Sl. No. : 1780865

Guru Jambheshwar University of Science & Technology, Hisar (Haryana)

BACHELOR OF EDUCATION

Second Year Examination October, 2021

REGN.-CUM-ROLL NO	:	192082240044
NAME	:	DEEKSHA
FATHER'S NAME	:	Shri DINDAYAL VATS
MOTHER'S NAME	:	Smt. OMPATI

DETAIL OF MARKS

Sl. No.	Subject with Paper Code	Maximum Marks	Marks Obtained	Minimum Pass Marks
1.	BED-201 Knowledge and Curriculum	External 70 Internal 30	35 25	External 25 Aggregate 40
2.	BED-202 Assessment for Learning	External 70 Internal 30	26 26	External 25 Aggregate 40
3.	BED-203 Creating an Inclusive School	External 70 Internal 30	42 26	External 25 Aggregate 40
4.	BED-204 Language Across the Curriculum	External 35 Internal 15	16 12	External 12 Aggregate 20
5.	BED-205 Understanding Disciplines and Subjects	External 35 Internal 15	22 13	External 12 Aggregate 20
6.	BED-206 Gender, School & Society	External 35 Internal 15	19 13	External 12 Aggregate 20
7.	BED-208 Health,Physical and Yoga Education	External 35 Internal 15	22 12	External 12 Aggregate 20
8.	BED-216 Pedagogy of Mathematics	External 70 Internal 30	60 26	External 25 Aggregate 40
9.	BED-222 Pedagogy of Physical Science	External 70 Internal 30	60 26	External 25 Aggregate 40
10.	BED-226 School Based Activities	External 35 Internal 15	30 12	External 12 Aggregate 20
	Total [if passed]	750	523	300
	Marks Obtained in Ist Year	700	583	280
	Grand Total [if passed]	1450	1106	580

Result : Passed and has obtained One Thousand One Hundred Six Marks

HISAR
Dated: 27-12-2021

Asstt./Deputy Registrar (Exams.)

Serial No. TGT 045287
Identity Type. AADHAAR CARD
Identity No. 677487292020

Registration No. 579362
Roll No. 2031984

हरियाणा विद्यालय शिक्षा बोर्ड
Board of School Education Haryana

(AN ISO 9001 : 2015 CERTIFIED ORGANIZATION)

Deeksha

HARYANA TEACHER ELIGIBILITY TEST (HTET)
Level - 2 Trained Graduate Teacher (TGT)

This is to certify that **DEEKSHA**
Father's Name **DINDAYAL VATS**
Mother's Name **OMPATI**
Category **GENERAL**
Resident of **HISAR**
Date of Birth **02/03/1997 (TWO MARCH NINETEEN HUNDRED NINETY SEVEN)**
Subject Specific as opted **Science**

appeared in Haryana Teacher Eligibility Test for Trained Graduate Teacher's held on **December 03,2023** and performed as follows :-

Subject	Maximum Marks	Marks Obtained
Child Development & Pedagogy	030	29
Languages (Hindi & English)	030	21
General Studies	030	22
Subject **Science**	060	28
Total Marks	150	100
Result : QUALIFIED		

This certificate is valid upto **December 31,2030**

BHIWANI

DATED December 18,2023

SECRETARY

Qualifying the HTET would not confer a right on any person for recruitment/employment as it is only one of the eligibility conditions for appointment and the Candidate will have to fulfill other additional qualifications/conditions as prescribed in the relevant Service Rules while applying for the said post.

Formulaes

① **A.P**

$\text{term}_2 = \text{term}_1 + d$ or $a, a+d, a+2d, a+3d, \ldots$

(a) $t_n = a+(n-1)d$

↓ first term ↳ common difference

(b) no of term $= \frac{l-a}{d} + 1$

$l \rightarrow$ last term

(c) Sum of first n terms

$S_n = \frac{n}{2}[2a+(n-1)d] = \frac{n}{2}[a+l]$

(d) If a, b, c are in AP

$2b = a+c$

To solve most of the problems related to A.P., the terms can be conveiently taken as.

3 terms : $(a-d), a, (a+d)$

4 terms : $(a-3d), (a-d), (a+d)(a+3d)$

5 terms : $(a-2d)(a-d), a, a+d\ (a+2d)$

(e) $T_n = S_n - S_{n-1}$

(f) If each term of an A.P is ↑sed ↓sed, ✕ or ÷ by same non-zero constant, the resulting sequence also will be in A.P

② Harmonic Progression (H.P)

(a) Non-zero nos are in H.P if $\frac{1}{a_1}, \frac{1}{a_2}, \frac{1}{a_3}, \ldots \frac{1}{a_n}$ are in A.P

↓

$a_1, a_2, a_3 \ldots a_n$

(b) if $\frac{1}{a}, \frac{1}{a+d}, \frac{1}{a+2d} \ldots$ are in H.P

n^{th} term of H.P $= \frac{1}{a+(n-1)d}$

(c) a, b, c are in H.P

b is the Harmonic mean b/w a & c

$b = \frac{2ac}{a+c}$

or

$\frac{2}{b} = \frac{1}{a} + \frac{1}{c}$

③ Geometric Prog

ratio of any term and its preceding term is constant

$a, ar, ar^2, ar^3 \ldots$

$a \rightarrow$ first term $r =$ common ratio

(a) n^{th} term $\Rightarrow t_n = ar^{n-1}$

(b) Sum of 1st n terms $= S_n$

$$S_n = \begin{cases} a(r^n-1)/r-1 & (\text{if } r>1) \\ \frac{a(1-r^n)}{1-r} & (\text{if } r<1) \end{cases}$$

(c) Sum of an infinite G.P

$S_\infty = \frac{a}{1-r}$ (if $0<r<1$)

Example: $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \infty$

(d) a, b, c are in G.P

b = Geometric mean (G.m) $= \sqrt{ac}$

Note: if a & b are of opposite sign, their G.M is not defined.

(e) or $b^2 = ac$

(f) or $\frac{a-b}{b-c} = \frac{a}{b}$

product of terms equidistant from beginning and end will be constant.

To solve G.P problems, terms can be taken as

(a) 3 terms: $\frac{a}{r}, a, ar$

(b) 5 terms $\frac{a}{r^2}, \frac{a}{r}, a, ar, ar^2$

(4) Relationship b/w A.P, G.P, & H.P.

$Gm^2 = Am \times HM$ of 2 positive no.

(i) a, b, c are in A.P if $b = \frac{a+c}{2}$

(ii) a, b, c are in H.P if $b = \frac{2ac}{a+c}$

(iii) a, b, c are in H.P if $\frac{a-b}{b-c} = \frac{a}{c}$

(iv) $A > G > H$ of 2 positive no. (distinct)

A, G, H are in G.P.

if a series is both an A.P & G.P $\Rightarrow$ all the terms of the series will be equal.

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

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

$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4} = \left[\frac{n(n+1)}{2}\right]^2$

Formulas of Log

(1) $10^3 = 1000$ $\log_{10} 10^3 = 3 \Rightarrow \log_x x^n = n$

a = positive real no. other than 1

$a^m = x$

$\log_a a^m = \log_a x$

$m = \log_a x$

(2) $\log_x x = 1$

(3) $\log_a 1 = 0$

$\log_a x^n = n \log_a x$

$$\log_a x = \frac{1}{\log_x a}$$

$$\log_a x = \frac{\log_b x}{\log_b a} = \frac{\log x}{\log a}$$

Q1

$$T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2)+n^2 & \text{if } n>0 \end{cases}$$

$$T(18) = T(8) + 10^2$$
$$= T(6) + 8^2 + 10^2$$
$$T(4) + 6^2 + 8^2 + 10^2$$
$$T(2) + 4^2 + 6^2 + 8^2 + 10^2$$
$$T(0) + 4^2 + 6^2 + 8^2 + 10^2$$
$$1 + 4^2 + 6^2 + 8^2 + 10^2$$

$$T(n) = n^2 + (n-2)^2 + (n-4)^2 + (n-6)^2 + \ldots + 1$$

~~$n^2 + n^2 + 1$~~

$$\Rightarrow 1 + 2^2 + 4^2 + 6^2 + 8^2 + \ldots (n-2)^2 + n^2$$

Q2

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

$$T(10) = T(8) + 10$$
$$= T(4) + 8 + 10$$
$$= T(2) + 4 + 8 + 10$$
$$= T(1) + 2 + 4 + 8 + 10$$

$$T(12) = T(6) + 12$$
$$= T(3) + 6 + 12$$
$$= T$$

$$T(n) = T(n/2) + n$$
$$= T(n/4) + n/2 + n$$
$$= T(n/8) + n/4 + n/2 + n$$
$$= n + n/2 + n/4 + n/8 + \ldots + 1$$

Q3 $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 8T(n/2) + n^2 & \text{if } n>1 \end{cases}$

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

$= 8\left[T(n/4) + (n/2)^2\right] + n^2$

Formulaes

1) $1 + 2 + 3 + 4 + \ldots + n = \frac{n(2n+1)}{2}$

2) $1^2 + 2^2 + 3^2 + 4^2 + \ldots + n^2 = \frac{n^3(n^3+1)(2n^3+1)}{6}$

3) $\log_a b = \frac{\log_2 b}{\log_2 a}$

4) Sum of terms in g.P $= \frac{a(1-r^n)}{1-r}$

5) $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots + \frac{1}{n} = O(\log n)$

Priority	Specialization
1	CS & Engg
2.	VLSI
3.	Integrated Elec & Circuit
4	CoA
5	OS
6	CS & A
7	Info sys
8	N/W & Mob Comm
9.	S/W Tech
10	I.T
11	Comp. Appn

www.facebook.com / made.easy12

www.madeeasy.in ⟹ friday / 2:00p.m / weekend schedule
(What's New / Notification)

ALGORITHMS → 10Ques (Min)

References
1. Introduction to algo by coreman

Syllabus
1. Analysis
2. Divide & Conquer
3. Greedy Technique
4. ~~Dooing~~ Dynamic Prog.
5. Hashing, Graph, Tree, P, np, npc & nph

Definition:
It is a combination of sequence of finite steps to solve a particular problem.

ex: To add 2 no.

```
ATN()
{
1. Take 2 no.s (a,b)
2. C = add(a,b)
3. Return (c);
}
```

Properties of Algo
1. It should terminate after finite time
   finite steps doesn't mean finite time.
2. Minimum One output.

3) It should take 0 or more (finite) i/p.
4). It should be Deterministic / Non ambiguous.
5). It is prog. lang independent.

Steps Required to construct Algorithm

1. Problems definition
2. Design Algo. → Divide & conq, Dynamic proble, Back Tracking, Greedy Tech. } choosing any one is Algo design
3. Draw flowchart.
4. Testing.
5. Coding
6). Analysis → (finite Time & Space Complexity)

Chapter - 1 (Analysis)

Time & Space complexity

Time have higher priority (less CPU time is better)

CPU time costs more than memory space.

If any problem is having more than one solution then best one will be decided by analysis based on 2 - factors.

1. Time (CPU-time)
2. Space (main-memory)

Time Complexity

Total Time of Prog (P) = Compile Time (P) + Run Time (P)

= depends on compiler → Software → Prog lang of compiler

depends on CPU → hardware → Type of h/w processor

Analysis

Apostiary Analysis
It is dependent on prog lang of compiler and type of processor.

Apriori Analysis
It is independent of prog. lang of compiler & type of processor

Apostiary Analysis	Apriori Analysis
2. Exact Answer	2. Approx. Answer.
3. Answer changing system to system (cor. of diff. config of compiler lang & CPU)	3. Same Answer everytime
4. It is relative analysis cor. it keeps on changing with environment	4. It is absolute anal.

Apriori Analysis

It is a determination of order of magnitude of a statement.

ex:

```
main()
{
    x = y + z;        => 1 (no. of times it executed)
}
```

Time complexity = $O(1)$

```
main()                          =>  1
{
    x = y + z                   =>  1
    for(i=1; i<=n; i++)         =>  n+1
    {
        x = y + z;              =>  n.
    }
}
```

$n+1 = O(n)$

③ ex:

```
main()
{                                          -> (1)
    x = y + z;
    for(i=1; i<=n; i++)
    {                                      -> n
        x = y + z;
    }
    for(i=1; i<=n; i++)                    -> if here n/2 then 1+n+n^2/2
    {                                          => O(n^2/2)
        for(j=1; j<=n; j++)                    => O(n^2)
        {
            x = y + z;                     -> n*n.
        }
    }
}
```

$1 + n + n^2 \Rightarrow O(n^2)$

(13) Finding Time complexity means the place where CPU is spending largest time (largest loop)

(14)

```
main()
{
    i = 0;                                 -> (1)
    while(i<=n)                                1 + n/2 = O(n/2)
    {                                              = O(n)
        i = i + 2;                         -> n/2
    }
}
```

(5)

```
main()
{ while(n>=1)
  {
    n=n-1;
  }
}
```

→ (n) ⇒ O(n)

↳ 10 ⇒ n/10 ⇒ O(n)

(6)

```
main()
{ while(n>=1)
  {
    n=n/2;
  }
}
```

O(log₂n)

↳ 3 ⇒ O(log₃n)

Coz of division → log. came

① n/2 ③ n/2³ ⑤ n/2⁵

② n/2² ④ n/2⁴ ⑥ n/2⁶ .... n/2^k = 1

n/2^k = 1

⇒ n = 2^k

log₂n = log₂2^k

k = log₂n

(9)

```
main()
{ i=1
  while(i<n)
  {
    i=2*i;
  }
}
```

↓

5 ⇒ O(log₅n)

(10)

```
main()
{ i=1
  while(i<n)
  {
    i=5*i;
    i=i/10;
  }
}
```

(11)

```
main()
{ i=2
  while(i<n)
  {
    i=i^2;
  }
}
```

↳ i^3

⇒ (2)³ = 2³¹

(2³)³ = 2⁹ = 2³²

(2⁹)³ = 2²⁷ = 2³³

⋮

2^3k

i=1

2*1

2²

2³

2⁴

⋮

2^k = n

log₂2^k = log₂n

k = log₂n

i = 1 × i/10 × 2 ⇒ log₂n

suppose n=1000

2 < 1000

4 < 1000

16 < 1000

256 < 1000

256² > 1000

log₂(log₂n)

