(1) = 1$

Telescoping

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

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

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

...

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

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

? Confused

Scanned by CamScanner

Example 2: Binary Search

Recurrence Relation

$T(n) = T(n/2) + 1 \quad , \; T(1) = 1$

Teleskoping :

$T(n/2) = T(n/4) + 1$

..

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

Add the equations and cross equal terms on opposite sides.

$T(n) = T(1) + \log(n) = O(\log(n))$

Master Theorem : A general divide and conquer recurrence

$T(n) = aT(n/b) + f(n)$

where $f(n) \in \Theta(n^k)$

$a < b^k \quad T(n) \in \Theta(n^k)$

$a = b^k \quad T(n) \in \Theta(n^k \log n)$

$a > b^k \quad T(n) \in \Theta(n$ to the power of $(\log_b a))$

Note: The same results hold with O instead of Θ

NOTLAR (Türkçe)

Algoritma Analizi ve Big-O Notasyonu

↳ Sonlu sayıda adımla bir hedefe ulaşmaya yarayan yöntem, işler listesi.

Input

Output

Definiteness Kesinlik (Algoritma adımları belirli)

Correctness Doğruluk (Bütün girdiler için tanımlı)

Finiteness Sonluluk (

Effectiveness Verimlilik

Generality Genellenebilirlik (Bir problem kümesi için)

Örn: Kümede bulunan en büyük değeri bulan pseudocode;

```
procedure max(a1, a2, .. an : integers)
max := a1
for i := 2 to n
    if max < ai then max := ai
```

Lineer Search: Array içinde baştan sona kadar tüm elemanları tek tek arar.

Örn:

```
Prosedure lineer_search(x: integer; a1...an integers)
                        aranacak eleman    int array
i := 1
while (i ≤ n and x ≠ ai)
    i := i+1
if i ≤ n then konum := i
else konum := 0
```

* Konum aranan elemanın yerini gösterir. "0" ise bulamadık

Binary Search:

Sıralı liste verilmiştir.

// Sayı oyunu, biri sayı tutuyor.

Sen yukarı aşağı diyecek

En fazla kaç adımda bulunur vb.

Bu logaritmanında formülü

1 ... 50 ... 100 oyun

yukarı 75 aşağı

* Kalan ihtimaller arasında ortaya bakmak..

* Logaritma, lineer'e göre daha iyidir.

→ Binary search'ün Pseudocodu

```
procedure binary-search(x: integer; a1..an: integers)

i:=1 (sol sınır)
j:=n (sağ sınır)

while (i<j)
begin
    m:=⌊(i+j)/2⌋        → ortadaki yer
    if x>am then i:=m+1
    else j:=m            ortadaki değer
end
if x=ai then konum:=i
else konum:=0
```

arama aralığı

Ortadaki değerden büyükse
* sol sınırı 51 yaptı.
* küçükse sağ sınırı
* 50 yapar

Karmaşıklık (Complexity)

Hafıza (space) veya zaman (Time) dan tasarruf sağlayan algoritmalar vardır.

For instance; n=10 için binary veya linear arama farkı çok değildir. Ama $n=2^{30}$ olursa aralarında yıllarca fark olabilir.

Eş: A için zaman karmaşıklığı 5000n
B için 1.1^n olsun

n=10 için A algoritması 50.000 adımda
B " 3 adımda

n=1000 için A " 5.000.000
B " $2,5.10^{41}$ ← çok fazla

iş büyüyünce olay değişiyor Dikkat!

A ve B algoritmaların karmaşıklığı

Girdi Boyutu	Algoritma A	Algoritma B
n	5000n	1.1^n
10	50 000	3
100	500 000	13,781
1000	5 000 000	$2,5 \cdot 10^{41}$
1000000	$5 \cdot 10^9$	$4.8 \cdot 10^{41392}$

Oha

Girdi üzerinden analiz yapılır.

Growth three

Bir fonksiyonun büyümesi (Growth)

* In math, büyüme hızı big-O olarak adlandırılan fonksiyon gösterir.

Tanım: f ve g fonkları reel sayılar dan reel sayılara tanımlı iki fonk

c, k sabit $O(g(x))$

$$|f(x)| \leq c|g(x)| + k$$

* $x>k$ olmak koşulu ile

Bir değerle çarpıp toplarsan

* f(x) ve g(x) fonkları hep pozitif

$$\boxed{f(x) \leq c\,g(x)+k, \quad x>k}$$

örnek:

$f(x) = x^2 + 2x + 1$ fonk'un büyme fonk x^2 (yani $O(x^2)$) old. göster.

$x > 1$ için ($x > k$)

HARİKA

$x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2$

$x^2 + 2x + 1 \leq 4x^2$

$C = 4$ $k = 1$ için

$f(x) \leq Cx^2$, $x > k$ sağlar

$f(x)$ is $O(x^2)$

En Yavaştan → Hızlıya doğru

1

$\log n$

n

$n \log n$

n^2

n^3

2^n

$n!$

Uysal Karmaşıklık:

Polinom zamanda çözülebilen algoritmalara uysal TRACTABLE ismi verilir. n^3 n^{25} ..

