Unit 1 2 3 - notes

Design and analysis of algorithm (Lovely Professional University)

1. Fundamentals of Algorithms

Algorithm: A sequence of unambiguous instructions for solving a problem i.e, For obtaining a required output for any legitimate input in a finite amount of time.

problem
↓
algorithm
↓
input → computer → output

Requirements of Algorithm:

1. Finiteness
2. Definiteness
3. clearly specified inputs
4. " expected output
5. Effectiveness.

Analysis: 1. Theoretical Analysis 2. Empirical Analysis

Algorithm Design Techniques:

1. Implementation Method – Recursion (or) iteration
   Ex: Tower of Hanoi
2. Design Method
   Greedy Method
   Ex: Factorial knapsack
   Activity Selection.
   Divide & conquer method
   Merge Sort
   Quick sort
3. Design Approaches.
   Ex: NP-Hard Problems, Sorting Algo's
   Approx Algo
   Exact Algo
4. Other classifications.
   Dynamic Programming
   - 0-1 knapsack
   - Subset-sum problem

   Linear problem
   - max flow of Directed graph

   Backtracking
   - N-queen problem
   - Maze problem.

Asymptotic Notations

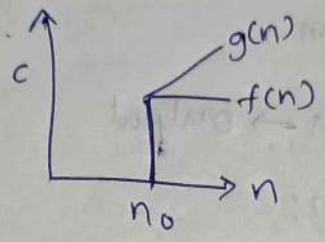
Big Oh (O) | Big omega (Ω) | Big Theta (Θ)

Big oh (O)

A B
f(n) g(n)

$f(n) = O(g(n))$

if $f(n) \leq c \cdot g(n)$



Eg: ① $f(n) = n+10$, $g(n) = n$

$f(n) = O(g(n))$

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

$n + 10 \leq c \cdot n$

$10 + 10 \leq c \cdot 10$

$n = 10, c = 2$

② $f(n) = n^2$ $g(n) = n$

$f(n) = O \cdot g(n)$

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

$n^2 \leq c \cdot n$

$1 \leq c \cdot 1$

$n = 1, c = 1$

```
→ for (i=0; i<n; i++)
  {
      j = 2;
      while (j<n)
      { j = j^2;
      }
  }
```

~~Complexity = nlogn~~

Complexity = $n \log(\log n)$

$2^{2^k} = n \Rightarrow 2^k = \log n$ (applied log on B.S)

$K = \log(\log n)$ (again applied log on B.S)

```
→ for (i=0; i<n; i++)
  { Cout "Hello"   O(n)
  }
→ for (i=0; i<n; i++)
  {   i = 2;
      while (j<n)
      { j = 2 x i;
      }
  }
```

Complexity = nlogn.

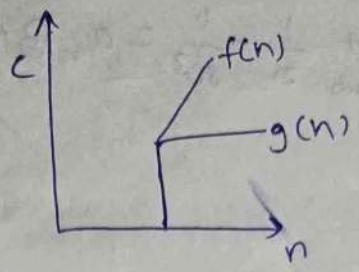
Big omega (Ω)

A B

$f(n)$ $g(n)$

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

if $f(n) \geq c \cdot g(n)$



Q) $f(n) = n$, $g(n) = 7n$

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

$f(n) \geq c \cdot g(n)$

$n \geq 1 \cdot 7 \cdot n$

$n = 1$ & $c = 1/7$

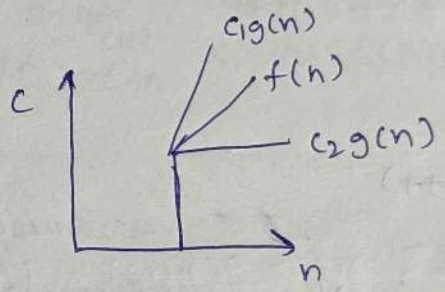
Big Theta (Θ)

A B

$f(n)$ $g(n)$

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

$f(n) \geq c_2 g(n)$



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

Limits L (L) — zero 0 (oh) — constant c (Θ) — infinity ∞ (Ω)

Q) $f(n) = n^2 + n + 1$ $g(n) = n^2$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = L \Rightarrow \lim_{n\to\infty} \frac{n^2+n+1}{n^2} = \lim_{n\to\infty} \frac{n^2}{n^2} + \frac{n}{n^2} + \frac{1}{n^2} = 1+0+0 = 1$ Big Θ

Q) $f(n) = 2n^4 - n^2 + 4$, $g(n) = n^4$

$L = \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2n^4 - n^2 + 4}{n^4} = \lim_{n\to\infty} 2 - \frac{1}{n^2} + \frac{4}{n^4} = 2$ Big Θ

Q) $f(n) = 2n^5 + n^2 - 4$, $g(n) = 2n^3$

$L = \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2n^5 + n^2 - 4}{2n^3} = \lim_{n\to\infty} n^2 + \frac{1}{2n} - 4 = \infty + 0 + 0 = \infty$ Big Ω

Q) $f(n) = n^2 - 4$, $g(n) = n^3$

$= \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2-4}{n^3} = \lim_{n\to\infty} \frac{1}{n} - \frac{4}{n^3} = 0$ Big oh.

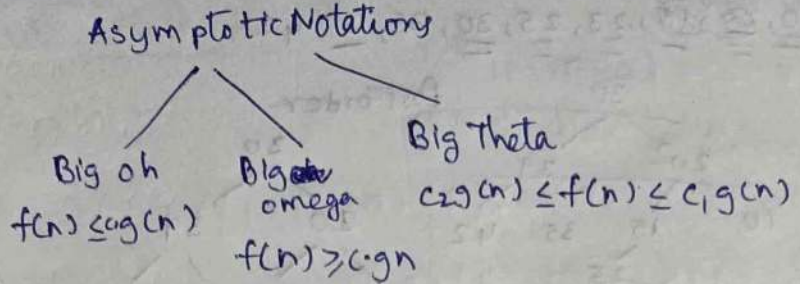
```
for(i=0; i<n; i++)
{
  for(i=0; i<n^2; i++)
   {
  for(i=0; i<n^3; i++)
   {
     cout<<i;
   }
   }   O(n^3)
}
```

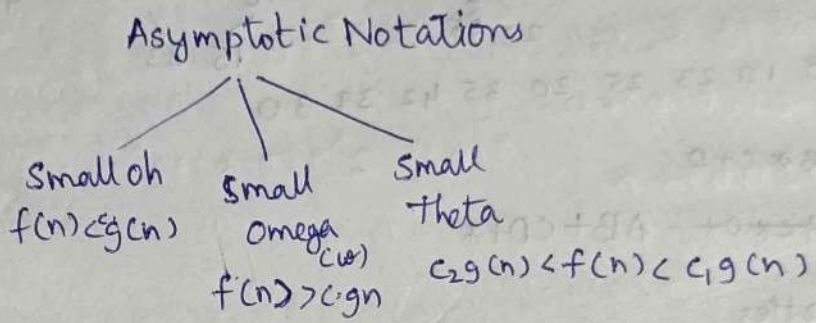
```
for(i=0; i<n^3; i++)
{
  for(i=0; i<n^2; i++)
  {
   for(i=0; i<n; i++)
   {cout<<i;
   }
   }   O(n^6)
}
```

```
for(k=n; k>=1; k=k/2)
{  x=y+z;   O(log n)
}
```

```
for(j=1; j<=n; 2* j)
{ for(k=n; k>=1; k=k/2)
{  x=y+z;    O((log n)^2)
}
```