$2^{3^k} = n$

$\log_2 2^{3^k} = \log_2 n$

$3^k \log_2 2 = \log_2 n$

$3^k = \log_2 n$

$k \log_3 3 = \log_3 \log_2 n$

$k = \log_3 \log_2 n$

```
main()
{  i=15
   while (i<n)
   {
      i = i^7
   }
}
```

$15^{7^k} = n$

$7^k \log_{15} 15 = \log_{15} n$

$7^k = \log_{15} n$

$\log_7 7^k = \log_7 \log_{15} n$

$k = \log_7 \log_{15} n$

$n \Rightarrow n^{1/2} \Rightarrow (n^{1/2})^{1/2} = n^{1/4}$

$(n^{1/4})^{1/2} = n^{1/8}$

```
main()
{  while (n>2)
   {
      n = n^1/2 => sqrt(n)
   }
}
```

$n \Rightarrow n^{1/2} \Rightarrow n^{1/2^2}$

$= n^{1/2^3} = n^{1/2^4} \ldots n^{1/2^k}$

$n^{1/2^k} = 2 \Rightarrow$

$1/2^k \log_2 n = \log_2 2$

$1/2^k \log_2 n = 1$

$1/2 \log_2 n = 2^k$

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

$\log_2 \log_2 n = k \log_2 2$

```
main()
{  while (n>2)
   {
      n = n^1/5
   }
}
```

$O(\log_5 \log_2 n)$

```
main()
{  i=2
   while (i<n)
   {
      i = i^5
      i = i^1/10
      i = i^4
   }
}
```

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

$2^{3^k} = n$

$\log_2 2^{3^k} = \log_2 n$

$3^k \log_2 2 = \log_2 n$

$3^k = \log_2 n$

$k \log_3 3 = \log_3 \log_2 n$

$k = \log_3 \log_2 n$

```
main()
{ i=15
  while (i<n)
  {
    i = i^7
  }
}
```

$15^{7^k} = n$

$7^k \log_{15} 15 = \log_{15} n$

$7^k = \log_{15} n$

$\log_7 7^k = \log_7 \log_{15} n$

$k = \log_7 \log_{15} n$

$n \Rightarrow n^{1/2} \Rightarrow (n^{1/2})^{1/2} = n^{1/4}$

$(n^{1/4})^{1/2} = n^{1/8}$

```
main()
{ while (n>2)
  {
    n = n^1/2 ⇒ √n
  }
}
```

$n \Rightarrow n^{1/2} \Rightarrow n^{1/2^2}$

$= n^{1/2^3} = n^{1/2^4} \ldots n^{1/2^k}$

$n^{1/2^k} = 2 \Rightarrow$

$1/2^k \log_2 n = \log_2 2$

$1/2^k \log_2 n = 1$

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

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

$\log_2 \log_2 n = k \log_2 2$

```
main()
{ while (n>2)                    O(log_5 log_2 n)
  {
    n = n^1/5
  }
}
```

```
main()
{ i=2
  while (i<n)
  {
    i = i^5
    i = i^1/10
    i = i^4
  }
}
```

$\left.\begin{array}{l} i = i^5 \\ i = i^{1/10} \\ i = i^4 \end{array}\right\} i = i^2 = \log_2 \log_2 n$

Q1 Consider the following C prog.

```
main()
{
  for(i=1; i<=n; i=2*i)
  {
    for(j=n; j>5; j=j^1/10)
    {
      a=b+c;
    }
  }
}
```

$\log_{10}\log$
$n \Rightarrow n^{1/10} \rightarrow n^{1/10^2} \rightarrow n^{1/10^3} \cdots$

inner loop
$n^{1/10^k} = 5$
$(\log_{10}\log_5 n)$
outer $\rightarrow \log_2 n$
simultaneous $\Rightarrow O(\log_2 n \times \log_{10}\log_5 n)$

Q2 Consider the following C prog.

```
main()
{
  for(i=n; i>=10; i=√i)  → O(log2 log10 n)
  {
    for(j=5; j<=n; j=7*j)  → O(log7 n)
    {
      for(k=1; k<=n; k=k+100)
      {
        a=b+c;   → n/100 → O(n)
      }
    }
  }
```

$O(n * \log_7 n * \log_2 \log_{10} n)$

Q3

```
i=5
while(i<n)
{
  i=i+10;
  i=i/10
  i=i^10;
}
```

log n →
$\log_{10} n$
$\log_{10}\log_{5} n$ → this would effect more so, final answer would be this only.

Q Consider the following C prog.

```
Main()
{
  for (i=1; i<=n^5; i= 10*i)
  {
     j=15
     while (j< n^3)
     {
        j= j^8
     }
  }
}
```

$\Rightarrow \log_{10} n^5$

$\Rightarrow 5 \log_{10} n$

$15^8 = n^3$

$(15^8)^8 = n^3$

$(15^8)^k = n^3$

$\log_n 15^{8k} = \log_n n^3$

$8k \log_n 15 = 3$

$8k \log_{15} 15 = \log_{15} n^3$

$8k = \log_{15} n^3$

$\log_8 8k = \log_8 \log_{15} n^3$

$k = \log_8 \log_{15} n^3$

$\Rightarrow k = \log_8 3 \log_{15} n$

$5 \log_{10} n \log_8 3 \log_{15} n$

$\Rightarrow O(\log_{10} n \log_8 \log_{15} n)$

(remove constants)

Q Consider the following C prog.

```
main()
{
  for(i=1; i<=n; i++)
  {
    for(j=1; j<=i; j++)
    {
      for(k=1; k<=500; k++)
      {
        a=b+c;
      }
    }
  }
```

1 1 n
2 1,2 n+n
3 1,2,3

i=n
j= 1 2 3 n
k = 500

$n \times n^n \times 500^n$

$T = 1*500 + 2*500 + 3*500 + \ldots n*500$

$500[1+2+3+\ldots+n)$

$O\left(\frac{500 \times n(n+1)}{2}\right) = O(n^2)$

Q Consider the following C prog.

```
main()
{
  for(i=1; i<=n^2; i++)
  {
    for(j=1; j<=i^2; j++)
    {
      for(k=n; k>1; k=k/10)
      {
        a=b+c
        c=d+e
      }
    }
  }
```

i j k
1 1 n/10
2 1,2,3,4 (n/10)/10

$\log_{10} n$

$\log_2 n^2 \log\log n^2 \log_{10} n$

$i=1$	$i=2$
$j=1$	$j=1,2,3,4$
$k=\log_{10} n$	$k=\log_{10} n \quad \log_{10} n \quad \log_{10} n$

$i = n^2$

$j = 1 \ldots (n)^{n^2}$

$1^2 \log_{10} n + 2^2 * \log_{10} n + 3^2 \log_{10} n + \ldots n^2 \times \log_{10} n + (n^2)^2 \log_{10} n$

$\Rightarrow \log_{10} n [1^2 + 2^2 + 3^2 + \ldots + n^2 + (n^2)^2]$

$\log_{10} n \; \frac{n^2(n^2+1)(2n^2+1)}{6}$

Q

```
main()
{
for (i=1; i<=n^2; i++)
  for (i=1; i<=n^3; i++)
    for (i=1; i<=n^4; i++)
    {
      a = b+c;
    }
}
```

$O(n^4)$

$i \quad i \quad i$

$n^4 \quad n^3$

i

$i \quad 1$

$i \quad 1 \; 2 \; 3^4$

$i \; n^4 \; n^4 \; n^4$

$\log n^4$

n^9

$n^4 + n^4 + n^4$ (5 terms) $= n \times n^4$

$n^4 * n^5 \; n^9$

$n^4 + n^4 + n^4 \ldots$

```
main()
{
  for (i=1; i<=n^2; i++)
  {
    for (i=1; i<=n^5; i++)
    {
      for (i=1; i<=n^4; i++)
      {
      }
    }
  }
}
```

infinite loop

Q consider the following C prog.

```
A(n)
{
  if (n <= 2) return;
  else
    return (A sqrt(n))
}
```

while ($n \leq 2$)
$n = \sqrt{n}$

$O(\log_2 \log_2 n)$

Asymptotic Notations

c → is a const that differs with processor speed
Same algo run on diff. system have diff constant value c

1). Big-OH
2) Omega Notation
3). Theta Notation

Let $f(n)$ & $g(n)$ be 2 +ve functions (for from 'n' onwards that gives always +ve value).

Big-OH Notation

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

iff $f(n) \leq c \cdot g(n)$, $\forall n$, $n \geq n_0$

such that there exists 2 +ve const. $c > 0$ && $n_0 \geq 0$

@ex1 $f(n) = n + 10$

$g(n) = n$

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

$n + 10 \leq C\, g(n) \Rightarrow n + 10 = c \cdot n$

where $c \geq 0$ $n \geq 10$

@ex2 $f(n) = 3n^2 + n + 10$ $g(n) := n^2$

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

$3n^2 + n + 10 \leq c \cdot n^2$

Assume (n^2)

Then cn^2 should be atleast $5n^2$

so $c \geq 5$ $n \geq 3$

$\therefore 3n^2 + n + 10 \leq 5 \cdot n^2$, $\forall n$, $n \geq 3$

$3n^2+n+10 = O(n^2)$

ex 3 $f(n) = n^2$ $g(n) = n^2+10$

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

$n^2 = c.(n^2+10)$

$\Downarrow$ $\Downarrow$

1 0

$n^2 = O(n^2+10)$, $\forall\ n,\ n \geq 0$

ex 4

$f(n) = n+10$

$g(n) = n-10$

$f(n) = O(g(n)) \Rightarrow (n+10) = c(n-10)$

$n+10 = 2(n-10)$, $n \geq$ 30

ex 5

$f(n) = n^2$ $f(n) = O\,g(n)$

$g(n) = n$ $n^2 = c.n$ where $c \geq n$

$n \geq 1$

Big-Oh is not possible because the value of C equivalent to n and n is func..

$n^2 \neq O(n)$

Omega Notation

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

iff $f(n) \geq c\,g(n)$ $\forall\ n,\ n \geq n_0$ such that there exists 2 +ve constants $c > 0$ && $n_0 \geq 0$

ex 1

$f(n) = n^2$, $g(n) = n$

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

$n^2 \geq c.n$, $c = 1$, $\forall\ n,\ n \geq 0$

ex 2

$f(n) = n^2$

$g(n) = n^2+n+10$

$n^2 \geq c(n^2+n+10)$

$c = 1/3$ $n \geq 3$

ex 3

$f(n) = n-10$

$g(n) = n+10$

$n-10 \geq c(n+10)$

$n-10 \geq \frac{1}{10}(n+10)$ $\forall\ n,\ n \geq$ 13

ex 4

$f(n) = n$ $g(n) = n^2$

$n \geq c.n^2$

$\downarrow$ $c = 1/2$

$\therefore$ $1/n \rightarrow$ function

so $n \neq \Omega\, g(n^2)$

Theta Notation

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

iff

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

$\forall n,\ n \geq n_0$

ex1 $f(n) = n+10$

$g(n) = n+20$

(a) $f(n) \leq c\, g(n) \Rightarrow n+10 \leq c(n+20)$ $\quad c=1$

(b) $n+10 \geq c(n+20)$ $\quad n \geq 0$

$\frac{1}{2}$

ex2 $f(n) = n^2$

$g(n) = n^2+n+10$

~~$f(n)$~~ $n^2 \leq c(n^2+n+10)$

$\downarrow$ $\quad n_0 = 0$

1

$n^2 \geq c_2(n^2+n+10)$

$\downarrow$ $\quad n_0 = 3$

$\frac{1}{3}$

$n^2 = \Theta(n^2+n+10)$

$\forall n,\ n_0 \geq 3$

$c_1 = 1$

$c_2 = \frac{1}{3}$

ex $f(n) = n^2$

$g(n) = n^2$

$n^2 \leq c\, g(n^2)$

$\downarrow$ $\quad n \geq 0$

1

$n^2 \geq c_2(n^2)$

$\downarrow$ $\quad n \geq 0$

1

imp **Note** (11)

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

and also

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

this implies

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

and also vice versa

ex $f(n) = n^2$ $\quad g(n) = n$

$n^2 \leq c\, n$

$\downarrow$

function $= n$ so $O(g(n))$ is not possible

so $f(n) \neq \Theta g(n)$

ex $f(n) = n$ $\quad g(n) = n^2$

* $n \leq c_1 n^2$ $\quad n = O(n^2)$

$\downarrow$

1

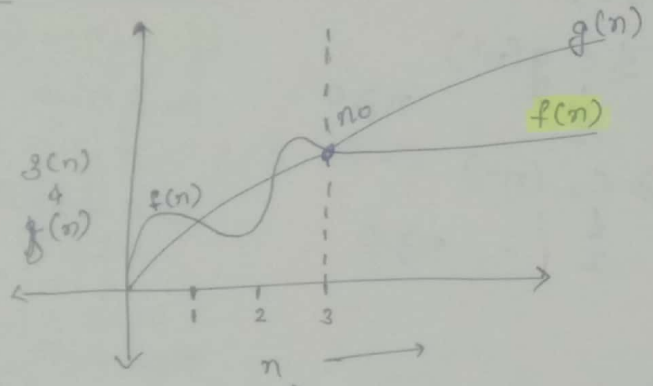
* $n \geq c_2(n^2)$

$\downarrow$

$\frac{1}{n}$ → function $\quad n \neq \Omega(n^2)$

so $n = \Theta(n^2)$ not possible

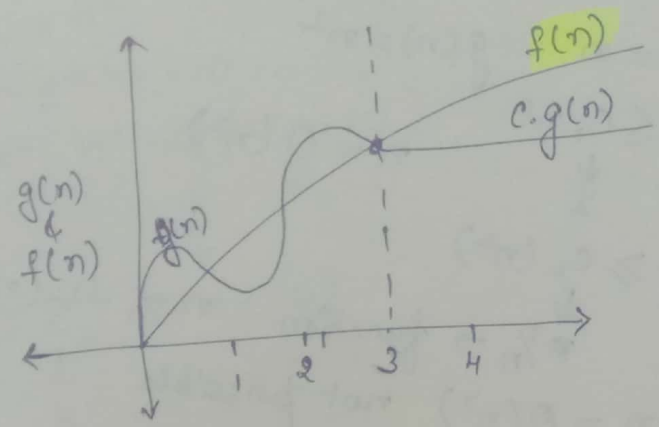
Big-OH Notation



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

$f(n) \leq c\,g(n), \quad \forall n, \; n \geq n_0$

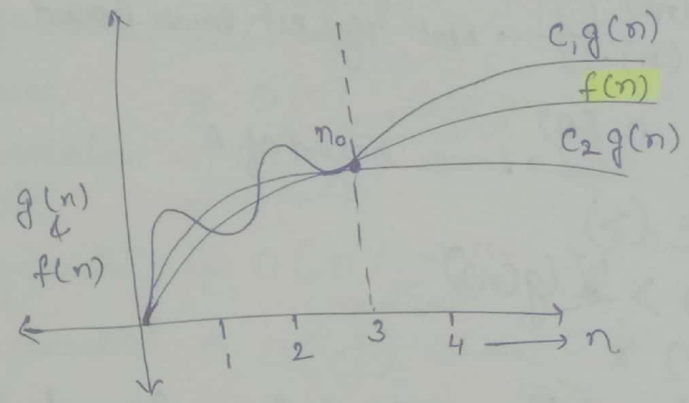
Omega



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

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

Theta Notation



Big-oh

It gives upper bounds.