Polinom zamandan daha hızlı büyüyen fonksiyonlara UNTRACTABLE denir. $n!$

Örnek: Aşağıdaki algoritma ne işe yarar

```
procedure gizemli(a1, a2, a3 .. an : integers)
m := 0
for i := 1 to n-1
  for j := i+1 to n
  if |ai - aj| > m then m := |ai - aj|
```

m verilen dizideki herhangi iki sayı arası en uzak mesafeyi verir.

Karşılaştırma: $n-1 + n-2 + n-3 + \cdots + 1$

$(n-1) \cdot n / 2 = 0.5n^2 - 0.5n$

Zaman karmaşıklığı $O(n^2)$

Aynı problemi çözen farklı bi algoritma

```
procedure max_diff(a1, a2, .. an : integers)
min := a1
max := a1
for i := 2 to n
  if ai < min then min := ai
  else if ai > max then max := ai
m := max - min
```

Karşılaştırma sayıları : $2n-2$

Zaman karmaşıklığı $O(n)$

NOTE 2: KARMAŞIKLIK SINIFLARI

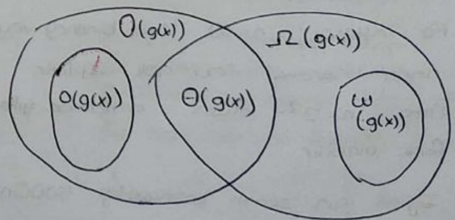
Complexity Classes

1 $\Omega(g(x))$ en iyi durum Büyük Omega

n $O(g(x))$ en kötü durum Big O

$\Theta(g(x))$ ortalama Theta

$\frac{1+n}{2}$ gibi



NOTE3

Selection Sort (Seçerek Sıralama)

Tüm listeyi geziyor, en küçük sayıyı bul. Bulunca ilk elemanla yer değiştir. 2. en küçük sayıyı bul, ikinci sıraya koy ...

$n \cdot n-1 \quad n-2 \quad \dots \quad 1$

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

$0.5n^2 + 0.5n$

Zaman karmaşıklığı $O(n^2)$ worst case

Cpp Kodu:

```
public static int[] selectionsort(int[] A, int n)
{
  int tmp;
  int min;

  for(int i=0; i<n-1; i++)
  {
    min=i;
     for(int j=i; j<n; j++){
        if(A[j] < A[min]){
           min=j;
        }
     }
    tmp=A[i];
    A[i] = A[min];
    A[min]= tmp;
  }
  return A;
}
```

Python Codu:

```
def selection_sort(mylist):
   for i in range(len(mylist)):
      minpos=i
      for j in range(i+1, len(mylist)):
         if mylist[j]< mylist[minpos]:
            minpos=j.
      temp =mylist[i]
      mylist[i] =mylist[minpos]
      mylist[minpos] = temp
```

NOTE 4

Insertion Sort (Ekleme Sıralaması)

[33,44,21,83,56,73,22]

33| 44 21 83 56 73 22

ilk sayı sıralı kabul edildi.

33 44 | 21 83 56 73 22

44 33'ten büyük mü OK kalsın

33 44 21 | 83 56 73 22

21 44'ten küçük

33 21 44 | ---

21 33'ten küçük

21 33 44 | ...

devam ...

$n.n+1/2$

$O(n^2)$ worst case

iyi ihtimal → n

Theta → $\frac{n^2+n}{2}$

Cpp Kodu:

```
void InsertionSort(int arr[], int n){
  int i, deger, j;
  for(i=1; i<n; i++){
    deger =arr[i];
    j = i-1;
    while(j>=0 && arr[j]>deger){
       arr[j+1] = arr[j];
       j =j-1; }
    arr[j+1]=deger;
  }
}
```

NOTE 5

Bubble Sort

5 7 2 9 6 1 3

ikili ikili bakma söz konusu.

5 7 5mi küçük 7mi → 5

5 7 2 7mi küçük 2mi → 2

5 2 7 6 1 3 9 → 1.adım En büyük

2 5 6 1 3 7 9 → 2.adım En büyük 2

Baloncuk şeklinde ilerleme gösteriyor.

2 5 1 3 6 7 9 → 3.adım

2 1 3 5 6 7 9 → 4.adım

1 2 3 5 6 7 9 → 5.adım

n elemanlı bir dizi için n kere tekrar edecek.