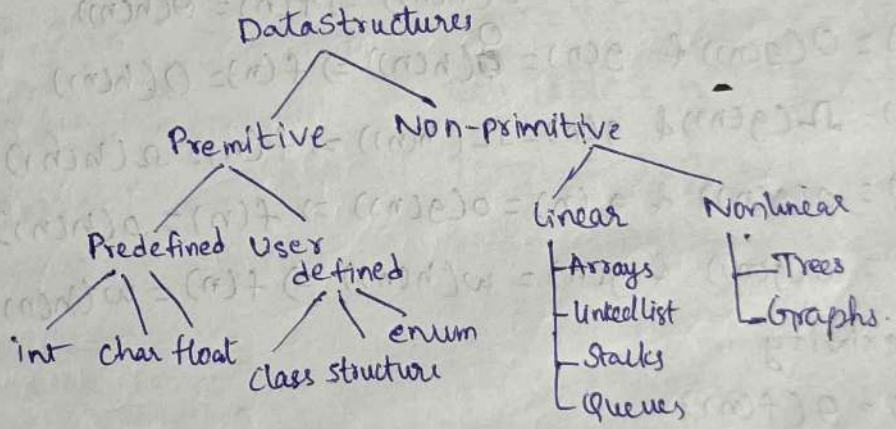
Asymptotic Notations



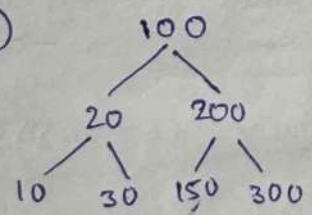


Data Structures

Program = Algorithms + Data Structures.



Q)

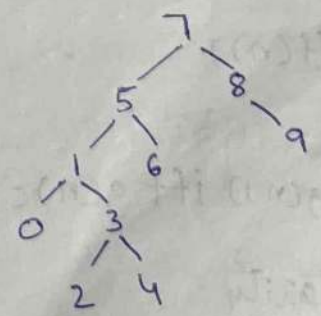


Inorder: 10, 20, 30, 100, 150, 200, 300

Preorder: 100, 20, 10, 30, 200, 150, 300

Postorder: 10, 30, 20, 150, 300, 200, 100

Q) 7, 5, 1, 8, 3, 6, 0, 9, 4, 2

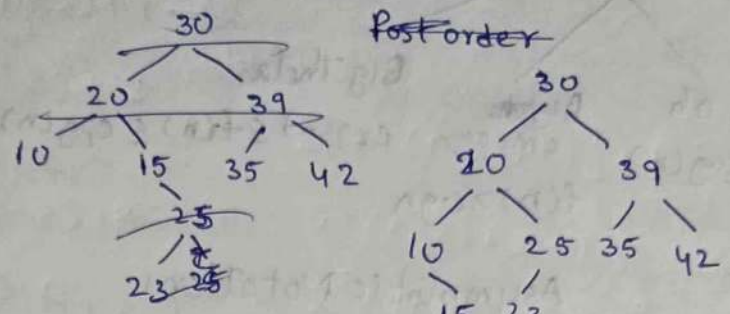


NLR

Q) Preorder - 30, 20, 10, 15, 25, 23, 39, 35, 42

Inorder - 10, 15, 20, 23, 25, 30, 35, 39, 42

LNR



Post - 15 10 23 25 20 35 42 39 30

Q) A+B*C+D

~~AB+C*D+~~ AB+CD+*

Properties

→ Transitivity

$f(n) = \theta(g(n))$ & $g(n) = \theta(h(n)) \Rightarrow f(n) = \theta(h(n))$

$f(n) = O(g(n))$ & $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$

$f(n) = \Omega(g(n))$ & $g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$

$f(n) = o(g(n))$ & $g(n) = o(g(n)) \Rightarrow f(n) = o(h(n))$

$f(n) = \omega(g(n))$ & $g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$

→ Reflexivity

$f(n) = \theta(f(n))$

$f(n) = O(f(n))$

$f(n) = \Omega(f(n))$

→ Symmentry

$f(n) = \theta(g(n))$ iff $g(n) = \theta(f(n))$

→ Complementarity

$f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

$f(n) = o(g(n))$ iff $g(n) = \omega((f(n)))$

Recurrance Relations

Decreasing function

↳ Substitution Method

```
void hello(int n)
{
  if(n==0)
   cout<<"Ok";
  if(n>0)
  {
   hello(n-1);
   cout<<"Fine";
  }
}
```

$T(n) = 1 \quad (n=0)$

$T(n) = T(n-1) + 1$ — ①

Replace n with n-1 in ①

$T(n-1) = T(n-2) + 1$ — ②

Substitute 2 in 1

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

⋮

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

Terminating condition $(k=n)$

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

$= T(0) + n$

$= 1 + n$

$T(n) = n.$

```
void hello(int n)
{
 if(n==0)
 cout<<"hello";
 if(n>0)
 {
  for(i=0; i<n; i++)
   cout<<"fine";
   hello(n-1);
  }
}
```

$T(n) = 1 \quad (n=0)$

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

$T(n) = T(n-1) + n$ — ①

Replace n with n-1 in ①

$T(n-1) = T(n-2) + n - 1$ — ②

Sub 2 in ①

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

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

⋮

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

Terminating condition $(k=n)$

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

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

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

$= n^2 - n + 2$

$T(n) = n^2$

Q) $T(n) = 2T(n-1) + 1$ — ① , $T(0) = 1$

Replace n with n-1

$T(n-1) = 2T(n-2) + 1$ — ②

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

⋮

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

~~Terminating $k = n$~~

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

~~$= 2T(0) + n$~~

Sub ② in ①

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

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

$T(n) = 2^2 T(n-2) + 2^1 + 2^0$

⋮

$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \cdots + 2^0$

Terminating condition $k = n$

$T(n) = 2^n T(0) + 2^{n-1} + 2^{n-2} + \cdots + 2^0$

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

$= 2^{n+1} - 1$

Dividing Function

Substitution

Q) $T(n) = T(n/2) + 1$ — ①

Replace n with $n/2$

$T(n/2) = T(\frac{n}{4}) + 1$ — ②

Substitute ② in ①

$T(n) = T(\frac{n}{4}) + 1 + 1$

$T(n) = T(\frac{n}{4}) + 2 = T(\frac{n}{2^2}) + 2^1$

⋮

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

Terminating condition is $2^k = n \Rightarrow k = \log_2 n$

$K = \log_2 n$

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

$= 1 + \log_2 n$

Q) $T(n) = T(n/6) + 1$ — ①

Replace n with $n/6$

$T(n/6) = T(\frac{n}{6 \times 6}) + 1$

$T(n/6) = T(\frac{n}{36}) + 1$ — ②

Sub ② in ①

$T(n) = T(\frac{n}{36}) + 2$

$T(n) = T(\frac{n}{6^k}) + k$

Terminating condition $6^k = n \Rightarrow k = \log_6 n$

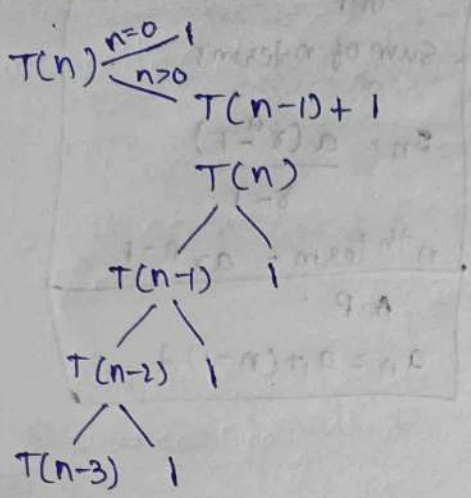
$T(n) = T(1) + \log_6 n$

$= 1 + \log_6 n$

Recursion Tree (Dividing Tree)