$n^2 = O(n^2)$ ↳ tightest upb

In all the upper bounds ~~equo~~, the exactly equal upper bound is tightest upper bound

$n^2 = O(n^3)$, $n^2 = O(n^{10})$ → not tightest upper bound.

$A = O(B)$ maybe

Not Tight upper bound

tight upper bound.

Small-OH Notation (<)

$n^2 = o(n^2)$ ✗

$n^2 = o(n^3)$ ✓

$n^2 = o(n^{10})$ ✓

$A = o(B)$

B is upper bound of A but not tightest

Omega Notation

$n^3 = \Omega(n^3)$ → Tighest lower bound
$n^2 = \Omega(n^2)$
$= \Omega(n)$
lower bounds → Not tighest lower bound

$A = \Omega(B)$ → lower bound of A

Small Omega ($>$)

$f(n) > \omega(g(n))$

$n^3 \neq \omega(n^3)$ ✗
$= \omega(n^2)$ ✓
$= \omega(n)$ ✓
] Not tighest lower bound

Theta Notation

$n^3 = O(n^3)$ && $n^3 = \Omega(n^3) \Rightarrow n^3 = \Theta(n^3)$

$O(n^3)$: tighest upper bound
$\Omega(n^3)$: tighest lower bound
$\Theta(n^3)$: tighest upper & lower bound

* $T(A) = O(n^3)$
Algo-A worst case is n^3
* $T(A) = \Omega(n^3)$
Algo-A best case is n^3
* $T(A) = O(n^3)$ && $T(A) = \Omega(n^3)$
$\Rightarrow T(A) = \Theta(n^3)$

Algo-A best case & worst case both are same.

COMPLEXITY CLASSES

(13)

1) Constant $\Rightarrow O(1)$ (smallest time complexity)
2) Logarithmic $\Rightarrow O(\log n)$ → $\log\log n$ → $\sqrt{n}$
3) Linear $\Rightarrow O(n)$ → $n\log n$ → $n^{3/2}$
4) Quadratic $\Rightarrow O(n^2)$
5) Cubic $\Rightarrow O(n^3)$
6) Polynomial $\Rightarrow O(n^c)$ where $c > 0$ && c is a constant
7) Exponential $\Rightarrow O(c^n)$ where c is a constant && $c > 1$

Note:
$\log n < \sqrt{n}$

$\log 1000 = 10$
$\sqrt{1000} = \text{app. } 33$

$n^{3/2} = n^{1.5} = n^1 * n^{0.5} = n * \sqrt{n}$

8) $2^n < 3^n < 4^n \ldots$

$2^n \cdot 1 \;|\; 3^n$
$2^n \cdot 1 \;|\; (2 \times 1.5)^n$
$2^n \cdot 1 < 2^n \cdot (1.5)^n$ coz $1 < 1.5$

* $3^n \cdot 1 \;|\; 5^n$
$3^n \cdot 1 \;|\; \left(3 \times \frac{2.5}{3}\right)^n = 3^n \times \left(\frac{2.5}{3}\right)^n$
$1 < \frac{2.5}{3}$

9) $2^n = O(n^n) \Rightarrow 2^n = O(n^4)$ ✓

10) $n! = \Omega(2^n)$

11) $(\log n)^2 < n$

12) $(\log n)^4 < n$.

13) $(\log n)^{10000} < n$

14) $\log n$ always log will be smaller one than $\log n$.

15) $\log(\log n) < n$

16) $\log n < n$.

17) $(\log n)^2 \not< n$

18) $(\log n)^3 < n$

19) $(\log n)^{1000} < n$

20) $(\log n)^{\log n} > n$. $\log\log n \log n > \log n$

21) $2^n = \Omega 2^{n \log n}$

22) $\log_2 n > \log_{10} n$.

23) $\log_2 n = \Omega(\log_3 n)$

$\log_2 n$ | $\log_3 n$

$\Downarrow$

$\frac{\log_2 n}{\log_2 3} = \frac{1}{\log_2 3} \log_2 n = \frac{1}{1.5} \log_2 n$

$\log_2 n > \frac{1}{1.5} \log_2 n$

24. $\log_2 n = \Omega \log_3 n$

$\neq \omega(\log_3 \frac{n}{3})$

12. (i) $n = O(5n)$

$n \neq O(5n)$, $c = 1/5$

(ii) $10n = \Omega(n)$

$10n \neq \omega(n)$

$c = 10$

(iii) $n = O(n^2)$

$n = o(n^2)$

(iv) $n^2 = \Omega(n)$ ✓

$n^2 = \omega(n)$ ✓

Note

Small O satisfied then Big Scale is not necessary - that Big-Oh is to satisfied.

Sol (b) $\sqrt{\log n} = (\log n)^{1/2}$

$(\log n)^{1/2} = \Theta(\log \log n)$ true

$\frac{1}{2}\log \log n = \Theta(\log \log n)$ true

(c) $2^x \leq 2^y$

$x \leq y \Rightarrow$

$(\log n)^{1/2} \leq \Theta \log \log n$

$\frac{\log \log n}{2} \geq \log \log \log n$

So False.

(c) $2^x \leq 2^y$ true

$x < y$

(b) $\log(n+k)$

So. $(n+k)^n$

$2^{n+1} = O$

$2^n \cdot 2 = C \cdot 2^n$

true

(c) $2^{2n} = O($

2^{2^n} ↓ exponential

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

Note

* If Θ is possib

(e) $2^{2n} = O(2$

2^{n+n} ∴ 2

$2^n \cdot 2^n \neq 2$

↳

Note

* Don't apply log directly
* If you want to apply log first simplify and afterwards apply log.

Ques check the following statements are true or false

(a) $n^2 \cdot 16^{\log_4 n} = O(n^8)$

$n^x \cdot 16^{\log_4 n}$ $\quad n^{x+6}$

$16^{\log_4 n} = n^6$

$\log_4 n \log 16 \leq 6 \log n$ (true

(b) $\frac{4^n}{2^n} = O(2^n)$

$\frac{4^n}{2^n} = O(2^n)$

$2^n = O(2^n)$ true

$c = 1$

(c) $64^{\log_2 n} = O(n^5)$

$n^{\log_2 64} = n^5$

$n^6 \nleq c \cdot n^5$

↳ $c = n \Rightarrow n^6 = n^6$

false but c cannot be function so this is false

d) $\frac{1}{n} = O(1)$

true

$1/n \leq 1$

Note

1) $n^{\log_b a} = a^{\log_b n}$

2) $n^{\log_4 16} = n^2$ coz $4^2 = 16$

3) $\log_a b \Rightarrow$ if $a^x = b$ then we can write $\log_a b = x$

Anywhere log is given as power use above formulas

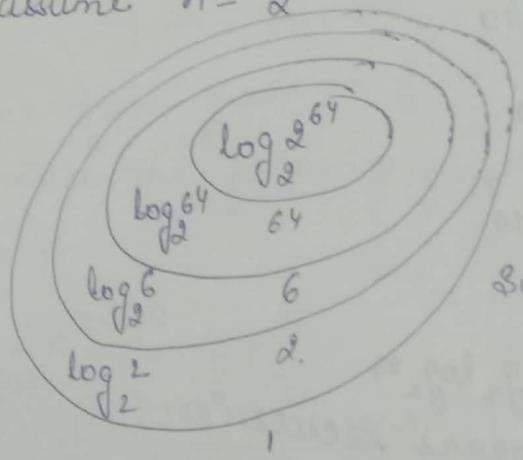
Ques consider the following 2 functions

$f(n) = \log^*(\log n)$

$g(n) = \log(\log^* n)$ - then find relation b/w $f(n)$ & $g(n)$

Soln

NOTE assume $n = 2^{64}$



To make $\log_2^* 2^{64} = 1$ we have applied log for 4 times

so $\log_2^* 2^{64} = 4$

$\log_2 2^{64} = 64 > \log_2 2^6 = 6 > \log_2^2$

Now to solve the above problem assume

$$n = \underbrace{2^{2^{2^{\cdots^{2}}}}}_{65,536}$$

$\log^* n = 65536$

$f(n) = \log^*(\log n)$

$= \log^* 65,535$

$= 65$

$g(n) = \log(\log^* n)$

$\log(65536) = 16$

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

ex 1 $f(n) = n - 10$

how many times will we sub 10 from n to make it 1

$f^*(n) = n/10$

ex 2 $f(n) = n/10$

$f^*(n) = \log_{10} n$

ex 3 $f(n) = \sqrt{n}$

$f^*(n) = \log_2 \log_2 n$

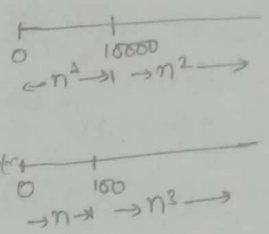
Note $f^*(n)$ here $(*)$ means recursion

Sub 6

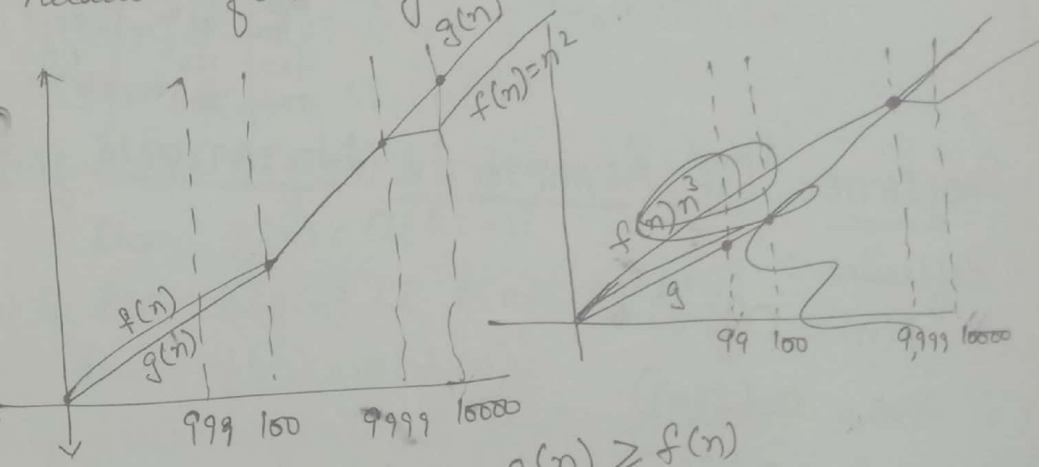
Consider the following two functions

$$f(n) = \begin{cases} n^3 & 0 \leq n < 10000 \\ n^2 & n \geq 10,000 \end{cases}$$

$$g(n) = \begin{cases} n & 0 \leq n < 100 \\ n^3 & n \geq 100 \end{cases}$$



relation $f(n)$ & $g(n)$



clear from graph → $g(n) \geq f(n)$

$n_0 > 9999$

Sub 7 consider the following functions

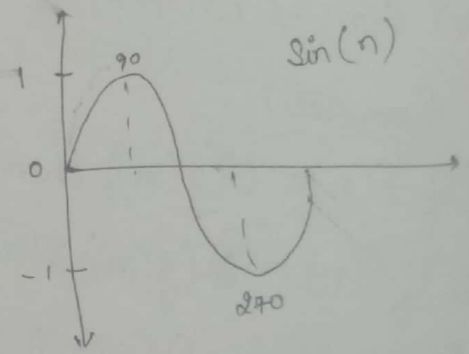
$f(n) = n$

$g(n) = n^{1+\sin n}$

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

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

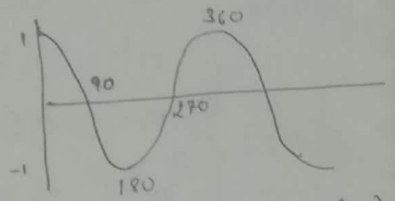
n	f(n)	g(n)
0	0	0
90	90	90^2
180	180	180
270	270	1
360	360	360



f(n) and g(n) are non comparable

Ques 8
what will be relation between the following func.