Java Kodu

```
public void bubblesort(int [] A) {
// bir diziyi parametre olarak alan metod
 int tmp;
for ( int i=0; i< A.length; i++){
 //for( int j=1 ;j<A.length-i+1 ; j++)
 //şeklinde de döngü yazılabilir.
 for( int  j=A.length-1 ; j>i ;j--){
     if ( A[j-1] > A[j] ){
         tmp = A[j-1];
         A[j-1]= A[j];
         A[j]=tmp;   }          }
     }
}
```

Worst n^2 → complexity

Best n^2

Ya sıralı bir dizi verilmişse o zaman sadece n elemanı kontrol ve eğer hiç yer değişikliği yapılmamışsa ; Best Case'i n oluyor.

Java Kodu:

```
public void bubblesort(int [] A){
// int tmp;
 for( int i=0; i<A.length; i++){
   int sirali=1;
   for (int j=A.length-1 ; j>0; j--){
     if( A[j-1] > A[j] ){
        sirali=0;
        tmp= A[j-1];
        A[j-1]= A[j];
        A[j]=tmp;
     }
   }
   if(sirali)
     break;
  }
}
```

Şayet dizinin üstünden geçtiğimiz halde hiç bir değer yer değiştirmiyorsa dizi sıralıdır. Döngüden çıkılabilir.

Best = n

Worst = n^2

T $\frac{n^2+n}{2}$

NOTE 6

Counting Sort (Sayarak sıralama)

Hafıza karmaşıklığıyla oynuyoruz.
n memoryli bi dizi tutuyor.
n memoryli bir dizi daha tutuyor

5, 7, 2, 9, 6, 1, 3, 7

* her elemandan kaç tane olduğunu sayıyor.

dizi	0	1	2	3	4	5	6	7	8	9
Değeri (sayma)	0	1	1	1	0	1	1	2	0	1

Print

0 'dan 0 tane var –
1 'den 1 tane var. + 1 yaz 1 kere
2 'den 1 tane var. + 2 yaz 1 kere
3 ' 1 "
4 ' 0 "
5 ' 1
6 ' 1 "
7 ' 2 "
8 ' 0 "
9 ' 1

↳ 12356779

Java Kodu:

```
public static void countingsort(int[]A, int[]B, int k){
  int C[] = new int[k]; // Sayma dizisi
  int i,j;

  for(i=0; i<k; i++){
     C[i]=0; }

  for(j=0; j<A.length; j++){

     C[A[j]] = C[A[j]]+1; }
}
```

$C[5] = C[5]+1 = 1$ (0+1)

Karmaşıklığı : n

n olurken 2n Big oh(2n)=n

Python Kodu:

```
# string arr[] alfabetic order'ı vercek

def countSort(arr):
   output = [0 for i in range(256)]
   count = [0 for i in range(256)]
   ans = ["" for _ in arr]
   for i in arr:
       count[ord(i)] +=1
   for i in range(256):
       count[i] +=count[i-1]
   for i in range(len(arr)):
       output[count[ord(arr[i])]-1] = arr[i]
       count[ord(arr[i])]-=1
   for i in range(len(arr)):
       ans[i]= output[i]
   return ans

arr = "hellofriend"
ans = countSort(arr)
print ("Sorted: %s" %("".join(ans)))
```

NOTE 7

Kabuk Sıralama (shell sort)

5 7 2 9 6 1 3

Kafamızdan atlama aralığı seçelim: 3

5 7 2
5 6 1
3

3 6 1
5 7 2
9

Colon

colon sırala ⟶ Tekrar oku

3 6 1 5 7 2 9

Atlama aralığını yarıya indir.

3/2 = 1

1 2 3 5 6 7 9

Kabuk sort performansı artırır.
Best, ve average case'de p. artar

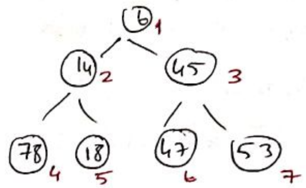
Kodu:

```
void shell_sort(int *p, int size) {
    int i, j, k, temp;
    for(k=size ; k>1 ; ) {
        k=(k<5) ? 1 : ((k*5-1)/11);
        for (i=k-1; ++i<size;) {
            temp=p[i];
            for(j=i; p[j-k]>temp;) {
                p[j]= p[j-k];
                if ((j-=k) <k)
                    break;
            }
            p[j]=temp;
        }
    }
}
```

NOTE 8

Binary Heap

Hep küçüklerin yukarıda olduğu : MIN HEAP



* Yığın Derinliği = $\log_2 N$
(n elemanlı)

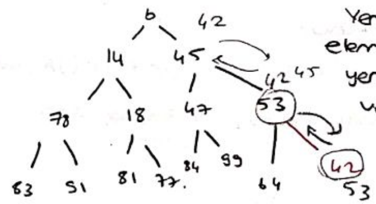
örn: N=7 $\log_2 7$ Derinlik = 2

Parent (i) = $\lfloor i/2 \rfloor$
Left (i) = $2i$
Right (i) = $2i+1$

6	14	45	78	18	47	53
1	2	3	4	5	6	7

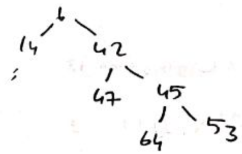
sol
sağ
sol sağ

Ekleme



Yeni gelen elemanı boş yere koy ve atasıyla karşılaştır.