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

$T(n) \xrightarrow{n=0} 1$, $\xrightarrow{n>0} T(n-1) + 1$



Q) $T(n) = T(n/2) + n$ (Substitution M)

Replace n with $n/2$

$T(n/2) = T(\frac{n}{4}) + n/2$

$T(n) = T(\frac{n}{4}) + \frac{n}{2} + n$

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

$T(n) = T(\frac{n}{2^k}) + \frac{n}{2^{k-1}} + 2^{k-2} \cdot n + \cdots + 2^0 \cdot n = T(\frac{n}{2^k}) + n\left(\frac{1}{2^{k-1}} + 2^{k-2} + \cdots + 2^0\right)$

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

$T(n) = T(1) + \left(2^k + 2^{k-2} \cdot 2^k + \cdots + 2^k\right)$

$= \left(1 + 2 + 2^{2k-2} + \cdots + 2^k\right) 2^k \; 1 + 2^k \left(\frac{1}{2^{k-1}} + 2^{k-2} + \right.$

$= n + 2 + 2^{2(k+1)} + \cdots + 2^k \; 1 + 2 \; 2^k \left(2^{k-2} + 2^{k-3} + \cdots 1\right)$

$= 1 + 2 + 2^k [2$

$T(n) = T(1) + 2^k (2 + 4 + \cdots \cdot 2^k \left(\frac{1}{}\right.$

$T(n) = T(\frac{n}{2^k}) + \frac{n}{2^{k-1}} + \frac{n}{2^{k-2}} \cdots + \frac{n}{2^0}$

$n \Rightarrow k = \log_2 n$

$$T(n) = T\left(\frac{n}{2^k}\right) + n\left[\frac{\left(\frac{1}{2}\right)^k - 1}{\frac{1}{2} - 1}\right] = T\left(\frac{n}{2^k}\right) + n\left[\frac{\left(\frac{1}{2}\right)^k - 1}{-1/2}\right]$$

$$= T\left(\frac{n}{2^k}\right) + 2n\left[1 - \left(\frac{1}{2}\right)^k\right]$$

$k = \log_2 n$

$$= T(1) + 2n\left(1 - \frac{1}{2}(\log_2 n)\right)$$

$$= 1 + 2n - 2n(2)^{\log_2 \frac{1}{n}}$$

$$= 1 + 2n - 2n \times \frac{1}{n} = 2n - 1$$

G.P

sum of n terms

$$S_n = \frac{a(r^n - 1)}{r - 1}$$

n^{th} term $= ar^{n-1}$

A.P.

$a_n = a_1 + (n-1)d$

Advance Limit functions

$\lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c$ and $c \to 0-1$ Ω (Big Omega)

(0 is not considered here)
(1 is considered here)

$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c$ and $c > 1$ O (Big Oh)

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \neq 0$ and Ω Θ (Theta)

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ small o

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ ω (small omega)

Q) $f(n) = 5n + 10$, $g(n) = 2n$,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{5n+10}{2n} = \lim_{n\to\infty} \frac{n+2}{n} = \lim_{n\to\infty} \frac{n}{n} + \lim_{n\to\infty} \frac{2}{n}$$

$$= 1 + \frac{2}{\infty} = 1 = \lim_{n\to\infty} \frac{5n}{2n} + \lim_{n\to\infty} \frac{10-5}{2n}$$

$$= \frac{5}{2} + \lim_{n\to\infty} \frac{5}{n} = \frac{5}{2} + 0 = \frac{5}{2}$$

Big Oh.

Q) $f(n) = n!$ $g(n) = n^n$

$$\lim_{n\to\infty} \frac{n!}{n^n} = \lim_{n\to\infty} \frac{n(n-1)!}{n^n} \leq \lim_{n\to\infty} \frac{(n-1)!}{n^{(n-1)}}$$

$$= \lim_{n\to\infty} \frac{n(n-1)(n-2)\cdots 1}{n^n}$$

$$= \lim_{n\to\infty} \frac{n^n\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\left(1-\frac{3}{n}\right)\cdots \frac{3}{n}\cdot\frac{2}{n}\cdot\frac{1}{n}}{n^n}$$

$= 0$

~~Big Omega~~. Small O

Q) $\lim_{n\to\infty} \frac{3^n}{2^n} = \lim_{n\to\infty} \left(\frac{3}{2}\right)^n = \lim_{n\to\infty} (1.5)^n = \infty$

Small omega.

Q) $\lim_{n\to\infty} \frac{n^2+2n+1}{n} = \lim_{n\to\infty} n + 2 + \frac{1}{n} = \infty + 2 + 0$

$= \infty$

Small omega.

Q) $\lim_{n\to\infty} \frac{n^2+2n+1}{n^4} = \lim_{n\to\infty} \frac{1}{n^2} + \frac{2}{n^3} + \frac{1}{n^4} = 0$

Small O.

Master's theorem (Difference function)

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

where $f(n) = n^k$

$a, b > 0$, $k \geq 0$

or

$T(n) = aT(n-b) + n^k$

① If $a = 1$

$O(n^{k+1})$

② If $a > 1$

$O(a^{n/b} \cdot n^k)$

③ If $(a < 1)$

$O(n^k)$ or $O(f(n))$

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

$T(n) = aT(n-b) + n^k$

$a=1, b=1, k=1$

$n^1 = n^k$

$k=1$

Here $a=1$

So, $O(n^{1+1})$

$= O(n^2)$

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

$T(n) = aT(n-b) + n^k$

$a=2, b=1, k=1$

Here $a>1$

$O(a^{n/b} \cdot n^k)$

$O(2^n \cdot n)$

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

$T(n) = aT(n-b) + n^k$

$a = 1/2, b=1, k=1$

Here $a<1$

$O(n^k) \Rightarrow O(n)$

Master's Theorem (Dividing Function)

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

$f(n) = n^k \log^p n$ $\quad a \geq 1,\ b > 1$] p is an integer.