$f(n) = n^2$
$g(n) = n^{1+\cos n}$
$f(n) \geq g(n)$



n	f(n)	g(n)
0	0	$n^2 = 0$
90	90^2	90
180	180^2	↑
270	270^2	270

so $f(n) = \Omega g(n)$
$n_0 \geq 0$

PROPERTIES OF ASYMPTOTIC NOTATIONS

1. Reflexive :
(i) $f(n) = O(f(n))$
(ii) $f(n) = \Omega(f(n))$
(iii) $f(n) = \Theta(f(n))$
(iv) $f(n) \neq o(f(n))$
(v) $f(n) \neq \omega(f(n))$

2) Symmetric
(i) if $f(n) = O(g(n))$ then $g(n) \neq O(f(n))$
(ii) if $f(n) = \Omega(g(n))$ then $g(n) \neq \Omega f(n))$
(iii) if $f(n) = \Theta(g(n))$ then $g(n) = \Theta f(g(n))$

(iv) if $f(n) = o(g(n))$ then $g(n) \neq o(f(n))$
(v) if $f(n) = \omega(g(n))$ then $g(n) \neq \omega f(n))$

3). Transitive
(i) if $f(n) = O(g(n))$ & $g(n) = O(h(n))$
then $f(n) = O(h(n))$
(ii) if $f(n) = \Omega(g(n))$ && $g(n) = \Omega(h(n))$
then $f(n) = \Omega(h(n))$
(iii) if $f(n) = \Theta(g(n))$ && $g(n) = \Theta(h(n))$
then $f(n) = \Theta(h(n))$
(iv) if $f(n) = o(g(n)$ && $g(n) = o(h(n))$
then $f(n) = o(h(n))$
(v) if $f(n) = \omega(g(n))$ && $g(n) = \omega(h(n))$
then $f(n) = \omega(h(n))$

4). If $f(n) = O(g(n))$
then
$f(n) \cdot h(n) = O(g(n) \cdot h(n))$

5). If $f(n) = O(g(n))$ && $d(n) = O(e(n))$
then
(i) $f(n) + d(n) = O[\max(g(n), e(n))]$

(ii) $f(n).d(n) = O(g(n).e(n))$

Ques1. Let $f(n), g(n)$ & $h(n)$ be three positive functions which are ~~dif~~ defined as follows.

(i) $f(n) = O(g(n))$ && $g(n) \neq O(f(n))$ $\Rightarrow g(n) > f(n)$

(ii) $g(n) = O(h(n))$ && $h(n) = O(g(n))$ $g(n) = h(n)$

then check following stmts are true/false

(a) $f(n) = O(h(n))$. True

(b) $f(n) * g(n) = O(g(n) . h(n))$ True

(c) $g(n) . g(n) = O(g(n) . h(n))$ True

(d) $h(n) . f(n) = O(g(n), h(n))$ True

Ques2 Suppose $T_1(n) = O(f(n))$ && $T_2(n) = O(f(n))$

check the following stmts $T_1 \lesseqgtr_{T_2} f(n)$

(a) $T_1(n) + T_2(n) = O(f(n))$ True

(b) $T_1(n) = O(T_2(n))$ False

$T_2(n) = O(T_1(n))$ False

$T_1(n) = O(T_2(n))$ False

} can't be determined

DIVIDE & CONQUER (DAC)

1. Recursion
2. Recurrence Relation
3. Recurrence Relation Solving

RECURSION

A function is calling itself to solve a particular problem is called a recursion.

ex: $f(6) \rightarrow 6 \times f(5) \rightarrow 5 \times f(4) \rightarrow 4 \times f(3) \rightarrow 3 \times f(2) \rightarrow 2 \times f(1) \rightarrow$

Solving of Bigger Problems into smaller problems.

Note:

1. Recursion is nothing but solving big problems in terms of smaller problem.
2. To execute the recursive programs we use stack data structure.
3. Every recursive prog should have termination condition otherwise prog will go to infinite loop and finally it will produce a error message stack overflow.
4. In the recursive prog from one func called another func the value / parameter will be changed but not no. of parameter. otherwise the prog will go to infinite loop & finally will produce stack overflow error msg.

5. Comparing recursive and non recursive prog, recu... prog take more stack space because of more ... function calls.

6. For every recursion prog equivalent non recu... prog is possible with the help of loop (for, while loop)

7. Recursive prog are very easy to write comparison ... to non recursive.

From computer pt. of view non recursive prog are ... best while users " " " recursive prog are best

Q WA.P (Recursive) in recurrence relation to factorial of n.

```
fact (n)
{
  if(n==1) return (1)
  else
    return (n*fact(n-1));
}
```

Recurrence Relation

W.A.P & recurs... b/w 2 +ve no...

```
multipl...
{
  if(...
  else
  ...
}
```

```
mul...
{
  if(...
  else
  ...
}
```

T (multiply (m...

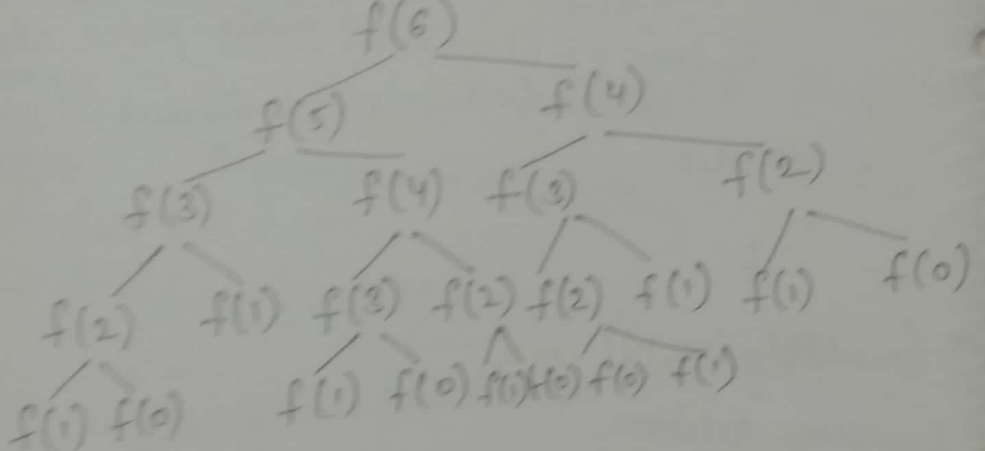
Recurrence Relat...

multiply(m,n)

WAP (recursive) to find nth fibonacci no.

```
fib(m, n, k)
{
  if(k=0)
    return(n)
  else
    return(n, m+n, k-1);
}
```

m n
0 1
1 m+n
0 1 1 2 3 5 8 13




```
fib(n)
{
  if(n==0||n==1)
    return n
  else
    return(fib(n-1)+fib(n-2))
}
```


Recurrence Relation

$$f(n) = \begin{cases} n & \text{if } n=0 \text{ or } n=1 \\ fib(n-1)+fib(n-2) & \text{otherwise} \end{cases}$$

$T(fib(n)) = O(2^n)$ — n level

↳ because binary tree is formed.

W.A recursive prog and recurrence relation to find gcd of two +ve no m & n.

```
gcd(m,n)
{
  if(m==0&&n==0)return(∞)
  if(m==0)
    return(n)
  if(n==0)
    return(m)
  return(m%n, m);
}
```

$$gcd(m,n) = \begin{cases} \infty & \text{if } m=0 \& n=0 \\ n & \text{if } m=0 \\ m & \text{if } n=0 \\ gcd(n\%m, m) & \text{otherwise} \end{cases}$$

$T(gcd(m,n)) = \log n$

GCD(23,29)

23)29(1
23
6

GCD(6,23)

6)23(3
18
5

GCD(5,6)

5)6(1
5
1

GCD(1,5)

1)5(5
5
0

gcd(0,1)
g

* if the given no.s are prime no. i.e the gcd of prime no is to be find then the gcd algo is in worst case.
* If both the no. are multiples of each other. then gcd algo would give the best case.

RECURRENCE RELATION SOLVING

1. Subsitution Method
2. Recursive Tree Method
3. Master - Theorem

SUBSTITUTION METHOD

Substituting the given function repealedly until given func. is removed.

ex 1

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

$T(50) = T(49)+50$

$T(48)+49+50$

$T(47)+48+49+50$

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

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

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

$\downarrow$ 10

$T(n-10)+(n-9)+(n-8)+\ldots+(n-1)+n$

$\downarrow n-1$

$= T(n-(n-1)+n-(n-2)+n-(n-3)+n-(n-4)+\ldots+(n-1)+n$

$T(1)+2+3+4+5+\ldots+(n-1)+n$

$1+2+3+4+\ldots+n$

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

$\log a + \log b + \log c = \log abc$

2x2

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

$T(10) = T(9)+\log 10$

$= T(9-1)+\log 9+\log 10$

$= T(7)+\log 8+\log 9+\log 10$

$\downarrow$

$T(1)+\log 2+\log 3+\ldots+\log 10$

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

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

$= T(n-3)+\log(n-2)+\log(n-1)+\log n$

$\downarrow$

$= T(n-(n-1))+\log(n-(n-2))+\log(n-(n-3))$

$+\ldots+\log(n-2)+\log(n-1)+\log n$

$= 1+\log 2+\log 3+\log 4+\ldots\log(n-1)+\log n$

$= 1+\log(2.3.4.5\ldots n)$

$1+\log(n!)$

$1 + \log n!$

$1 + \log n^n$

$1 + n \log n \Rightarrow O(n \log n)$

upper bound of $n! = n^n$

$n \times (n-1) \times (n-2) \times \ldots 1$

↓ ↓ ↓

assume 'n' at each place

so upper bound

$n \times n \times n \times \ldots$ (n times) $= n^n$

ex: $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2) + \log(n) & \text{if } n>0 \end{cases}$

$T(10) = T(8) + \log 10$

$T(6) + \log 8 + \log 10$

$T(4) + \log 6 + \log 8 + \log 10$

$T(2) + \log 4 + \log 6 + \log 8 + \log 10$

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

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

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

$T((n-2)-(n-3)) + \ldots + \log(n-6) + \log(n-4) + \log(n-2) + \log n$

$1 + \log 2 + \log 4 + \log 6 + \ldots + \log(n-2) + \log n$

$n - 2k = 0$

$k = n/2$

$1 + \log(2.4.6 \ldots n)$

$1 + \log 2^n (2.3.4.5.6 \ldots n/2)$

$T(n-2k) + \log(n-(2x-2)) + \log(n-(2x-4)) + \ldots \log(n-(2k-6)) + \ldots + \log n$

$= 1 + \log 2 + \log 4 + \log 6 + \ldots + \log n$

$1 + \log(2.4.6 \ldots n) = 1 + \log(2^n (1.2.3 \ldots n/2))$

$(n-2) - k = 1$

$n - 3 = k$

$\log a \cdot \log b = \log a + \log b$

$1 + \log(2.1) + \log(2.2) + \log(2.3) + \ldots \log(2 \times n/2)$

$1 + \log 2 + \log 1 + \log 2 + \log 2 + \log 2 + \ldots \log 2 + \log n/2$

$1 + n/2 + \log_2^2 + (\log 1 + \log 2 + \ldots \log n/2)$

$1 + n/2 + \log(1.2.3.4 \ldots n/2)$

$1 + n/2 + \log(n/2)!$

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

$= O(n \log n)$

Ques $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2) + n^2 & \text{if } n>0 \end{cases}$

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

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

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

Ques $T(n) = \begin{cases} 2 & \text{if } n=2 \\ \sqrt{n}\, T(\sqrt{n}) + n & \text{if } n>2 \end{cases}$

① $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2)+n^2 & \text{if } (n>1) \end{cases}$

$\Rightarrow T(n) = 1 + 2^2 + 4^2 + 6^2 + 8^2 + \ldots k^2$

$= \frac{(n^2)(2n^2+1)(n^2+1)}{6}$

$\Rightarrow O(n^6)$

Q3

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

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

$= 8\left[8T(n/2^2) + (n/2)^2\right] + n^2$

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

$= 8^3 T(n/2^3) + (n/2^2)^2] + 2n^2 + n^2$

$= 8^3 T(n/2^3) + 4n^2 + 2n^2 + n^2$

$= 8^k T(n/2^k) + 2^2 2^2 n^2 + 2^1 n^2 + 2^0 n^2$

$n/2^k = 1 \Rightarrow n = 2^k \Rightarrow \log n = k \log_2 2$

$\Rightarrow 8^{\log_2 n} T(n/2^k) + n^2[2^{k-1} + 2^{k-2} + \ldots + 2^1 + 2^0]$

$\Rightarrow 8^{\log_2 n} T(1) + n^2\left[2^0 + 2^1 + \ldots + 2^{\log n}\right]$

$n^3 \times 1 + n^2\left[1\frac{(2^{\log n}-1)}{2-1}\right]$

$= n^3 + n^2(n-1) = n^3 + n^3$

$= 2n^3 = O(n^3)$

(2)

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

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

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

$$= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2} + n$$

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

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \log n = k$$

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

$$= T(1) + n\left[\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \left(\frac{1}{2}\right)^{\log n}\right]$$

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

$$= 1 + n\left[\frac{1-0}{1/2}\right] = 1 + 2n$$

$r < 1$ → decreasing

whenever decreasing G.P series is there, G.P series is 1

(4) $$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n/2) + c & \text{if } n>1 \end{cases}$$ where c is const

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

$$= T\left(\frac{n}{2^2}\right) + c + c$$

$$= T\left(\frac{n}{2^3}\right) + c + c + c.$$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \log n = k$$

$$T\left(\frac{n}{2^{\log n}}\right) + c \log n$$

$$1 + \log n\, c$$

$$O(\log n)$$

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

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

$$= 2\left[2\left[2T\left(\frac{n}{2^3}\right)\right] + \frac{n/2^2}{\log\left(\frac{n^2}{2}\right)}\right] + \frac{n/2^2}{\log n/2^2} + \frac{n}{\log n}$$

$$2^3\left(T\left(\frac{n}{2^3}\right)\right) + \frac{n/2^2}{\log(n^2/2)} + \frac{n/2^2}{\log(n/2^2} + \frac{n}{\log}$$

$\Rightarrow 2^3 T\left(n/2^3\right) + \frac{n}{\log n/2^2} + \frac{n}{\log n/2^1} + \frac{n}{\log n}$.