53 ? 42
45 ? 42
42 ? 6

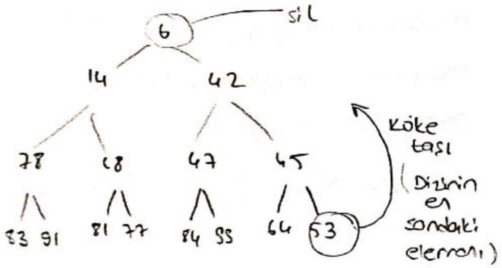


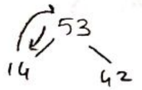
Scanned by CamScanner

* O(logN) adım 'da biter.

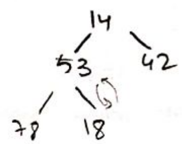
Silme işlemi

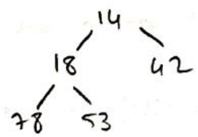
* Genelde root silinir. Neden en küçük veya en büyük silinmek istenir.
* En sağdaki elemanı köke taşı
* Tekrar heapli düzelt. (heapify)





14 'mü küçük
42 mi küçük
14





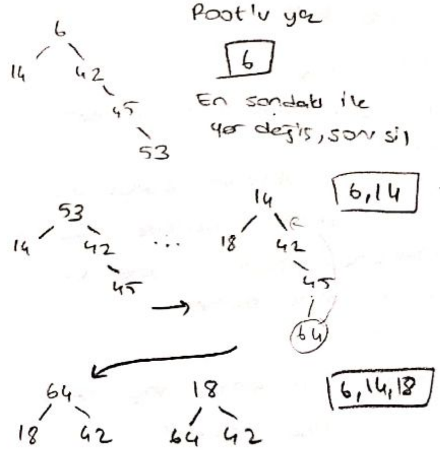
Yığın sıralı $\boxed{\text{DUR}}$

silme işleminin maliyeti: O(LogN)

YIĞIN SIRALAMASI

Heap Sort

- N elemanı yığına ekleyerek yığın oluştur.
- N adımlı silme işlemi yap
- O(N log N) sıralamadır. *
- Hafıza karmaşıklığı O(N) dir.



Bu şekilde devam ediyor. Yani sondaki elemanla sürekli yer değiştirip heapify yapıyoruz. Rootları elde ederek başka bir yerde sıralıyoruz.

* Sıralama algoritmaları arasında iyidir.
* Parçala, fethet.
* O(N) hafıza karmaşıklığı, yani N kadar eleman için yer tutuyoruz. Bir dizi.

- NOTE 9 -

Divide and Conquere

merge sort'un özelliği problemi parçalara bölmesi

Problem parçalara bölündüğünde zaman karmaşıklığı logaritmik zamana iniyor.

Divide and Conquer

Parcala & Fethet

* Recursive

Divide: Problem minik parcalara bölünür.

Conquer: Problemin özyineli olarak minik parcalar üzerinde çözümü

Combine: Problemin geneli için çözüm haline gelmesi

Divide : n elemanlı sayı dizisi n/2 bölünür

Conquer : Her iki dizinin de kendi içinde (recursive) sıralama

Birleştirme

Example: 18 26 32 6 43 15 9 1

Böldük:

* 1 6 9 15 18 26 32 43

* $\boxed{6 \mid 18 \mid 26 \mid 32}$ * $\boxed{1 \mid 9 \mid 15 \mid 43}$

$\boxed{18 \mid 26}$ $\boxed{6 \mid 32}$ $\boxed{15 \mid 43}$ $\boxed{1 \mid 9}$

18 26 32 6 43 15 9 1 ← Sırala. 1 (2'li)

* 18 mi küçük 6 mı → 6
18 mi küçük 32 mi → 18
26
32

$\boxed{6 \mid 18 \mid 26 \mid 32}$

* 15 mi küçük 1 mi → 1
* 15 mi küçük 9 mu → 9

* 6 mı 1 mi → 1
* 6 mı 9 mu → 6
9 mu 18 mi → 9
18 mi 15 mi → 15
18 mi 43 mü → 18

26 mı 43 mü → 26
32 mi 43 mü → 32

(*) Diziyi sabit tutup yerlerini ayırmak isteriz.
Diziye indisler geçir. indis tut.
Ramde bir kopyası kalsın sadece

```
mergesort(A,p,r)  // A[p..r]
if p<r
  then q ← [(p+r)/2]   // orta
       mergesort(A,p,q)   // sol
       mergesort(A,q+1,r)  // sağ
       merge(A,p,q,r)  //
       // merges A[p..q] with A[q+1,..r]
```

Çağırma mergesort(A,1,n)

Karmaşıklığı

Runnig Time T(n)