Case 1

If $\log_b a > k$

$\Rightarrow O(n^{\log_b a})$

Case 2

If $\log_b a = k$

→ If $p > -1$

$O(n^k \log^{p+1} n)$

→ If $p = -1$

$O(n^k \log\log n)$

→ If $p < -1$

$O(n^k)$

Case 3

If $\log_b a < k$

→ If $p >= 0$

$O(n^k \log^p n)$

→ If $p < 0$

$O(n^k)$

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

$T(n) = aT(n/b) + n^k \log^p n$

$a=1, b=2, k=1, p=0$

$n' = n^k \log^p n$

$\log_2 1 = 0$

$0 < 1 \Rightarrow 0 < k$

Here $\log_b a < k$

$p = 0$

Then $O(n' \log^0 n)$

$O(n)$

Q) $T(n) = 8T(n/2) + n$

$T(n) = aT(n/b) + n^k \log^p n$

$a=8, b=2, k=1, p=0$

$\log_2 8 = 3$

$3 > 1$

Here $\log_b a > k$

$\Rightarrow O(n^{\log_b a})$

$\Rightarrow O(n^{\log_2 8}) = O(n^3)$

Q) $T(n) = 2T(n/2) + n \log n$

$a=2, b=2, k=1, p=1$

$\log_2 2 = 1$

$k = \log_b a$

$p = 1, p > -1$

$O(n \log^{A+1} n) = O(n \log^2 n)$

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

$T(n) = aT(n/b) + n^k \log^p n$

$a=2, b=2, k=1, p=0$

$n' = n^k \log^p n$

$\log_b a = \log_2 2 = 1$

$k=1$ & $\log_b a = 1$

$p=0$ Here $p > -1$

$O(n \log^{p+1} n)$

$O(n \log n)$

Q) $T(n) = 8T(n/2) + n \log n$

$T(n) = aT(n/b) + n^k \log^p n$

$a=8, b=2, k=1, p=1$

$\log_2 8 = 3$

$3 > 1$

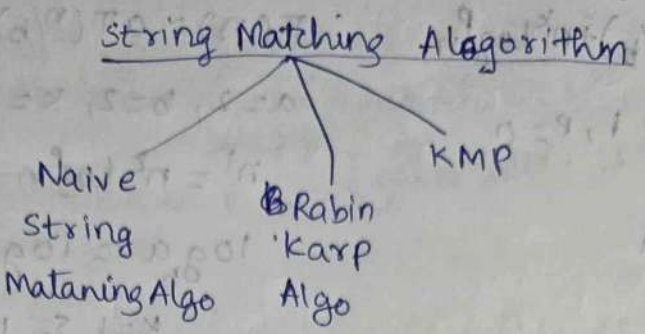
Here $\log_b a > k$

$\Rightarrow O(n^{\log_b a})$

$\Rightarrow O(n^{\log_2 8}) = O(n^3)$

String Matching Algorithms

Unit 2



Naive string Mataning Algorithm

Ex:- Text acaabc

Pattern aab

S=0 [aab ≠ aca]

S=1 [caa ≠ aab]

S=2 [aab = aab]

S=3 [abc ≠ aab]

~~Shifts is~~

Algo

```
Naive string Matching (T, P)
{
n ← length(T)
m ← length(P)
for s=0 to n-m {
  ~~if P[1 to m]~~
  if P[1.. m] = T[s+1 ..... s+m]
Cout << "Pattern occurs at shift s"
 }
}
```

* ~~Shifts is~~ No of shifts = n-m+1

n - length of text

m - " " Patter

* Complexity is [n-m+1]*m
* Complexity in worst case is nm

Ex:- Text 16 abcaaeecbbaadeaf

Pattern 4 baad shifts = 16-4+1 = 13

Rabin Karp Algorithm

Pattern = aab

1+1+2 = (4) → Hashcode

hash function

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

Text- a a a a a b

1+1+1=3

1+1+1=3

1+1+1=3

1+1+2=4.

Ex:- Pattern = abce

2+3+5=10

Text = a b c d a b c e

1+2+3=6

2+3+4=9 or 1+2+3-1+4=9

Rolling hash function

* Rolling hash function is applied only on Text.

Ex:- Pattern - dba

4+2+1=7

Text - c c a d c a a e d b a

3+3+1=7

3+1+3=7

1+3+3=7

} spurious hits

* Spurious hits means if value of pattern (3) & Text (3) units is equals and the pattern is different then known as spurious hits. In above example dba means 4+2+1=7 & cca means 3+3+1=7, here both values are same but the words are different. This is known as spurious hits.

*To counter spurious hits we changing the hash function.

Ex:- Pattern – d b a

m = 3 $4\times10^{3-1}+2\times10^{3-2}+1\times10^{3-3} = 400+20+1 = 421$

Text - c c a c c a a e d b a

$3\times10^2+3\times10^1+1\times10^0$

$= 331$

$3\times10^2+1\times10^1+3\times10^0$ (or) $\left[[3\times10^2+3\times10^1+1\times10^0]-3\times10^2\right]\times10+3\times10^0$

$= 313$

Rolling Hashing function

Algo

~~Rabin Karp Algo~~

Rabin Karp Algorithm (T, P, d, q)

p – small p

P – captical P.

```
{
  n ← length(T)
  m ← length(P)
  h ← d^(m-1) mod q
  p ← hash value(P) ← 0
  t ← hashvalue(T) ← 0
  for i = 1 to m
  {
    p ← (dp + P[i]) mod q
    t ← (dt + T[i]) mod q
  }
  for s = 0 to n-m
  {
    if (p == t)
    {
      if P[1..m] == T[s+1..s+m]
      {
        " pattern matched"
      }
    }
    else if (s < n-m)
    { t = (d(t - T[s+1]h) + T(s+m+1)) mod q
    }
}
```

Eg:- P = 45

m=2 P[1] P[2]

Text - 2 3 4 5 6 7 8 9 . . .

n=8 T[1] T[2] T[3] . . . . . T[8]

$h \leftarrow 10^{2-1} \bmod 13 \leftarrow 10 \bmod 13 = 10$

$P \leftarrow 0, t \leftarrow 0$

i=1

$P \leftarrow (10*0+4) \bmod 13 \leftarrow 4$

$t \leftarrow (10*0+2) \bmod 13 \leftarrow 2$

i=2

$P \leftarrow (10*4+5) \bmod 13 \leftarrow 45 \bmod 13 \leftarrow 6$

$t \leftarrow (10*2+3) \bmod 13 \leftarrow 23 \bmod 13 \leftarrow 10$

$(10(10-2(10)+4) \bmod 13$

$(10(-10)+4) \bmod 13 \Rightarrow (-100+4) \bmod 13 \Rightarrow -96 \bmod 13 (-5 \bmod 13)$ then $13-5=8$

$= 8$

$(10(8-(3)10)+5) \bmod 13$

$(10(-22)+5) \bmod 13 \Rightarrow -215 \bmod 13 = 6.$

P[45] == T[45]

pattern matched.

* Complexity of Rabin Karp is $(n-m+1)*m$ ~~nm~~.

* " " " " in worst case nm

KMP Algorithm

* Given by Knuth Morris Pratt.