$\Rightarrow 2^3 T\left(n/2^0\right) + n\left[\frac{1}{\log n/2^2} + \frac{1}{\log n/2^1} + \frac{1}{\log n/2^0}\right]$

$\Rightarrow 2^k T\left(n/2^k\right) + n\left[\frac{1}{\log n/2^0} + \frac{1}{\log n/2^1} + \frac{1}{\log n/2^n} + \cdots + \frac{1}{\log n/2^{k-1}}\right]$

$\Rightarrow 2^{\log_2 n} T(1) + n\left[\frac{1}{\log n/2^0} + \frac{1}{\log n/2^1} + \frac{1}{\log n/2^n} + \cdots + \frac{1}{\log n/2^{\log n - 1}}\right]$

$\Rightarrow n + n\left[\frac{1}{\log_2 n - 0} + \frac{1}{\log_2 n - 1} + \frac{1}{\log_2 n - 2} + \cdots + \frac{1}{\log_2 n - (\log_2 n - 1)}\right]$

$n + n[$

Q5 $T(n) = \begin{cases} 2 & \text{if } n=2 \\ \sqrt{n}\, T(\sqrt{n}) + n & \text{if } n>2 \end{cases}$

$T(n) = \sqrt{n}\, T(\sqrt{n}) + n$

$= n^{1/2} T(n^{1/2}) + n$

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

$= n^{3/2^2}\left(T(n^{1/2^2}) + n + n\right)$

$= n^{3/2^2}\left[n^{1/2^3} T(n^{1/2^3}) + n^{1/2^2} + n + n\right]$

$= n^{7/2^3} T(n^{1/2^3}) + n + n + n$

$n^{\frac{1}{2^k}} = 2$

$\frac{1}{2^k} \log n = \log_2 2 = 1$

$\log n = 2^k$

$\log \log n = k \log_2 2$

$\log \log n = k$

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

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

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

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

$= \frac{n}{2} \cdot 2 + n \log \log n$

$\Rightarrow n + n \log \log n \Rightarrow O(\log \log n)$

$n^{1/2^k} = 2^{\frac{1}{2^k} \log n} = \frac{1}{2^{\log \log n}} = \frac{1}{\log n}$

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

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

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

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

$T(10) = 2T(9)+1$

$= 2(2T(8)+1)+1$

$= 2(2(2T(7)+1)+1)+1$

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

$(2(2+1)+1)$

$6+1$

$n-k=1$

$k=n-1$

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

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

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

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

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

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

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

$= 2^3T(n-3)+2^2(n-2)+2(n-1)+n$

$\Rightarrow 2^{n-1}T(1)+2^{n-2}T(2)+\ldots 2^2(n-2)+2(n-1)+n$ → AP

$\Rightarrow 2^0(n-0)+2^1(n-1)+\ldots 2^{n-2}(2)+2^{n-1}(1)$ → G.P.

To solve A.P, G.P series (i.e combination of A.P & G.P)
Convert into GP.
Multiply whole series by 2

~~$2T(n) = 2^1(n-0)$~~ (here nothing)

$2T(n) = 0+2^1(n-0)+2^2(n-1)+\ldots+2^{n-2}.3+2^{n-1}.2+2^n.1$

$T(n)-2T(n) = n-2^1.1-2^2.1-2^3.1\ldots-2^{n-2}.1-2^{n-1}.1-2^n.1$

$-T(n) = n-[2^1+2^2+2^3+\ldots+2^{n-1}+2^n]$

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

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

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

$O(2^{n+1}) \Rightarrow O(n)$

Note

* $T(n) = 2T(n/2) + n$
  ans $n \log n$
* Now $T(n) = 2T(n/2 - 10) + n$

here also the ans = $n \log n$ coz '10' won't effect more

* $T(n) = 2T(\sqrt{n} + n)$
  Neglect 'n'
  $T(n) = 2T(n/2) + 2n$
  ans $\Rightarrow 2n \log n$
  $T(n) = 2T(n/2) + n/2$
  ans $= \frac{n}{2} \log n$

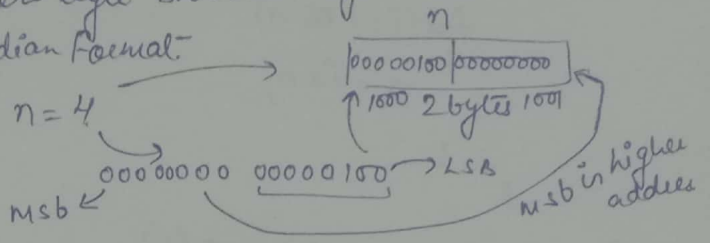
changes are in constant only

* How Recursive prog execute inside the comp.

main memory is 100 times faster than harddisk.

Slower byte stored in lower address (LB-LA)
and higher byte stored in higher address (HB-HA)

Little Endian Format



Space fact(n) = no. of function * One function space
= n * 6 Bytes
= 6n Bytes
= O(n)

Total space = 2 Bytes (in main func) + 6n Bytes (in sub fun fact)
= 6nB
= O(n)

```
main()
{
  int n;
  printf("%d", fact(n));
}
fact(int n)
{
  int a, b;
  if (n==1)
    printf(" return(1);
  else
  {
    a = fact-(n-1)
    b = a*n;
    return b;
  }
```

fact(n) Time = no. of func * Time of
= n * O(1)
= O(n)

for non-recursive factorial sub-prog

```
fact(int n)
{ int a,i;
  for(i=1 to n)
  {
    a = i*n;
  }
}
```

fact(n) space = 6B = O(1)

Total space = 2B + 6B = 8B = O(1)

Recursive prog will take more space but time we can't say (depends on logic)

DIVIDE & CONQUER (DAC)

It is used to solve big problems

1) Divide the given big problem into smaller size problems.
2) Conquer the sub-problems by calling recursively so that we will get sub problem solutions
3) Combine the sub-problem solutions to get original problem solution.

CONTROL ABSTRACTION OF DAC

```
DAC (a,P,q)
{
    if (small P,q,n)
        return (solution (a,P,q))
    else
        m = Divide (a,P,q)
        b = DAC (a,P,m)
        c = DAC (a,m+1,q)
        d = Combine (b,c)
        return (d)
}
```

How to find time Complexity of any problem if it is ~~used~~ to solved by divide and conquer.

$$T(n) = \begin{cases} O(1) & \text{if } n \text{ is small} \\ \text{if } n \text{ is big} & \\ \text{Divide} \Rightarrow f_1(n) & \\ n/2 \quad n/2 & \\ T(n/2) \quad T(n/2) & \\ \text{Combine } f_2(n) & \end{cases}$$

$$T(n) = \begin{cases} O(1) & \text{if } n \text{ is small} \\ f_1(n) + T(n/2) + T(n/2) + f_2(n) & \text{if } n \text{ is big.} \end{cases}$$

$$T(n) = 2T(n/2) + f_1(n) + f_2(n)$$

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

Ingeneral DAC Recurrence relation will appear look like as

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

$a \rightarrow$ no. of subproblems

n/b = size of subproblem

$f(n)$ = Divide and conquer func.

APPLICATIONS OF DAC

1) Find maxmin
2) Power of an element
3) Binary Search
4) Merge Sort
5) Quick Sort
6. Selection procedure
7) Strassen's Matix Multiplication.

1). Find MaxMin.

i/p: An array of n elements

o/p: Return Maximum element & & Minimum eleme

ex:

i/p: A[10, 20, 30, 11, 21, 31, 1, 2, 3]

o/p: min → 1
max → 31

without divide and conquer.

```
straightmaxmin(a, 1, n)
{
    max = min = a[1];
    for(i=2 ; i<=n; i++)
    {
        if(a[i]< min)
            min = a[i]
        else if(a[i] > max)
            max = a[i]
    } return(max, min)
}
```

Best case = n-1

worst case = n-1

Let $T(n)$ be the no. of comparisons for n-element in the above algo.

* Best case : Min. ant. of time req. to solve a problem.
No failure at all (always true)

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

(no. of comparisons)

* Worst case : max. ant of time req.

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

(no. of comparison)

Average case :

$$T(n) = \underset{\downarrow}{1}\cdot\frac{(n-1)}{2} + \underset{\downarrow}{2}\left(\frac{n-1}{2}\right) = \frac{3}{2}(n-1)$$

only first 'if' checked for half time

Both 'if' checked for half time

Best case $\leq$ Average case $\leq$ Worst case

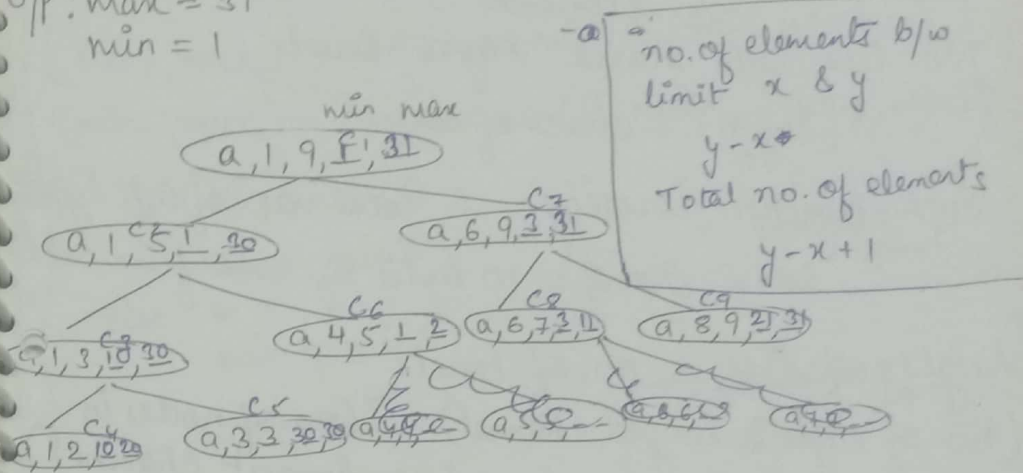
In all the cases here time complexity $= O(n)$

Space complexity = for fun + for main func
12 Bytes + 2 × n leyles
$2n + 12 = O(n)$

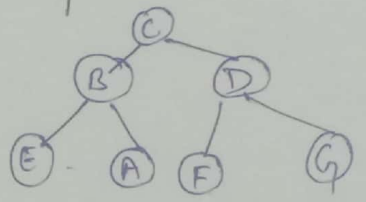
* With DAC

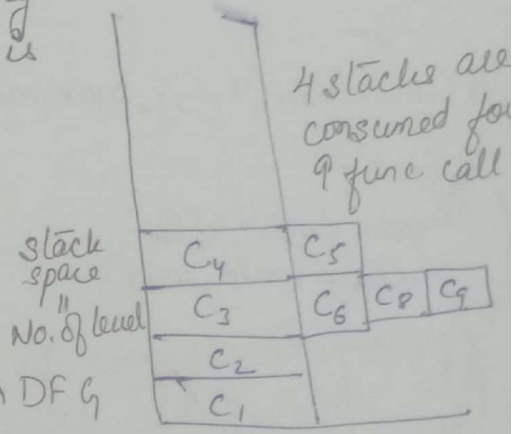
i/p : A[10, 20, 30, 1, 2, 3, 11, 31, 21]
1 2 3 4 5 6 7 8 9

o/p : max = 31
min = 1



In all programming lang func calling sequence is called pre-order.





calling : Preorder : CBEADFG

Completion : Postorder : EABFGDC

In every prog. lang. func. execution order is post order

Note :
No. of levels in function call = no. of stacks
In above ex → only 4 (i.e one stack for one level function

tree is formed by dividing no. of element by 2.
that means for 'n' element each element will take $n/2$ elements.
So. tree will have $\log n$ levels.

$O(\log_2 n)$ = space of stack

Each function is using = 5 local var. of int type
so = $2 \times 5 = 10$ bytes by each func call

* height of tree = no. of levels − 1
(no. of edges in longest channel) → (no. of nodes in longest channel)

```
DAC maxmin(a, i, j, max, min)
{
  if(i==j)
  {
    max = min = a[i]
    return (min, max);
  }
  if(i==j-1)
  {
    if(a[i] > a[j])
      max = a[i], min = a[j]
    else
      max = a[j], min = a[i]
    return (min, max);
  }
  else
  {
    mid = ⌊(i+j)/2⌋        } divide statement
    (min1, max1) = DAC maxmin(a, i, mid, max, min)
    (min2, max2) = DAC maxmin(a, mid+1, j, max, min)
    if (min1 > min2)
      min = min2
    else
      min = min1
    if (max1 > max2)
      max = max1
    else
      max = max2
    return (min, max);
  }
}
```

conquer stmts

combining stmts

Let $T(n)$ be the no. of comparisons required for the n-elements array. using above algo then, Recurrence Relation

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

for small

divide → 0 ; conquer stmts → $2T(n/2)$; combining → 2 (no. of subproblems)

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

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

$= 2\left(2\left(2T\left(\frac{n}{2^3}\right)+2\right)+2\right)+2$

$T(16) = 2T(8)+2$

$= 2(2T(4)+2)+2$

no. of level $= 2^3 T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2$

extra space $\leftarrow$ k $\rightarrow$ $= 2^k T\left(\frac{2n}{2^k}\right) + 2^k + 2^{k-1} + \ldots 2^3 + 2^2 + 2$

$= 2^{\log_2 n - 1} T\left(\frac{n}{2^{\log_2 n - 1}}\right) + 2 + 2^2 + 2^3 + \ldots 2^{\log_2 n - 1}$

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

$\frac{n}{2} \cdot 1 + \left[2\left(\frac{2^{\log_2 n - 1} - 1}{2-1}\right)\right]$

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

$\downarrow$ comparison + time also

Space consumed = no. of local variables * 2

10 * 2 = 20 Bytes

$n/2^k = 2$

$n = 2^{k+1}$

$\log n = k+1 \log 2$

$\log n + 1$

Total space $\Rightarrow$ i/p + extra stack byte

$2nB + \log nB$

$\Downarrow$

$2n \Rightarrow O(n)$

Let $T(n)$ be the comparison between the elements not between the position.

{ Best case time = Average case = Worst case time is above algo (coz in else part we cannot skip any of the stmt.)

space = n (array) + $\log n$ (stack extra space.)

$\Rightarrow O(n)$

Note:

Comparing straightmaxmin & DACmaxmin straightmaxmin is better bcoz both the times are $O(n)$ and DACmaxmin will take more space in the form of stack.