Divide: orta nokta $\Theta(1)$

Conquer: her iki alt parça sırala $2T(n/2)$

Combine: n adet eleman birleş $\Theta(n)$

Toplam zaman

$$T(n) = \Theta(1) \quad \text{if } n=1$$

$$T(n) = 2T(n/2) + \Theta(n) \quad \text{if } n>1$$

$$T(n) = \Theta(n \log n)$$

Bölünmesi n

Birleşim için her satırda n

8 elemanın kaç seviyeye indiği

log n (Her adımda n tane kaç adım log n yani n log n)

Recursive Equations

Faktöriyel: $n! = n.(n-1)!$ ← $f(1)=1$, $f(0)=1$

Fibonacci series: $f(n-1)+f(n-2)$ ($f(1)=1$, $f(0)=1$)

basis step'inde ne olacağı önemli

Example: merge sort

```
merge sort ( A, left, right){                    T(n)
  if (left < right) {                            Θ(1)
    mid = floor (( left + right) / 2 ) ;         Θ(1)
    mergeSort ( A, left, mid);                   T(n/2)
    merge Sort (A, mid+1, right);                T(n/2)
    merge ( A, left, mid, right);                Θ(n)
  }
}
```

$T(n) = \Theta(1)$, $n=1$ için

$T(n) = 2T(n/2) + f(n)$, $n>1$

Recursive Denklemlerin çözümü

1. Yerine koyma (ikame, substitution method)
2. iteration method
3. Master method

İTERATION METHOD

$$T(n) = \begin{cases} c & n=1 \\ aT\left(\frac{n}{b}\right)+cn & n>1 \end{cases}$$

a: kaç tane birleşim mevcut

b: ne kadar hızlı

$a^{\log n} = n^{\log a}$

Çözüm

k = kaç adımda basis step'e geldiğimiz

$T(n) = aT(n/b)+cn$

$T(n/b) = aT(n/b/b)+cn/b$

$T(n) = a(aT(n/b/b)+cn/b)+cn$

$a^2T(n/b^2)+acn/b+cn$

$a^2T(n/b^2)+cn(a/b+1)$

$\vdots$

$a^kT(n/b^k)+cn(a^{k-1}/b^{k-1}+a^{k-2}/b^{k-2}+\dots$
$a^2/b^2+a/b+1)$

(her defasında 2'ye bölünerek gidecek $\log_2 \text{sayı}$)

$k = \log_b n \qquad n=b^k$

$T(n) = cn(a^k/b^k+\dots+a^2/b^2+a/b+1)$

* $a=b$? olsa ne olur?

$T(n) = cn(k+1)$

$cn(\log_b n+1)$

$\Theta(n\log n)$

* $a<b$? ise

$\Sigma(x^k+x^{k-1}+\dots x+1) = (x^{k+1}-1)/(x-1)$

$$\frac{a^k}{b^k}+\frac{a^{k-1}}{b^{k-1}}+\dots+\frac{a}{b}+1 = \frac{(a/b)^{k+1}-1}{(a/b)-1} =$$

$$\frac{1-(a/b)^{k+1}}{1-(a/b)} < \frac{1}{1-a/b}$$

$T(n) = cn.\Theta(1) = \Theta(n)$

* $a>b$? ise

$$\frac{(a/b)^{k+1}-1}{(a/b)-1} = \frac{a^k}{b^k}+\frac{a^{k-1}}{b^{k-1}}+\dots+\frac{a}{b}+1 = \Theta(a^k/b^k)$$

$T(n) = cn.\Theta(a^k/b^k)$

$= cn.\Theta(a^{\log n}/b^{\log n}) = cn.\Theta(a^{\log n}/n)$

$= cn\,\Theta(n^{\log a}/n) = \Theta(cn.n^{\log a}/n)$

$\Theta(n^{\log a})$

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

Master Theorem (Divide-conquer)

Parçala-Fethet problemleri için

- Problemi a adet n/b boyutunda parçaya bölen algoritmadır.
- Her aşamanın maliyeti hesaplanır.

D-C Kaç parçaya a
her parça boyutu n/b
birleştirme .. f(n)

$$\boxed{T(n) = aT(n/b)+f(n)} \ **$$

$$T(n)=\begin{cases}\Theta(n^{\log_b a}) & f(n)=O(n^{\log_b a-\varepsilon})\\ \Theta(n^{\log_b a}\log n) & f(n)=\Theta(n\log_b a)\\ \Theta(f(n)) & f(n)=\Omega(n^{\log_b a+\varepsilon}) \text{ ve}\\ & af(n/b)<cf(n)\end{cases}$$

$\varepsilon>0$
$c<1$

Regularity condition

1. Algoritmayı recursive denkleme çevir.
2. Recursive denklemde master teorem kullanılabilir mi?
   $a>=1$? $b>1$? f(n) asimtotik mi?
   Yukarıdaki lerden uygun var mı?
3. f(n) i tanımla
   $n^{\log_b a}$ yı yaz } karşılaştır

örnek: $T(n) = \overset{a}{9}T(n/\overset{b}{3}) + \overset{f(n)}{n}$

Açıklama: 9 eşit parçaya problem bölmüşler
Her birinin boyutu n/3.