Ex:- Pattern a b c d a b c

Prefix a, ab, abc, abcd x

Suffix c, bc, abc

Ex: abc abf

① Pattern a b c a b e a b f

Pi-Table (or) π table

a	b	c	a	b	e	a	b	f
0	0	0	1	2	0	1	2	0
1	2	3	4	5	6	7	8	9

Ex:- Pattern a b c d e a b f a b c

a	b	c	d	e	a	b	f	a	b	c
0	0	0	0	0	1	2	0	1	2	3
1	2	3	4	5	6	7	8	9	10	11

Ex:- a a b c a d a a b e

a	a	b	c	a	d	a	a	b	e
0	1	0	0	1	0	1	2	3	0
1	2	3	4	5	6	7	8	9	10

KMP Algorithm

1) Create Pi table (π table) for pattern

2) if S[i] == P[j+1]

```
{
    i++;
    j++;
}
else
{
    move j to the index mentioned in Pi table
    if j can't be moved further
    move i
}
```

Eg: String abcabcabcabcabd

Pattern abd

Eg: i s abcabcabcabcabd

j p abd

	a	b	d
j=0	0	0	0
	1	2	3

i) s[i] = a
p[j+1] = a

ii) s[i] = b
p[j+1] = b

iii) s[i] = c
p[j+1] = d

iv) s[i] = a
p[j+1] = a

v) s[i] = b
p[j+1] = b

vi) s[i] = c
s[j+1] = a

vii) s[i] = a
p[j+1] = a

viii) s[i] = b
p[j+1] = b

ix) s[i] = c
s[j+1] = a

x) s[i] = c
s[j+1] = a

xi) s[i] = a
p[j+1] = a

xii) s[i] = b
s[j+1] = b

xiii) s[i] = a
s[j+1] = a

xiv) s[i] = b
s[j+1] = b

xv) s[i] = c
s[j+1] = d

* Complexity of KMP Algo is n+m i.e. O(n+m)

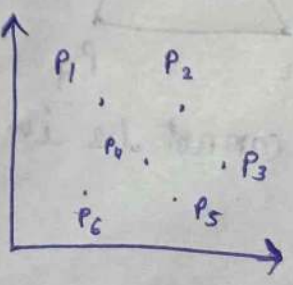
Sequential search and Brute Force

→ Closest Pair Problem
→ Convex Hull Problem.
→ Exhaustive search.
→ Voronoi Diagram.

Closest Pair Problem

* Geometrical Representation (2D Plane)
* Distance b/w points is used to calculate the closest pair.

Eg: (P_1, P_2) (P_2, P_3) (P_3, P_4) (P_4, P_5) (P_5, P_6)
(P_1, P_3) (P_2, P_4) (P_3, P_5) (P_4, P_6)
(P_1, P_4) (P_2, P_5) (P_3, P_6)
(P_1, P_5) (P_2, P_6)
(P_1, P_6)



$$D(P_i, P_j) = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$$

* Complexicity of Closest Pair Problem is $(n-1)n$
  i.e $O(n^2)$

Convex Hull Problem

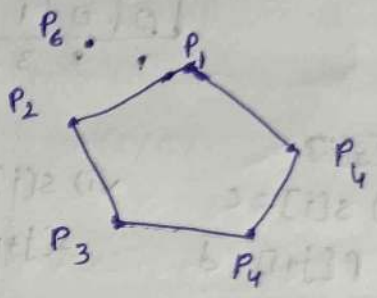
* Geometrical representation (2D plane)

$ax + by = c$

$a = y_2 - y_1$

$b = x_2 - x_1$

$c = x_1 x_2 - y_1 y_2$



* No lines should intersect each other.
* Points should lie only either side of line segment.

*Cases

i) $ax + by < c$

all points are above line segment

ii) $ax + by > c$

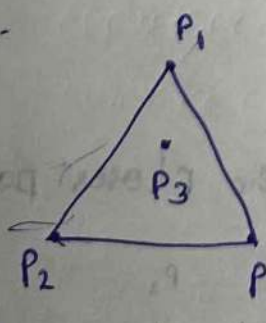
all points are bellow line segment

iii) $ax + by = c$

all points are on the line segment.

* Complexity of Convex Hull Problem is $O(n^3)$

Ex:-



$P_1 - P_2$ $P_3 \neq P_2$

$P_1 \neq P_3$ $P_3 \neq P_4$

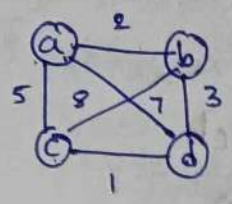
$P_2 - P_4$

$P_1 - P_4$

P_3 cannot be in Convex Hull

Exhaustive Search

Travelling Salesman Problem



Feasible solution

$a \to b \to c \to d \to a$	$2+8+1+7 = 18$
$a \to c \to b \to d \to a$	$5+8+3+7 = 23$
$a \to d \to b \to c \to a$	$7+3+8+5 = 23$
$a \to b \to d \to c \to a$	$2+3+1+5 = 11$
$a \to c \to d \to b \to a$	$5+1+3+2 = 11$
$a \to d \to c \to b \to a$	$7+1+8+2 = 18$

Optimal solution because giving less no. 11.

* Complexity of Exhaustive search is $(n-1)!$

Voronoi Diagram

* It is also a kind of geometric representation. (2D plane)
* In voronoi diagram, there must be only one point in a one block.

Ex: $P_1, P_2, P_3, P_4, P_5, P_6$

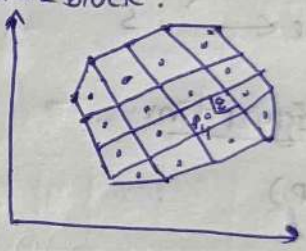
q

(P_1, q) (P_4, q)

(P_2, q) (P_5, q)

(P_3, q) (P_6, q)

(P_4, q) has less distance



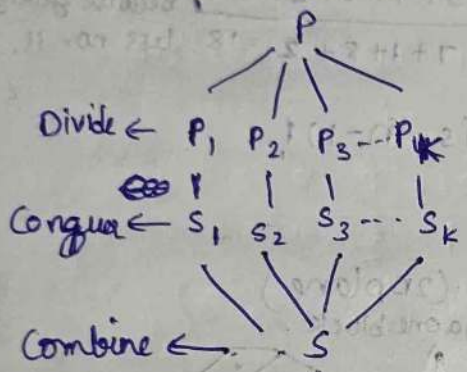
* It is combination of Closest Pair and Convex Pair problem.

Unit-3 Divide & Conquer & ordered statistics

statergies

~~1) Divide 2) conquer 3) Combine~~

1) Divide - Dividing big problem into subproblems
2) conquer - Solve subproblems + Recursive Approach.
3) Combine - Combine subsolutions to get final solution.



DAC Algorithm

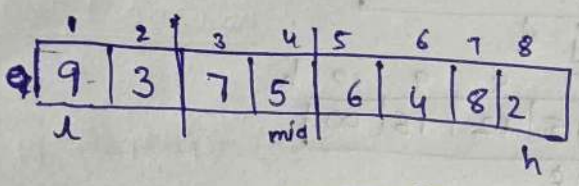
```
DAC (P)
{
  if (Small(P))
  {
      S(P)
  }
  else
  {
    Divide P into P1, P2, .. Pk
    Apply (DAC(P1), DAC(P2)....., DAC(Pk))
    Combine (DAC(P1), DAC(P2)... DAC(Pk) into S)
  }
}
```

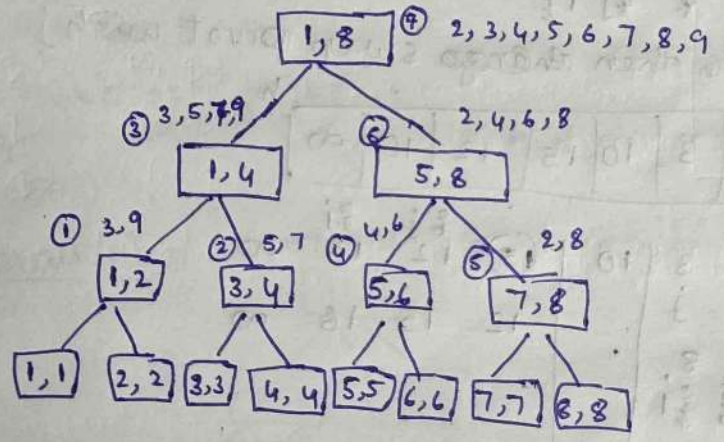
MergeSort

Algo

```
Mergesort(l,h)
{
  if(l<h)
  {
   mid=(l+h)/2
   Mergesort(l,mid)
   Mergesort(mid+1,h)
    Merge(l,mid,h)
  }
}
```

Q)





* Complexity of Mergesort is $O(n \log n)$

i.e. $T(n) = T(n/2) + T(n/2) + n$

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

$a=2, b=2, k=1, \log_b^a = \log_2^2 = 1$

$O(n \log^{a+1} n) \Rightarrow O(n \log n)$.