POWER OF AN ELEMENT

i/p : 2 +ve integers a & n where a>1, n>1

o/p : Return a^n

* Predefined func (or any thing) takes constant time while calculating execution time

Without DAC

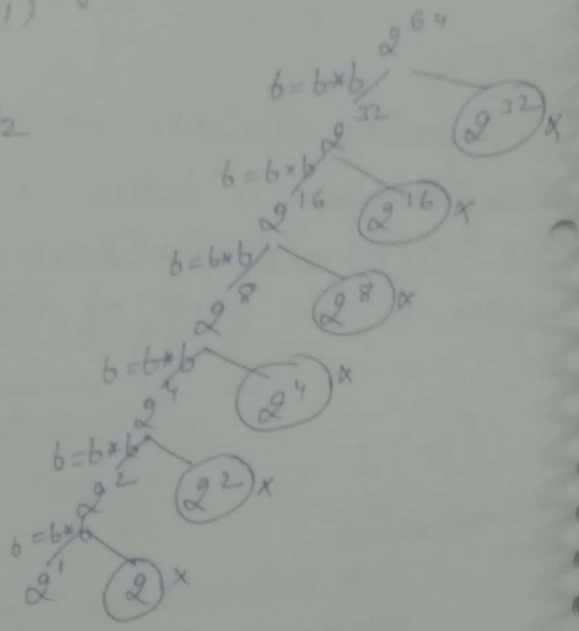
```
s=1;
for(i=1 to n)
  s=s*a;
```

Space consumed = 4 variable
= 4×2 Bytes = 8 Bytes
= O(1)

WITH DAC

$a^n = a^{n/2} \cdot a^{n/2}$

```
DACPOW(n, p)
{
  if(n==1)
    return(p)
  else
    mid = n/2;
    DACPOW(mid
```




```
Pow(a,n)
{
  if(n==1)
    return a;
  else
  {
    mid = n/2;
    pow=(a,mid);
    c=b*b
    return c;
  }
}
```

Let T(n) be the no of multiplications

Recurrence Relation :-

$$T(n) = \begin{cases} n=0 & \text{(small problems)} \\ T(n/2)+1 & \text{(big problems) if } n>1 \end{cases}$$

① $T(n) = T(n/2) + 1$

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

$= T(n/8) + 1 + 1 + 1$

To — k

$T(n/2^k) + 1 + 1 + \ldots$ k times 1

$T(n/2^{\log n}) + 1 + 1 + \ldots + 1$ k times last time

$= T(1) + \log_2 n + 1 \cdot k = 0 + \log_2 n$

$= \log_2 n = O(\log_2 n)$

$n/2^k = 1$

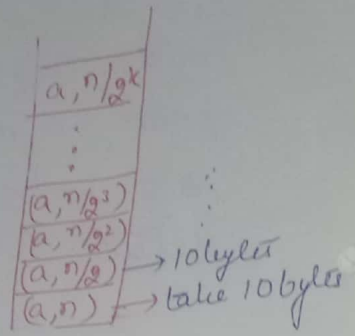
$n = 2^k$

$\log n = k \log 2$

$\log n = k$

In the else case before return statement there is no exit/break/return so we cannot skip the 3 stmts in else part. Because of this average = worst = best case are same.

Total Space = i/p + extra stack space
= 4Bytes + stack of $10\log n$
= $10 \log n$
= $O(\log n)$



If the power is not in the form of 2^x eg: 64, 128, etc we can make a sub function named "adjustment" where we can find the nearest 2^n of power of that value.

eg: 2^{100} ⇒ adjustment ; find 2^{64} $2^{64}/2^{36}$,

becoz
2^{100}
2^{50}
2^{25}
call adjustment

SEARCHING

* **Linear Search**

i/p: array of n element, element-x
o/p: position of x-element : if x is found
else
return (-1)

ex:
i/p: A[10, 20, 30, 1, 2, 3, 31, 21, 11] (1 2 3 4 5 6 7 8 9)

x = 31	x = 50
o/p: 7	-1

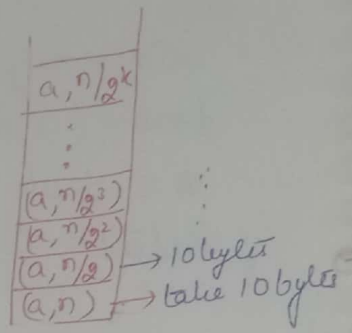
No. of Comparisons
Best case: x at first position
: O(1)
worst case: O(n)
Average case: $\frac{n(n+1)/2}{n} = \frac{(n+1)}{2}$

* **Binary Search**

	Graph	Tree
1)	Combination of (V, E)	Combination of (V, E)
2)	No root element.	Always root
3)	Connected / Disconnected	Always connected
4)	It is directed / undirected	Always directed
5)	May be cyclic / non cyclic.	It don't have cy

In the else case before return statement there is no exit/break/return so we cannot skip the 3 stmts in else part. Because of this average = worst = best case are same.

Total Space = i/p + extra stack space
= 4 Bytes + Stack of 10 log n
= 10 log n
= O(log n)



* If the power is not in the form of 2^x eg: 64, 128, etc we can make a sub function named "adjustment" where we can find the nearest 2^x of power of then value.

eg: 2^{100} ⇒ adjustment
? find 2^{64}
$2^{64}/2^{36}$

becoz

2^{100}
2^{50}
2^{25}
? call adjustment

SEARCHING

* Linear Search

i/p: array of n element, element-x
o/p: position of x-element : if x is found
else
return (-1)

ex:

i/p: A[10, 20, 30, 1, 2, 3, 31, 21, 11]
(1) (2) (3) (4) (5) (6) (7) (8) (9)

x = 31	x = 50
o/p: 7	-1

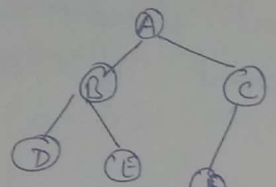
No. of Comparisons

Best case: x at first position
: O(1)
worst case: O(n)
Average case: $\frac{n(n+1)/2}{n} = \frac{(n+1)}{2}$

* Binary Search

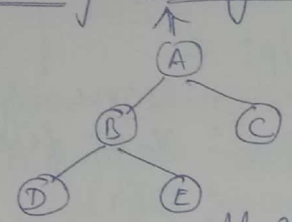
	Graph	Tree
1).	Combination of (V, E)	Combination of (V, E)
2).	No root element.	Always root
3).	Connected / Disconnected	Always connected
4).	It is directed / undirected	Always directed
5).	May be cyclic / non cyclic.	It don't have cycl

* In the given tree, everyone has no more than 2 children. So it is a binary tree.



* No node / vertices ⇒ empty binary tree.

* In a binary tree if it can have 0 or 2 or 0 child only then it is strictly binary tree.



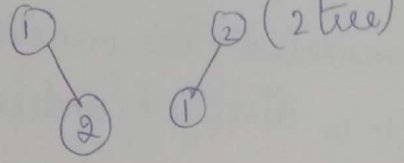
Binary Search Tree

In the given binary tree comparing root data all the element present on left hand side is smaller and all element present on right side are greater people call it BST.

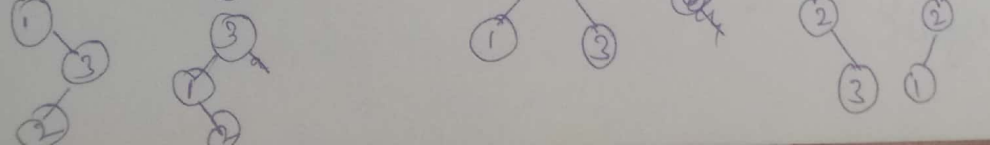
1) How many BST possible with n distinct nodes

(a) $n=1$ ⇒ 1 tree

(b) $n=2$ ⇒ (1) (2)

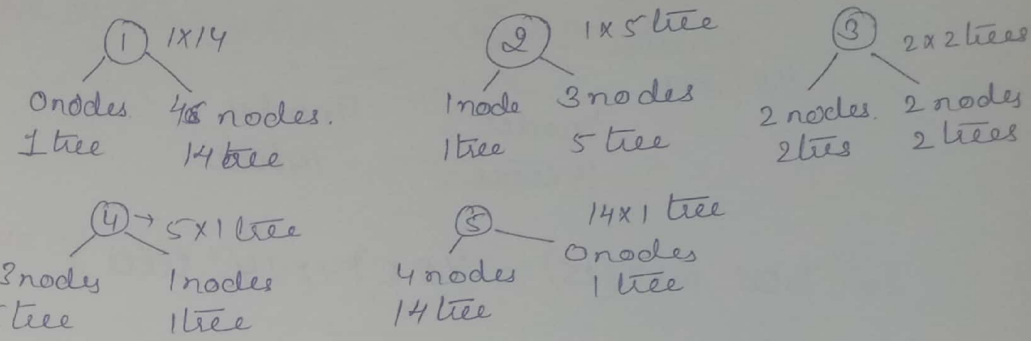


(c) $n=3$ ⇒ (1,2,3) ⇒ 5 tree



(d) $n=4$ (1,2,3,4)

(3) $n=5$ (1,2,3,4,5) BST?



$14+5+4+5+14 = 42$ BST

no. of BST with n nodes $= BST(n) = \sum_{i=1}^{n} BST(i-1).BST(n$

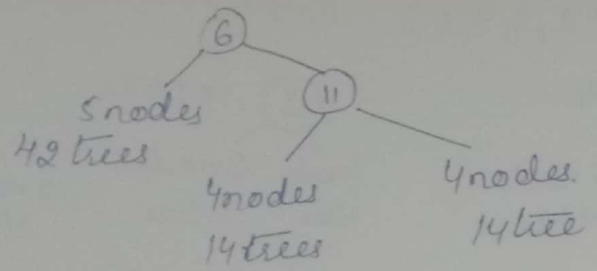
(we will assume i as root node each time we calculate eg: node 1 as root

$BST(0).BST(n-0)$

Or

$= BST(\text{left side nodes}).BST(\text{Right side nodes})$

? How many BST with nodes 15 and labels (1,2,3,4,5 ... 15) in such a way that root node is 6 and right hand side root node is 11



So Total BST(15) = 42 × 14 × 14 trees.

No. of Binary tree = No. of BST × n!

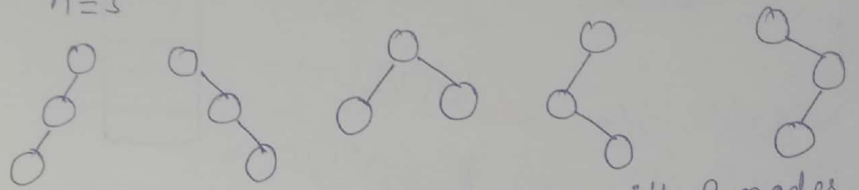
$$= \frac{{}^{2n}C_n}{n+1} \times n!$$

$$\text{No. of BST}(n) = \frac{{}^{2n}C_n}{n+1}$$

$\frac{BT}{2n}$

Q. How many unlabelled binary tree possible with n nodes?

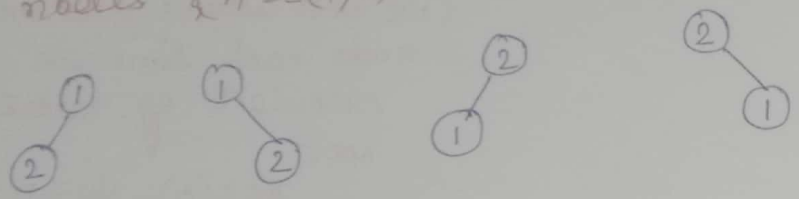
n = 3



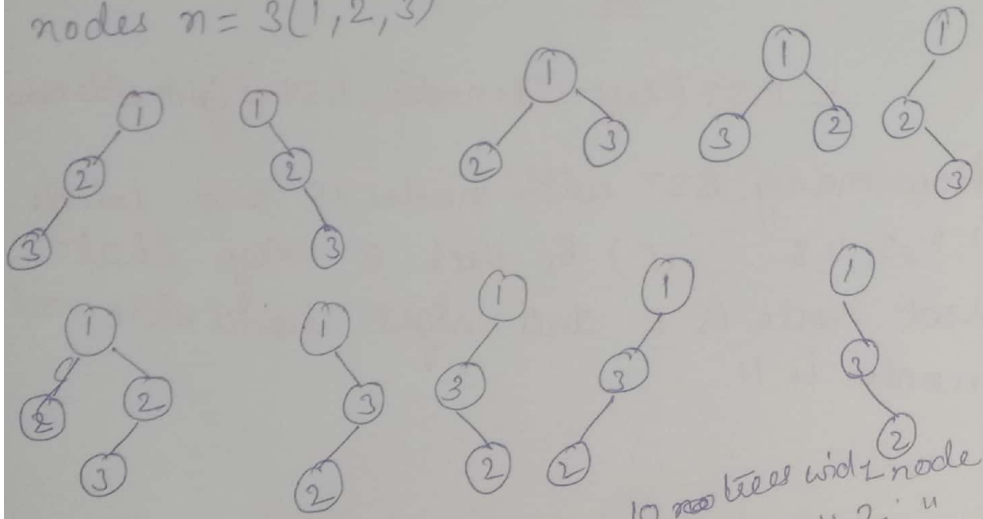
5 unlabelled BT with 3 nodes.
& shape different

Q How many Binary tree possible with n nodes {n = 2(1,2)

4 trees



nodes n = 3(1,2,3)



= 10 trees with 1 node as root
10 " " 2 " " "
10 " " " " "
30

no. of BST(n) = no. of unlabelled Binary tree(n)

Ques i/p: A set with n-element and an unlabeled BT with n node

o/p: How many ways you can insert lables into unlabeled BT so that it will become BST

(a) 0 (b) 1 (c) n! (d) $\frac{{}^{2n}C_n}{n+1}$

(if not mentioned then 'b' answer should be answer)

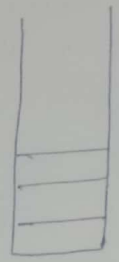
unlabelled BT & BST are same in diagram.