$a=9 \quad b=3 \quad f(n)=n$

$n^{\log_b a} = n^{\log_3 9} = n^2 \qquad f(n)=n$

$n^2 > n$ olduğu için $\Theta(n^2) \Leftrightarrow f(n)=n$ karşılaştır!

$n^{\log_b a-\varepsilon}$

1. kural geçerli. **

$\Theta(n^{\log_b a}) = T(n) = n^2$

$T(n) = \Theta(n^2)$

örnek 2

Merge Sort

$T(n) = \overset{a}{2}T(n/\overset{b}{2}) + \overset{f(n)}{n}$

$a=2 \quad b=2 \quad f(n)=n$

$n^{\log_b a} = n^{\log_2 2} = n = \Theta(n^1)$

$f(n)=n$

$f(n) \Leftrightarrow O(n^{\log_2 2})$
$n = n$
eşitler

2. kural geçerlidir.

$T(n) = \Theta(n^{\log_b a}\log n)$
$= \Theta(n\log n)$

örnek 3

$T(n) = 3T(n/4) + n\log n$ (a = 3, b = 4, f(n) = $n\log n$)

$a=3$
$b=4$
$f(n) = n\log n$

① dene: $n^{\log_b^a} = n^{\log_4^3}$

② f(n) ile karşılaştır

$n\log n \; ? \; n^{\log_4^3}$

$>$

③ f(n) 'in büyük olma durumu

$f(n) = \Omega(n^{\log_b^{a+\varepsilon}})$

3. durum: Regularity condition var

kontrol et: $af(n/b) < cf(n)$

$c<1$

$3f(n/4)$

$\to 3.n/4 \log n/4 < cn\log n$

$f(n) = n\log n$

$n \to n/4$ yaz

$f(n/4) = n/4 \log n/4$

sağlar Bu doğru

④ $T(n) = \Theta(f(n))$

$f(n) = \Omega(n^{\log_b^{a+\varepsilon}})$ için çıktı.

$T(n) = \Theta(f(n)) = \Theta(n\log n)$

QUICK SORT: Hızlı Sıralama

Divide and conquer

> sol	[] Pivot	> sağ

26 33 35 29 19 12 22

① Bu yedi noktadan birini pivot seç

Pivotu 26 seçtim

22 12 19 (26) 33 35 29

Sırala Sırala

...

İki pointer sol ve sağda sürekli hareket edecek

26	33	35	29	19	12	22
pivot	sol					sağ

22 ve 33 yer değiştirmeli.

26 [] 22 35 29 19 12 33

yine yer değiştirmeli

26 [] 22 12 29 19 35 33

Yer değiş

26 [] 22 12 19 29 35 33

22 12 19 (26) 29 35 33

Pivot Pivot

12 22 19 29 33 35

Eğer verilen dizi şu şekilde olsa idi.

12 22 19 26 29 33 35

Pivot, sol pointer, sağ pointer

Pivottan 22 büyük, 35 büyük

sağ bir adım ← ilerle

12 19 22 26 29 33 35

sol sağ

sağ sağ sağ sağ

ALGO

- sonlu
- Çıktı olmalı
- Belli veri kümesi (Geçerli input)
- kesin komutlar
- Problem çözümü
- Tutarlı

* Bi algoritma farklı şekillerde, yollarda ifade edilebilir.

* Bir problem farklı algo'lar kullanılarak çözülebilir.

* Doğruluğu kontrolü: Her geçerli input için sonlu zamanda doğru output'u vermesi. Doğru math formüline dayandırarak bunu ve tümevarım yöntemi ile yapılabilir.

* Algo Tasarımı: Çözüm sürecinin hesaplama modeli inşa edilir.

Algoritma Analizi

- Efficiency : time, space
- Simplicity : understable
- Generality : tüm input aralığı için çalışacak
- Optimality : en iyisi — en geniş input aralığı

GRAPHS: non linear data structures

Trees: connected graph without cycles

Rooted trees — 20

Ordered trees — 10 L, 12 R

Binary trees — ile çalışır

Set: küme, dizen gerek yok, tekrar olmaz.

Bag: Tekrar edebilir hali.

Dictionary = Key : Value , search, add, delete

---

input size'ı etkileyen hususlar

- Data representation /matrix
- Oper. of the algo /spellchecker
- Proper. of the obj /prime

$T(n) \approx c_{op}\, C(n)$

running time — execution time — kaç kere op. yapılcak

Ex $C(n) = n.(n-1)/2$

$n^2 - n/2 \rightarrow 0.5n^2 - 0.5n$

$C(n) = 0.5n^2 \rightarrow C(n) \approx cn^2$

Ex: $C(n) = 3n(n-1) \rightarrow 3n^2 - 3n$

$C(n) \approx 3n^2$

// double the input size.