Quick Sort

Eg:- A=

l	1	2	3	4	5	6	7	8	9	10	h
	10	16	8	12	15	6	3	9	5	∞	
	i								j		

Pivot = 10

i > 10 then i++

j < 10 then j--

swap i & j

~~Increments are done upto i<j~~

i increments until i<j then swap pivot element to j

l	1	2	3	4	5	6	7	8	9	10	h
	10	5	8	12	15	6	3	9	16	∞	
			i						j		

l	1	2	3	4	5	6	7	8	9	10	h
	10	5	8	9	15	6	3	12	16	∞	
				i		j					

l	1	2	3	4	5	6	7	8	9	10	h
	10	5	8	9	3	6	15	12	16	∞	
						j	i				

Here i > j ⇒ 7 > 6 then swap pivot with j.

l										h
	6	5	8	9	3	10	15	12	16	∞

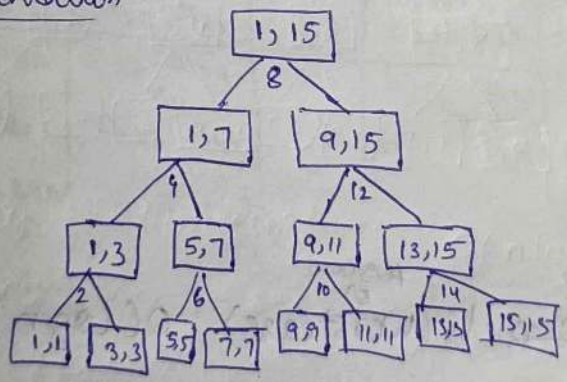
(6)	5	8	9	3	10	(15)	12	16	∞	
		i		j			j	i		
(6)	5	3	9	8			12	15	16	∞
		j		i						
3	5	6	9	8						
3	5	6	8	9						
3	5	6	8	9	10	12	15	16	∞	

Algo

```
Partition(l,h)
{
  Pivot = A[l];
   i=l;
   j=h;
 while(i<j
{
   do
    { i++;
    } while(A[i] ≤ Pivot)
```

```
    do
    {
      j--
    } while(A[i]>Pivot)
    if(i<j)
     swap(A[i],A[j])
    }
     swap(A[l],A[j])
     return j;
  }
Quicksort(l,h)
{
   if(l<h)
   { j=Partition(l,h);
     Quicksort(l,j);
     Quicksort(j+1,h);
   }
}
```

Tree representation



* Complexity of Quick sort when pivot at middle is $\Omega(n \log n)$ (Best or Average case)
* Complexity of Quick sort when pivot is at starting or ending but elements are sorted then $O(n^2)$ ie. $\frac{n(n+1)}{2}$ (worst case)

Binary search :-

Algo

```
Rbinarysearch (l, h, key)
{
    if (l == h)
    {
      if (key == A[l])
        return l;
      else
        return -1;
    }
    else
    {
      mid = (l+h)/2
      if (key == A[mid])
      return mid;
      if (key < A[mid])
      return Rbinarysearch (l, mid-1, key)
      else
      return Rbinarysearch (mid+1, h, key)
    }
}
```

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

* Complexity of Binary search (Avg case or worst case) $O(\log n)$
* Complexity of Binary search (Best case) 1.

Strassin's Matrix Multiplication

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}_{2 \times 2} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}_{2 \times 2}$$

$$A \times B = C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$

$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$

$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$

$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$.

Code

```
for (i=0; i<n; i++)
{
    for(j=0; j<n; j++)
    {
     for( k=0; k<n; k++)
     {
      c[i,j] = c[i,j] + a[i,k] * b[k,j]
     }
    }
}
```

$O(n^3)$

Now divide & Conquer strategy

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

(A_{11}, A_{12}, A_{21}, A_{22} and B_{11}, B_{12}, B_{21}, B_{22} mark the corresponding quadrants)

Algorithm MM(A, B, n)

```
{
  if (n<2)
  {
    4 formulae
  }
  else
  { mid n/2
```

MM(A_{11}, B_{11}, n/2) + MM(A_{12}, B_{21}, n/2)

MM(A_{11}, B_{12}, n/2) + MM(A_{12}, B_{22}, n/2)

MM(A_{21}, B_{11}, n/2) + MM(A_{22}, B_{21}, n/2)

MM(A_{21}, B_{12}, n/2) + MM(A_{22}, B_{22}, n/2)

```
}
```

$$T(n) = \begin{cases} 1 & n \le 2 \\ 8T(n/2) + n^2 & n > 2 \end{cases}$$

*Complexity is $O(n^3)$

Strassen's formula

$P - (A_{11} + A_{22})(B_{11} + B_{22})$

$Q - (A_{21} + A_{22}) B_{11}$

$R - A_{11}(B_{12} - B_{22})$

$S - A_{22}(B_{21} - B_{11})$

$T - (A_{11} + A_{12}) B_{22}$

$U - (A_{21} - A_{11})(B_{11} - B_{12})$

$V - (A_{12} - A_{22})(B_{21} + B_{22})$

$C_{11} - P + S - T + V$

$C_{12} - R + T$

$C_{21} - Q + S$

$C_{22} - P + R - Q + U$

$$T(n) = \begin{cases} 1 & n \le 2 \\ 7T(n/2) + n^2 & n > 2 \end{cases}$$

$= O(n^{2.81})$

Insertion Sort

A = | 8 | 5 | 7 | 3 | 2 |

Pass 1 :- Take out 5 → | 5 | 8 | 7 | 3 | 2 |

No. of comparison = 1
max. no. of shifts (swaps) = 1

Pass 2 : | 5 | 8 | 7 | 3 | 2 | ⇒ | 5 | 7 | 8 | 3 | 2 |

Take out 7

No. of comparision = 2
Max no. of shifts = 2

Pass 3 : | 5 | 7 | 8 | 3 | 2 | ⇒ | 3 | 5 | 7 | 8 | 2 |

Take out 3

No. of comparision = 3
Max no. of shifts = 3

Pass 4:- | 3 | 5 | 7 | 8 | 2 | ⇒ | 2 | 3 | 5 | 7 | 8 |

Take out 2

no. of comparisions = 4
max no. of shifts = 4

* no. of passes for insertion sort is $n-1$
* No. of comparisions is $1+2+3+4+\cdots n-1 = \frac{n(n-1)}{2}$
* Max no. of shifts is $1+2+3+4+\cdots n-1 = \frac{n(n-1)}{2}$
* Complexity of insertion sort is $O(n^2)$