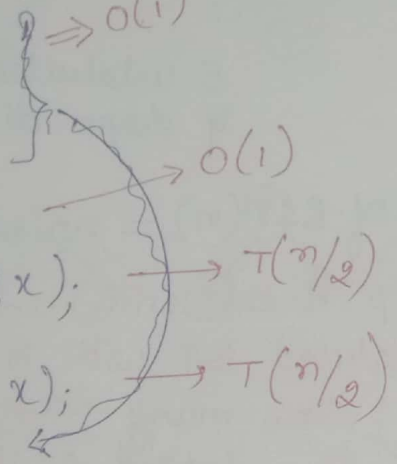
Binary Search Algo

i/p: Sorted array of n elements, element - x
o/p: Position of x-element

ex: i/p: A[10, 20, 30, 40, 50, 60, 70] x = 40

```
BS(a, i, j, x)
{
  if(i==j)                          } constant time
    if(a[i]==x)                     } O(1)
      return(i)                     }
    else                            }
      return(-1)                    }
}
else
{ mid=(i+j)/2.;          ⇒ O(1)
  if(a[mid]==x)          ⇒ O(1)
  {
    return mid;
  }
else
  if(a[mid]>x)
    BS(a, i, mid-1, x);
  else
    BS(a, mid+1, j, x);
}
```





Let T(n) be the time complexity on n element using above algo. Then

recurrence Relation, T(n)

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

In binary search there is divide, conquer but no combine that why it is also called partial divide and conquer algo.

$$T(n) = T(n/2) + O(1)$$
$$= T(n/2) + C$$
$$= T(n/2^2) + C + C$$
$$= T(n/2^3) + C + C + C.$$
$$\vdots$$
$$= T(n/2^k) + \log n\, C.$$
$$= T(n/2^{\log n}) + \log n\, C.$$
$$= T(1) + \log n\, C$$
$$O(1) + C.\log_2 n = C.\log_2 n$$
$$= O(\log n)$$

for last level

$n/2^k = 1$
$n = 2^k$
$\log n = k$

Worst case
Average case

for best case

$$T(n) = O(1)$$

Total Space = i/p + extra space

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

Sum of

Average case;

$$(1 + 2 + 3 + \ldots + \log n)$$

$$\log n \frac{(\log_n n + 1)}{2} = \frac{\log n + 1}{2}$$

$$= O(\log n)$$

In case of linear search will take more space in comparison to binary searching

If the array would not be sorted $= O(n \log n)$

Q// If: A sorted array of n-distinct element find any ele in A[i] such that a[i] = i

(Best algo, worst case time.

A[-50, -40, -30, -20, -10, -2, 0, 3, 6, 8, 11, 15, 20, 44,
0 1 2 3 4 5 6 7 8 9 10 11 12 13

55, 90, 100, 150, 250]
14 15 16 17 18

① Linear Search is $O(n)$ LS ⇒ OS

② BS is possible. P > V go right
else go left.

Ques i/p: An array of n-elements until some place all are integers and afterwords all are infinite

o/p: find position of 1st infinite. i.e 19

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
A[	50	5	26	37	-55	-42	0	76	94	16	-5	-7	20	46	29	90	100	150	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞]

mid = $\frac{33+1}{2} = \frac{34}{2} = 17$

A[17] = 100 ⇒ move right

mid = $\frac{18+33}{2} = \frac{51}{2} = 25$.

A[25] = ∞ ⇒ not 1st infinite move left → A[24] = ∞ so

mid = $\frac{18+24}{2} = \frac{42}{2} = 21$ A[20] = ∞ so

A[21] = ∞ ⇒ not 1st infinite move left

mid = $\frac{18+20}{2} = \frac{38}{2} = 19$

A[19] = ∞ and A[18] = 150 (integer)

So 19 is answer

② i/p: An array of n-ele where until some place all are integers and after words all are infinite (assume n is unknown & after our array all are ∞)

o/p: find position of 1st infinite.

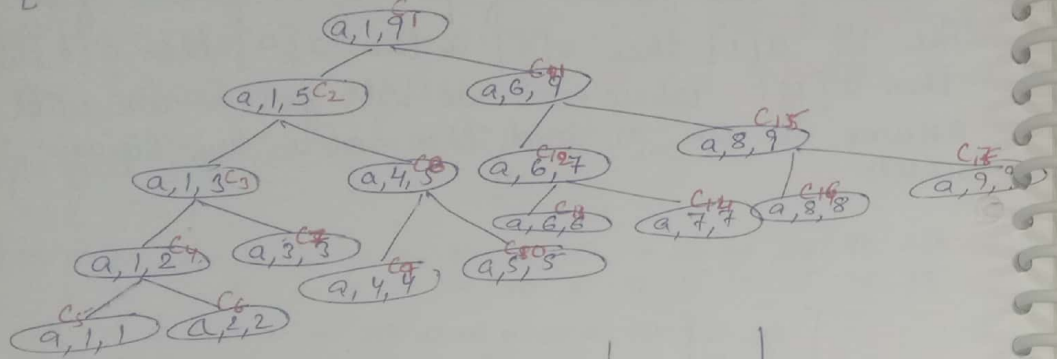
① checking the values of position in power of 2 like 1st a[1] then a[2] then a[4] then a[8] then a[16] where ever we will get ∞ we will assume it as n. and then solve by Binary search

Merge Sort

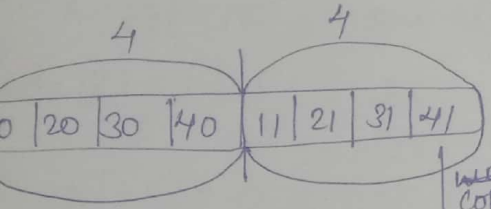
Note: Merging 2 sorted subarrays.

ex:

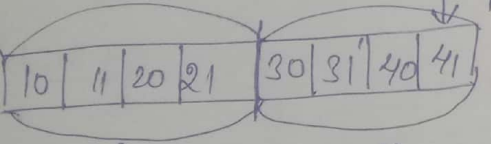
	1	2	3	4	5	6	7	8	9
A[	25	68	16	42	15	91	77	19	20]

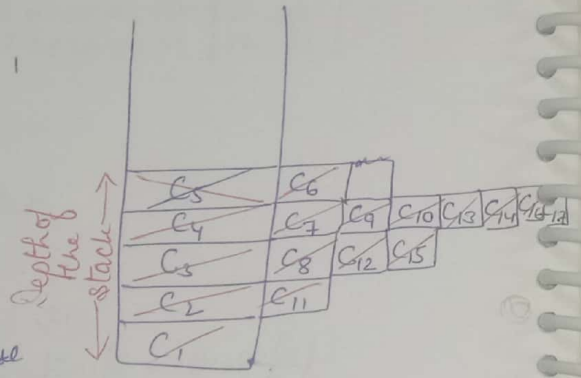


Merge Algorithm



move copy





(10, 11) = 10
(20, 11) = 11
(20, 21) = 20
(30, 21) = 21
(30, 31) = 30
(40, 31) = 31
(40, 41) = 40

comparisons.

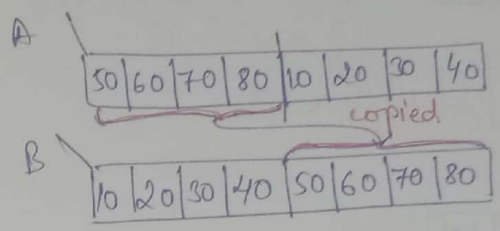
$4+4-1$
$n/2+n/2-1$
$m+n-1$

moves

$4+4$
$n/2+n/2$
$m+n$

whichever is more b/w comparisons & moves. the time complexity = $O(n)$ or $O(m+n)$

Merge Algo best case



(50, 10) = 10
(50, 20) = 20
(50, 30) = 30
(50, 40) = 40

Moves

$4+4$
$n/2+n/2$
$m+n$

comparisons.

$4, 4 \Rightarrow 4$
$n/2, n/2 \Rightarrow n/2$
$m, n \Rightarrow$ whichever is smaller $\min(m,n)$

Time complexity = moves + comparison
= but we will take whichever is larger
= in above case: moves. $(m+n)$
$O(n) = O(m+$

ex:

A (500 elements sorted) B (600 elements sorted)

worst case comparision = $m+n-1 = 500+600-1 = 1099$
best case = 500
movement in best case / worst case = $500+600 = 11$

* Merging two sorted sub array of size m & n req $O(m+n)$
* Because the merging take place in some other algo that's why it is called out-place m algo. (not-in-place)

Time complexity of out-place merge algo = $O(n)$

" " " in-place " " = $O(n^2)$

MergeAlgo [worstcase, Inplace]

A

50	60	70	80	10	20	30	40

(50,10) => 10 swap(50,10)

50	20	30	40

unsorted

so (50,20) = 20 swap

20	50	30	40

unsorted

(50,30) = 30 swap

20	30	50	40

(50,40) = 40 swap

20	30	40	50

so

$n/2$ swaping for $n/2$ times $n/2$

$n/2 \cdot n/2 = n^2/4$

$= O(n^2)$

Merge_Sort (a, i, j)

```
{
  if(i==j)
    return(a[i]);          } O(1)
  else
  {
    mid = [(i+j)/2]        } O(1)
    merge_sort(a, i, mid);   } 2T(n/2)
    merge_sort(a, mid+1, j);
    merge(a, i, mid, mid+1, j);  } O(
    return(a);
  }
}
Merge(
```

Recursion Recurrence Relation (merge_sort_out-place)

Let $T(n)$ be the time complexity of merge sort

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

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

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

$$= 2\left[2\left(2T(n/2^3) + n\right) + n\right] + n$$

$$= 2^3 T(n/2^3) + 3n$$

$$\downarrow k$$

$$2^k T(n/2^k) + k.n$$

$$2^{\log n} T(n/2^{\log n}) + n.\log n$$

$$2^{\log n} T(1) + n.\log n$$

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

$n/2^k = 1$

$n = 2^k$

$\log n = k$

If a sorting algo takes more than $\log n$ extra space then it is out-place algo

vice versa

max of $\log n$ space then in-place

imp Ques

Total space = i/p + extra

n (array) + n (merge) + $\log n$ (stack)

$= O(n)$

Mergesort_In-place

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

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

decreasing g.p series summation = 1

$$= O(n^2)$$

($O(n\log n)$ = Best case = Worst case = Average case because in 'else' part of merge-sort algo there are no r/n, exit / break (we cannot minimize any stmt there whichever sequence of i/p came))

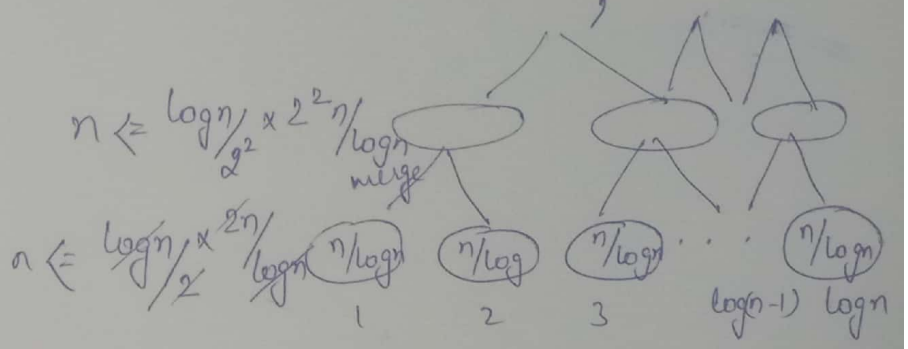
i/p : $\log n$ sorted subarrays each of size $n/\log n$

o/p : find one sorted subarray of size n

$\downarrow\downarrow$

Best algo worst case

$n = \log n/2^k * 2^k n/\log n$



$n \leq \log n/2^2 \times 2^2 n/\log n$ merge

$n \leq \log n/2 \times 2n/\log n$

$\frac{\log n}{2^k} = 1$

$\log n = 2^k$

$\log\log n = k$

$n * k \Rightarrow n \times \log\log n$

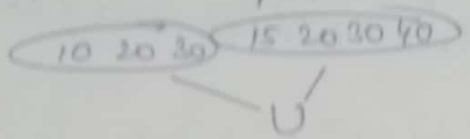
because grouping each time of base of log would be 2

if we group 3 at a time base of log would be 3.

we assume
Ques k subarrays | each subarray (size of subarray) n^2/k

Ans $n^2 . \log k$ (discuss how?)

Ques i/p: 2 sorted subarray A contain m elements and B contain n elements (apply merge algo)
o/p: Find A ∪ B & A ∩ B.
In union no repeatation



10, 15, 20, 30, 40

* Tips
* Any problem 2 sorted / 5 sorted / logn sorted subarray
any no. of sorted subarray ↓ more than 1 use merge sort

* "One sorted array" → B

```
swap(a[i],a[j])
}
}
swap(a[p],a[i])
return(i);
}
```

i is going to execute (n-1) times both in (best & worst case)

29 42 10 19

29 42 65 15 32 91 10 75 19 32
1 2 3 4 5 6 7 8 9 10

26 | 15 10 19 | 29 | 91 65 75 42 32 |
smaller greater

(26 15 10 19) 29 (91 65 75 42 32)
1 2 3 4 5 6 7 8 9 10
i

eg → (2)

55 26 99 42 23 79 56 21 99
1 2 3 4 5 6 7 8 9
i j i j

21 26
21 23 99 26 55 79 56 42 99
i

21 23 29 26 55 79 56 42 99

21 20 42 23 55 79 56 99 99

(n-1) times (both in best and worst case)

So time complexity of partition = $o(n)$

total no of swaps = max n (worst case)
min swap = 1 (best case)

Ex (3)

60 21 35 15 42 10 20 90 70 80

i j j j j j i i i

i i i i i j

60

20 60 21 35 15 42 10 20 90 70 80

(20 21 35 15 42 10) 60 (90 70 80)

smaller greater

Quicksort Algorithm

```
Quicksort(a, p, q)
{
  if(p==q)
    return(a[p]);
  else
  {
    m = partition(a, p, q)
    quicksort(a, p, m-1)
    quicksort(a, m+1, q)
    return a);
  }
}
```

P.R (let T(n) be the time complexity of quick sort algo of n element)

$$T(n) = \begin{cases} & \\ & \end{cases}$$

then R.Recurrence Relation

$$T(n) = \begin{cases} o(1) & \text{if } (p=q) \\ o(n) + T(n/2) + T(n/2) + 0 & \text{else} \end{cases}$$

↳ (when elements are dividing into 2 equal part)

In case of balanced partition

$$T(n) = 2T(n/2) + n$$ (only one case)

$$2\left[2T\left(n/2^2\right)\right] + n/2 + n$$

$$2^2 T\left(n/2^2\right) + \frac{2n}{2} + n$$

$$2^2\left[2T\left(n/2^3\right)\right] + n/2^2 + \frac{n}{2} + n$$

$$2^3 T\left(n/2^3\right) + n/2^2 + n/2 + n$$

⋮ k times

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

$n = 2^k$

$k = \log_2 n$

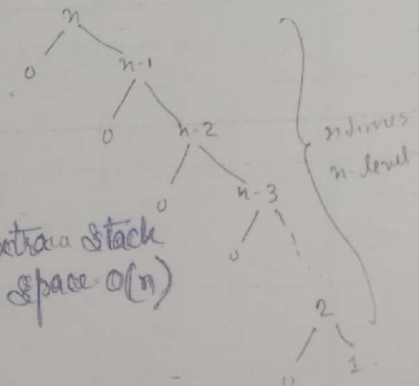
$= \log n\, T(1) + n\left[\frac{1}{2^0} + \frac{1}{2^1} + \cdots + \frac{1}{2^{k-1}} + \frac{1}{2^k}\right]$

$= \log_2(n) + n \log n$

generalised case

$T(n) \Rightarrow O(n) + T(m-p) + T(q-p) + 0$ if $(n>1)$

best case (balanced partition)	worst case (unbalanced partition)
$T(n) = n + 2T(n/2)$	$T(n) = n + T(0) + T(n-1)$
$= O(n \log_2 n)$	$n + T(n-1)$
no. of levels $= \log_2 n$	$= T(n-1) + n$
each costs $= O(n)$	$= O(n^2)$
extra space $(\log n)$	



if pivot element position keeps on changing then it is called randomized quicksort

Randomized Quicksort

quicksort → best → $O(n \log n)$
worst → $O(n^2)$

extra space can be reduced to from n to $\log n$ by writing better algo i.e. $O(\log n)$

Average case

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

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

$2T(n/2) + 2n \Rightarrow 2n \log n$

$= O(n \log n)$ {which is nothing but best case only}

[Avg case and best case are same]

in case unlucky case first then

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

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

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

$= 2n \log n$

$O(n \log n)$

Avg case of quicksort is same as that of best case of quicksort.

quicksort is applied on following 2 i/p

increasing order

i) 1,2,3,4,5 ..... n $O(n^2)$ swaps.

decreasing order

ii) n, n-1, n-2, n-3, ... 2,1 $O(n^2)$ swaps

Let C_1, C_2 be the two comparison made for the i/p 1 & 2 respectively then what will be the relation between C_1 & C_2

i) $C_1 < C_2$.

ii) $C_1 > C_2$.

iii) $C_1 = C_2$

iv) non-comparable.