$6n(2n-1) = Cn = 12n^2$

Worst	max inp
Best	min inp
Ave.	ave inp

Brute Force:

non-trivial problemler için — önemli — olası çözümleri control eder

* unacceptable in practice.

Desirable scaling properties:
when input size doubles
algo should slow down by some constant C.

→ bu varsa algoritma poly-time

* An algorithm is efficient if its running time is polynomial.

$6.02 \times 10^{23} \times N^{20}$ is tecnically poly time

but in pratice, low constant
low exponents (iş)

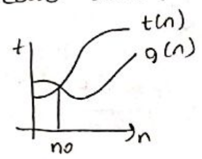
ASYMPTOTIC BOUNDS 10.s/t

O — Big O

Ω — Omega

Θ — Theta

Lower Bound (Ω) →



$t(n) \geq c.g(n)$

for every $n \geq n_0$

$t(n) = \Omega(g(n))$

Lower B → $c>0$, $n_0>0$, $n \geq n_0$ $T(n) \geq c.f(n)$ O

Upper B → $n \geq n_0$ $T(n) \geq c.f(n)$ Ω

Tight B → if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$ Θ

1

$O(g(n))$ — class of functions $f(n)$ that grow no faster than $g(n)$
$\Theta(g(n))$ — at same rate as $g(n)$
$\Omega(g(n))$ — at least as fast as $g(n)$

$f(x) = x^2 + 2x + 1$

$x > 1$

$x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2$

$\leq 4x^2$

$C = 4 \quad k = 1$

$f(x) \leq Cx^2 + k \quad , x > k$

Big O'da en küçük büyüme fonksiyonu bulunur.

Transitivity

if $f = O(g)$ $g = O(h)$

$f = O(h)$

$f = \Omega(g)$ $g = \Omega(h)$

$f = \Omega(h)$

$f = \Theta(g)$ $g = \Theta(h)$

$f = \Theta(h)$

Additivity

$f = O(h)$, $g = O(h)$ $\quad f+g = O(h)$

$f = \Omega(h)$, $g = \Omega(h)$ $\quad f+g = \Omega(h)$

$f = \Theta(h)$, $g = \Theta(h)$ $\quad f+g = \Theta(h)$

```
for(i=1; i<n; i++) { }      O(n)
```

```
for (int i=1; i<n; i+=c)
  for (int j=1; j<n; j+=c)
  {                          O(n^2)
```

$O(n^c)$ — c: içteki döngü sayısı; dıştaki dönme sayısı

Big O

$f(x) < C \cdot g(x) \qquad f(x) \in O(g(x))$

$|5x^3 - 2x^2 + 3| \leq 5x^3 + |2x^2| + 3$

$\leq 5x^3 + 2x^3 + 3x^3$

$\leq 10x^3$

$\leq 10|x^3|$

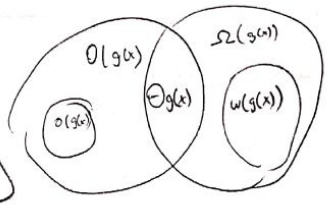
$C = 10 \qquad g(x) = x^3$

```
for(i=1; i<n; i*=c) { }      O(logn)
```

```
for( i=1 i<m i+=c)
{ }
for(i=1 i<n i+=c)             O(m)+O(n) = O(m+n)
{ }
```

```
int Topla ( int A[], int N)
{
  int topla=0              -> 1
  for (i=0 ; i<N; i++)     -> N
     topla += A[i];        -> N
  }
  return topla;            -> 1
}
```

$\boxed{2N + 2 = T(N)}$



$10n^2 - 16n + 100$ is $O(n^2)$
also $O(n^3)$

$10n^2 - 16n + 100 < 11n^2$ for all $n \geq 10$

$10n^2 - 16n + 100$ is $\Omega(n^2)$
also $\Omega(n)$

$10n^2 - 16n + 100 \geq 3n^2$ for all $n \geq 16$

$10n^2 - 16n + 100$ is $\Theta(n^2)$

is not $O(n)$
is not $\Omega(n^3$

$\overbrace{1+1+1+\cdots+1}^{n}$ $\Theta(n)$

$1+2+\cdots+n$ $n.n+1/2 = n^2/2 = \Theta(n^2)$

$1^2+2^2+\cdots+n^2$ $n.n+1 . 2n+1/6$ $n^3/3 \in \Theta(n^3)$

$1+a+\cdots+a^n$ $(a^{n+1}-1)/(a-1)$

WHY WE ARE ONY LOOK WORST CASE

* It gives us a guarantee
algo will never take any longer

*

Big-Oh Notation

$T(n) = 2n+5$ is $O(n)$

$2n+5 <= 3n$ for all $n >= 5$

$T(n) = 5n^2 + 3n + 15$ is O

$5n^2 + 3n + 15 <= 6n^2$ $n >= 6$

Ω Notation

$T(n) = f(n) >= c*(g(n))$

$T(n) = 2n+5$ is $\Omega(n)$ why?