Code

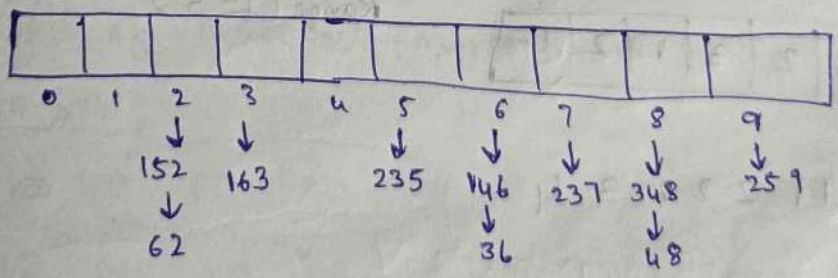
```
void insertionsort(A[], n)
{
   for (i=1; i<n; i++)
   {
      j = i-1
      x = A[i]
      while(i>-1 && A[j] > x)
      {
         A[j+1] = A[j]
            j--;
      }
      A[j+1] = x
   }
}
```

Radix Sort (Non Comparison based).

*It is example of Transform and conquer method.

A | 237 | 146 | 259 | 348 | 152 | 163 | 235 | 48 | 36 | 62 |

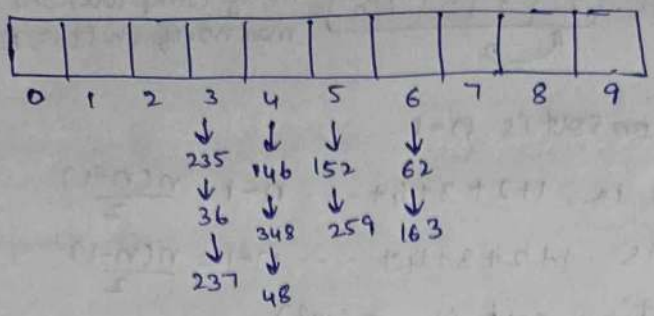
Pass 1



Pass 0: A[i] % 10 (or) (A[i]/1) % 10

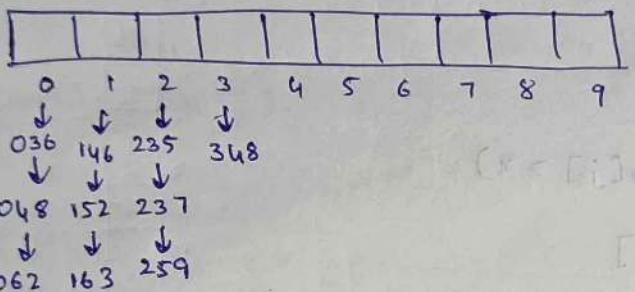
152, 62, 163, 235, 146, 36, 237, 348, 48, 259

Pass 2:- (A[i]/10) % 10



235, 36, 237, 146, 348, 48, 152, 259, 62, 163

Pass 3:- (A[i]/100)% 10



036, 48, 62, 146, 152, 163, 235, 237, 259, 348.

* Complexity of Radix sort is O(n) i.e. O(dn).

Count Sort (non comparision based)

* It is also transform & sort.

1) Given inputsize 'n'

2) Given Range. 'k'

Example | 2 | 1 | 2 | 3 | 1 | 2 | 4 | Range (1-5)

1 1 2 2 2 3 4

* Time Complexity of Countsort is O(n+k)

* Space complexity of Countsort is O(k)

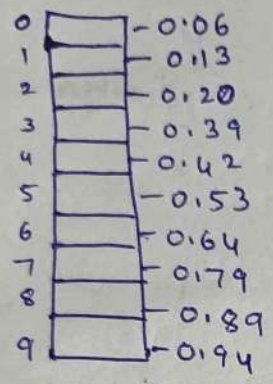
Bucket Sort (Non Comparision based)

→ Floating point values

0.1 - 1.0

Eg 0.79, 0.13, 0.64, 0.39, 0.20, 0.89, 0.53, 0.42, 0.06, 0.94

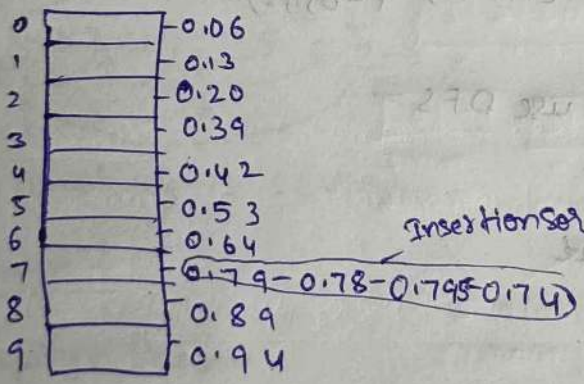
floor(A[i]*10)



* Complexity of Bucket sort (Best or Avg Case) is n.

Eg: 0.79, 0.13, 0.64, 0.39, 0.20, 0.89, 0.53, 0.42, 0.06, 0.94, 0.78, 0.795, 0.74

floor(A[i]*10)



* Complexity of Bucket sort (worst Case) is n^2+n i.e $O(n^2)$

Selection Sort

Eg: 40 60 20 10 50 30 (10 → min)

Pass 1: 10 60 20 40 50 30 (20 → min)

Pass 2:

Pass 2: 10 20 60 40 50 30 (30 → min)

Pass 3: 10 20 30 40 50 60

* Complexity of selection sort (Best, Avg, Worst) is $O(n^2)$.

Hashing

Linear search $O(n)$

Binary search $O(\log n)$

open hashing – chaining

closed hashing – linear probing
Quadratic probing.

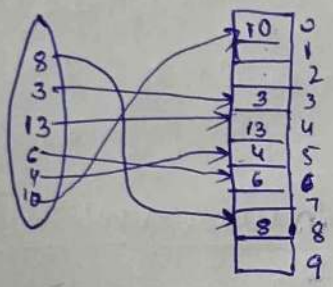
Ex:- 8, 3, 13, 6, 4, 10

Linear probing

~~(a) $h(x) = x \% size$~~ $h(x) = x$

$h'(x) = [h(x) + f(i)] \% size$

$f(i) = i$; $i = 0, 1, 2, \ldots$



Quadratic probing

$h(x) = x$, $h'(x) = [h(x) + f(i)] \% size$

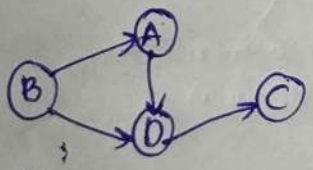
$f(i) = i^2$; $i = 0, 1, 2, \ldots$

Topological sort (graphs)

* we use DFS.

* Directed graph without cycles.