$2n+5 >= 2n$ $n > 0$

$T(n) = 5n^2 - 3n$. $\Omega(n^2)$
$n >= 4$

$5n^2 - 3n >= 4n^2$

Θ Notation

$T(n) = 2n+5$ is $\Theta(n)$ why

$2n <= 2n+5 <= 3n$ for all $n >= 5$

$T(n) = 5n^2 - 3n$ is $\Theta(n^2)$ why

$4n^2 <= 5n^2 - 3n <= 5n^2$ for all $n >= 4$

Common Functions

Constant	$O(1)$	
Log log	$O(\log\log N)$	
Logarithmic	$O(\log N)$	
Linear	$O(N)$	
N logN	$O(N\log N)$	sorting algo
Quadratic	$O(N^2)$	
Cubic	$O(N^3)$	
Exponential	$O(2^N)$	

POLYNOMIAL TIME

Time and space Tradeoff
dengesi

- To make algo faster → you have to use more space
- use less space → algo will run slower

A SURVEY OF COMMON RUNNING TIME

Linear Time: Running time is, most constant factor times the size of input

Merge: Combine

Little Oh o

$f(n) < c g(n)$ $n \geq n_0$

↑ daha yavaş büyük olanlar

Little Omega ω

$f(n) > c g(n)$ $n > n_0$

↳ grow faster

Big Oh & Big Omega

$O(g(n)) = o(g(n)) \cup \Theta(g(n))$

$\Omega(g(n)) = \omega(g(n)) \cup \Theta(g(n))$

Scanned by CamScanner

$T_1(N) + T_2(N) = \max(O(f(N)), O(g(N)))$

$T_1(N) * T_2(N) = O(f(N) * g(N))$

ALGORITHM ANALYSIS

brute force: For many nontrivial problems, there is a natural brute-force search algo that checks every poss. p solution

$32n^2 + 17n + 1$

	Lower Bound	Tight Bound
$32n^2 + 17n^2 + n^2$	$\Omega(n^2)$	$\Theta(n^2)$
$50n^2 \leq cn^2$	$c = 32$ $n_0 = 1$	$c_1 = 32$
$c = 50$ $n_0 = 1$		$c_2 = 50$
$\boxed{O(n^2) = T(n)}$		$n_0 = 1$

With multiple Variables

$T(m,n) = 3mn^2 + 17mn + 32n^3$

$O(mn^2 + n^3)$ and $O(mn^3)$

~ EXAMPLES ~

```
for (i = 1 ; i <= n^2 ; i++)      n^2
    for (j = 0; j < i; j++)       <= n^2
        sum++;                    O(1)
```

Running time $O(n^4)$

$$\sum_{i=1}^{n^2} i = \frac{n^2(n^2+1)}{2} = \frac{n^4 + n^2}{2} \quad \Theta(n^4)$$

Building A Better Power

```
long power (long x, long n)
  if (n==0) return 1;
  if (n==1) return x;
  if ((n%2)==0)
     return power(x^2, n/2);
  else
     return power(x^2, n/2) * x;
```

$T(0) = c_1$

$T(1) = c_2$

$T(n) = T(n/2) + c_3$ (power of 2)

$T(n) = T(n/2) + c_3$

$T(n) = T(n/2) + c_3$

$T(n/2) = T(n/4) + 2c_3$

$T(n/4) = T(n/8) + 3c_3$ ← c_3 yazıyor sade

yerine koy

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

$T(n/2^k) + kc_3 =$

$n/2^k = 1$

$n = 2^k$

$\log_2 n = k$

$T(n) = T(n/2^{\log n}) + \log n \, c_3$

$= \underbrace{T(1)} + c_3 \log n$

$c_2 + c_3 \log n$

$\Theta(\log n)$

```
long power(long x, long n)
  if (n==0)
    return 1;
  else
    return x * power(x, n-1);
```

$T(0) = c_1$

$T(n) = c_2 + T(n-1)$

we should know $T(n-1)$

$T(n-1) = c_2 + c_2 + T(n-2)$

$T(n-1) = 2c_2 + T(n-2)$

$T(n-2) = \underbrace{c_2 + c_2}_{3c_2} + T(n-3)$

$3c_2 + T(n-3)$

$T(n-1) = T(n-2) + c_2$

$T(n-2) = T(n-3) + c_2$

$T(n-3) = T(n-4) + c_2$

$T(n-k) + kc_2$

$T(0) = c_1$

$T(n) = T(n-k) + k * c_2$

if $k = n$

$T(n) = T(0) + nc_2$

$c_1 + nc_2$

$\in \Theta(n)$

Not: $T(n)$ size n olan recursive case'i ifade eder