Ex:-



~~order B A D C~~

order

i) B A D C (u, v)

u v ✓

u v

ii) A B C D B (u, v)

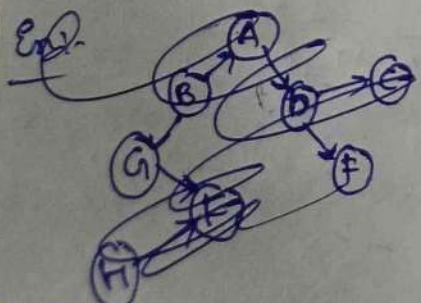
u v ✗

we must not consider this

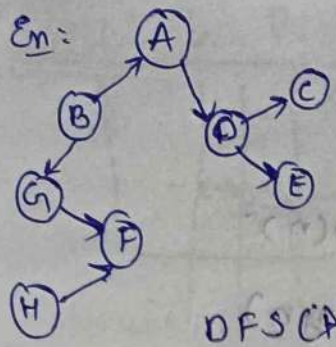
Imp

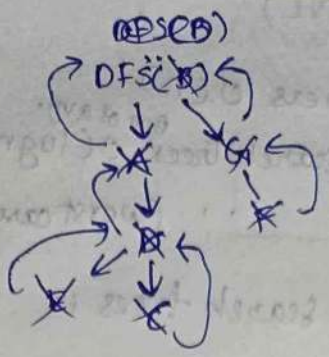
* Condition. u must appear before V.

Ex:-



Eg:





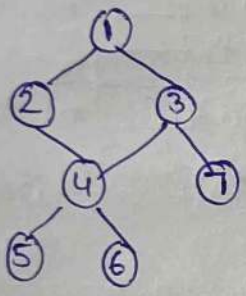
DFS(A)

↓

HBGFADEC

* Complexity is O(V+e)

BFS (Queues)

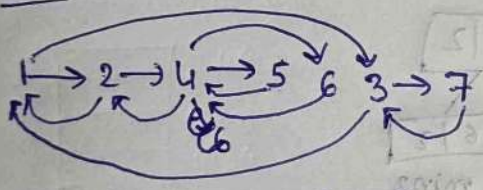


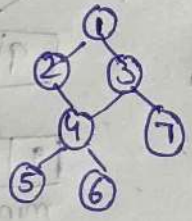
1	2	3	4	7	5	6

1 2 3 4 7 5 6

* Complexity is O(V+e)

DFS (Stacks)



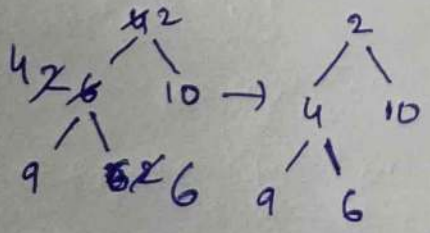


* Complexity is O(V+e)

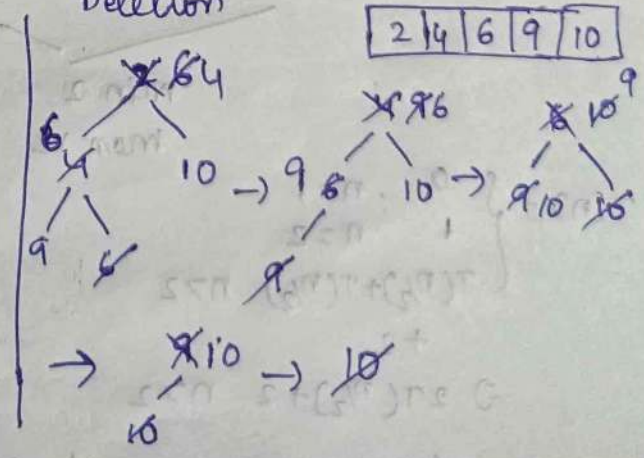
HeapSort

Heapify + Deletion

Eg: 4 6 10 9 2



Deletion



2	4	6	9	10

* Complexity of Heapify is O(n)

* Complexity of Heapify + Deletion

O(n) + O(n log n) i.e. O(n log n)

Balanced Search Trees (AVL)

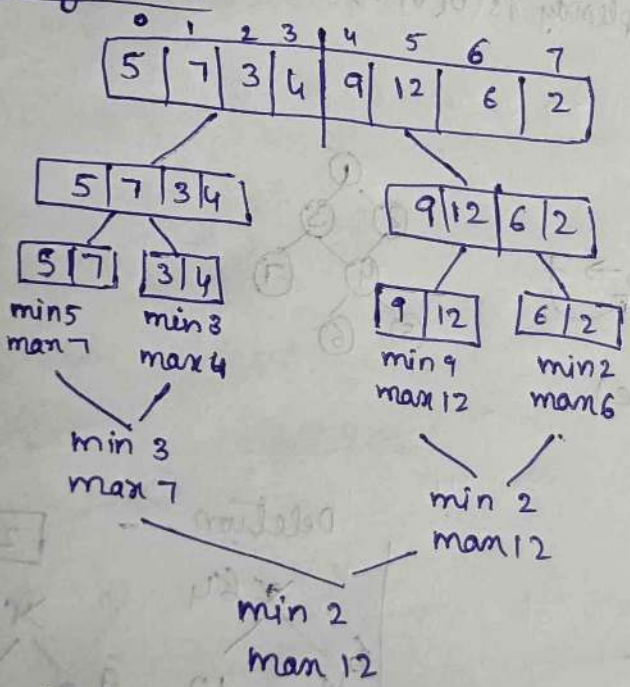
* Complexity of Binary trees O(n)
* Complexity of Binary search trees (Best & avg.) O(log n)
* " " " (worst case) O(n)
* Complexity of Balanced search trees is O(log n)

Balance factor :- -1, 0, 1

1) RR - 1 rotation - anticlockwise
2) LL - 1 rotation - clockwise
3) RL - 2 rotation - RR-anticlockwise
4) LR - 2 rotations - LL-clockwise

Min-Max Algorithm

Eg:-

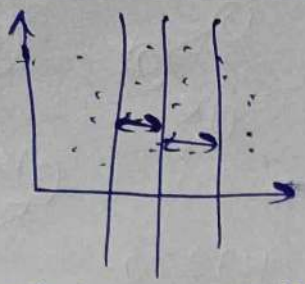


$$T(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ T(n/2)+T(n/2)+2 & n>2 \end{cases}$$

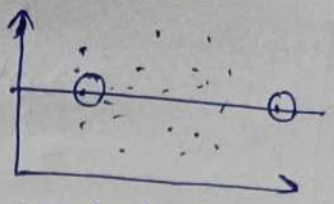
$\Rightarrow 2T(n/2)+2 \quad n>2$

* Complexity of Min Max Algorithm is O(n)

Closest Pair Problem



Convex Hull



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

Complexity of Convex Hull is i.e.

Connected components:-

Connected Graph:-

An undirected graph is called a connected graph if there is a path b/w any 2 vertices.

Connected components

A connected graph is a subgraph in which any two vertices in ~~which any 2 vertices~~ are

Closed hashing - linear probing

20

1 B	11 C	21 A
2 A	12 B	22 A
3 D	13 D	23 A
4 C	14 B	24 A
5 D	15 A	25 C
6 B	16 D	26 A
7 C	17 C	27 C
8 C	18 B	28 B
9 B	19 C	29 C
10 A	20 B	30 B

a^3+b^3

$(a+b)(a^2-ab+b^2)$

$(a+b)((a+b)^2-3ab)$

$1(1-3)$

-2

$a \cdot a^{20} = a \cdot a^{}$

$a^{10} = a^5$

$a \cdot a^{4} = a$

$a^5+b^5 = (a+b)(a^4 - a^3b + a^2b^2 -$