(ii) if array is in increasing order then that is the worst case of quicksort

let say.

10, 20, 30, 40, 50, 60, 70

(1) 10 (20, 30, 40, 50, 60, 70) → 6-1

20 (30, 40, 50, 60, 70) → 5-1

→ 4-1

$T(7) = 6 + T(6)$

$6 + 5 + T(5)$

$4 + T(4)$

$3 + T(3)$

$2 + T(2)$

$1 + (T(1)$

$n + T(0)$

$= 1 + 2 + 3 + \dots + n$

$\frac{n(n+1)}{2}$

$O(n^2)$

Swaps $= n-1$,

(ii) for decreasing series

10 60 50 40 30 20 70

70 60 50 40 30 20 10

$\Rightarrow n$ swaps.

P(7) ⇒ 7 =

(10, 20, 30, 40, 50, 60) 70 (0)

6-1 ⟶ n swaps

(10) (20, 30, 40, 50, 60)

P

5-1 ⟶ 1 swaps

(20, 30, 40, 50) (60) →

4-1 ⟶ n swaps

(1) 20 (30, 40, 50)

3-1 = ⟶ 1 swaps

(30, 40) (50) (1)

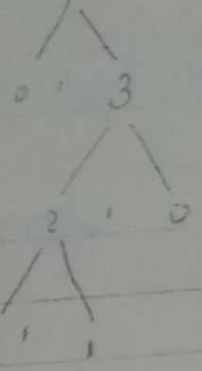
1

1 swaps

total no of swaps

$= n + 1 + n + 1 \dots + 1$

$= \frac{n}{2} * n \rightarrow O(n^2)$



So for $c_1 = n$
$c_2 = n^2$

So $c_2 > c_1$

If all elements are equal

~~10_f~~ 10_b 10_c 10_d 10_e ~~10_f~~ 10_a
~~10_a~~, ~~10_b~~, ~~10_c~~, ~~10_d~~, 10_e, ~~10_f~~
1 2 3 4 5 6

10_f 10_b 10_c 10_d 10_e 10_a. ( )

⇓

always right hand side is gap.

comparison = n^2
swaps = n^2

8- In case of sorted

if the array is

case

4- quick sort is bet

not

In the

quick sort

sort is

Recursive tree

$T(n) = n + T($

$\log_2 n$

$n/2$

$n/2^2$ $n/2^2$

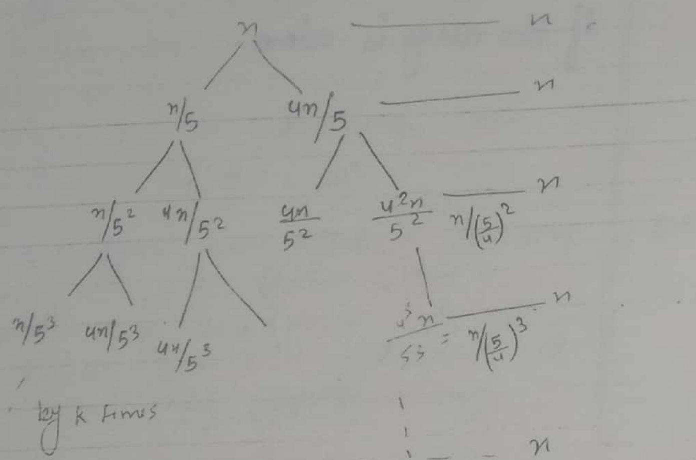
k time

II $T(n) = n + T(n/5) + T(4n/5)$

$n + \frac{n}{5} + T(n/5^2) + \frac{4n}{5} + T\left(\left(\frac{4}{5}\right)^2 n\right)$

$n + \frac{n}{5} + \frac{n}{5^2} + T\left(\frac{n}{5^3}\right) + \left(\frac{4}{5}\right)^2 n + T\left(\left(\frac{4}{5}\right)^3 n\right)$

$n + n + T\left(\frac{n}{5^3}\right) + T$



$n/(5/4)^k$ ⇒ this will have more no. of levels

no. of levels $= \log_5 n$

no. of levels $= \log_{5/4} n$

Boz. $\frac{n}{(5/4)^k} > \frac{n}{5^k}$

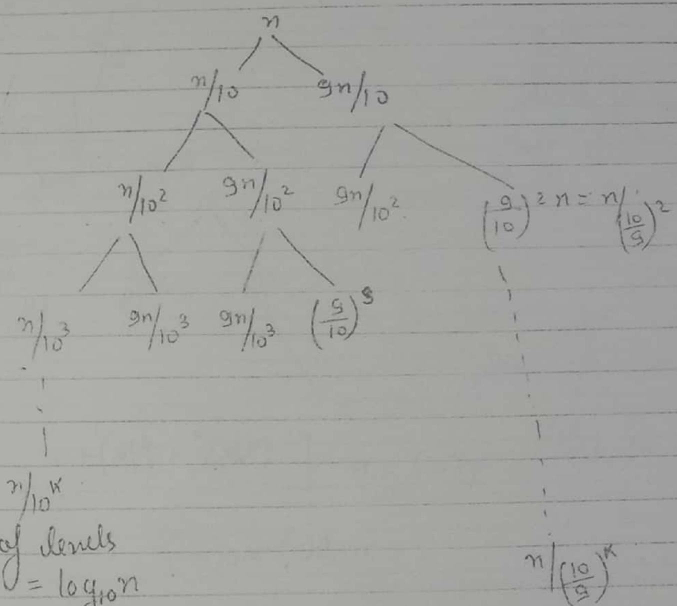
$n\log_5 n \le T(n) \le n\log_{5/4} n$

$T(n) = O(n\log_{5/4} n)$

$T(n) = \Omega(n\log_5 n)$

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

3⇒ $T(n) = n + T(n/10) + T(9n/10)$



no. of levels $= \log_{10} n$

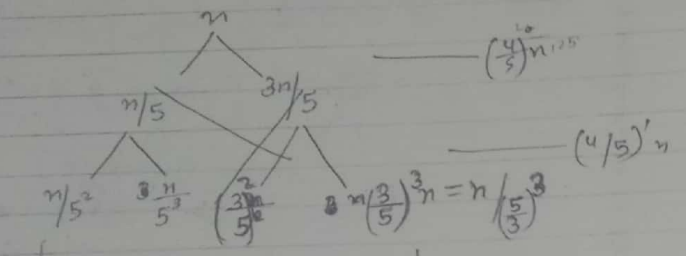
no. of levels $= \frac{n}{(10/9)^k} = 1$

$n = \left(\frac{10}{9}\right)^k$

$k = \log_{10/9} n$

IV

$T(n) = n + T(n/5) + T(3n/5) \Rightarrow O(n)$



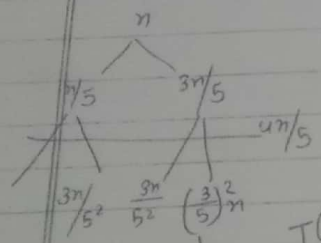
$(4/5)^0 n$

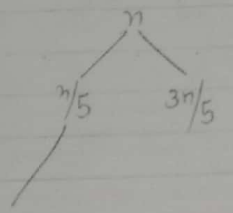
$(4/5)^1 n$

$(4/5)^{\log_5 n}$

$n/5^k$

no of levels
$k = \log_5 n$





$(4/5)^{\log_{5/3} n}$

$T(n) = n \left[(4/5)^0 + (4/5)^1 + \cdots (4/5)^{\log_{5/3} n} \right]$

$= n\, O(1) = O(n)$

$\frac{n}{(5/3)^k}$

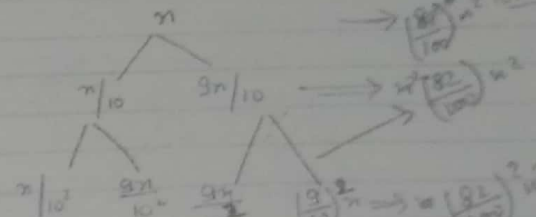
here $k = \log_{5/3} n$

??

V

$T(n) = n^2 + T(n/10) + T(9n/10)$

Q2.



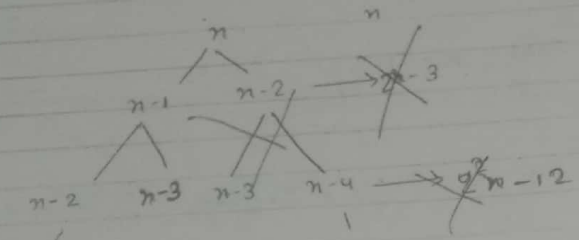
$n/10^k$

no of levels
$k = \log_{10} n$

$n/(10/9)^k$

no of levels
$k = \log_{10/9} n$

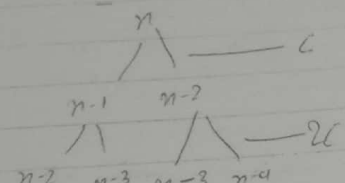
VI) $T(n) = C + T(n-1) + T(n-2)$



no. of levels $K = n$

$2^K n$ - (same constant)

no. of levels $K = \log n$



no. of levels $K = n$

no. of levels =

$T(n) = 2^0 C + 2^1 C + \cdots + 2^n \cdot C$

$C[2^0 + 2^1 + \cdots + 2^n)$

$= 2^0 \left\{ \frac{2^n - 1}{2-1} \right\} = O(2^n)$

$T(n) = T(n/2) + T(n/3) + T(n/5) + C \Rightarrow O(3^n)$

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

$T(n) = n + T(n/5) + T\left(\frac{4n}{5}\right)$

$T(n) = n + T(n/100) + T\left(\frac{99n}{100}\right)$

$T(n) = n + T(\alpha \cdot n) + T((1-\alpha) \cdot n)$] generalized form

$0 < \alpha < 1$ (for best & average case)

then stack on no. of levels

$= \log_{\frac{1}{\alpha}} n$

$= \log_2 n$

stack space

$\log_{(\frac{1}{1-\alpha})} n$

let suppose $\alpha = \frac{1}{5}$

then $\frac{1}{\alpha} = 5$

so $\log_5 n$

So stack size $= \max\left(\log_{\frac{1}{\alpha}} n, \log_{\frac{1}{1-\alpha}} n\right)$

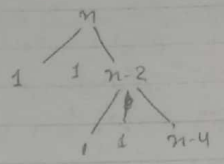
Time : $O(n \log n)$

base doesn't matter.

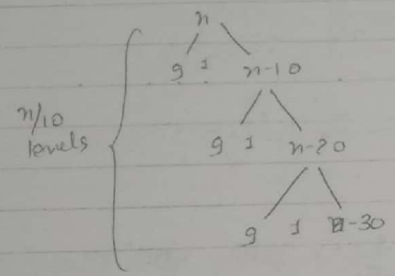
Worst_case

$T(n) = n + T(0) + T(n-1) \Rightarrow n * n \rightarrow O(n^2)$

$T(n) = n + T(1) + T(n-2) \Rightarrow \frac{n}{2}(\text{levels}) * n = O(n^2)$



$T(n) = n + T(9) + T(n-10) \Rightarrow \frac{n}{10} * n \Rightarrow O(n^2)$



$T(n) = n + T(c-1) + T(n-c) = \frac{n}{c} * n = O(n^2)$

constant stack space

n/c levels

n/c

O(n) — extra space

using better algo it can be reduced to $\log_2 n$

where c is a constant & & $c \geq 1$

Prob In quick sort the sorting of n numbers, the $n/5^{th}$ smallest element is selected as pivot using $O(n)$ time complexity Algorithm; then what will be the best case time complexity of quicksort

i) $O(n^2)$
ii) $O(n \log n)$
iii) $O(n^3)$
iv) $O(n)$

$T(n) = O(n)$ (for pivot) $+ O(n)$ (partition) $+ T\left(\frac{n}{5} - 1\right) + T\left(n - \frac{n}{5}\right)$

$= 2n + T(n/5) + T\left(\frac{4n}{5}\right)$

$= O(n \log n)$

Prob In quick sort the sorting of n numbers the $n/5^{th}$ smallest element is selected as pivot using $O(n^2)$ time complexity algo, then what will be the worst case time complexity

$\Rightarrow O(n^2)$

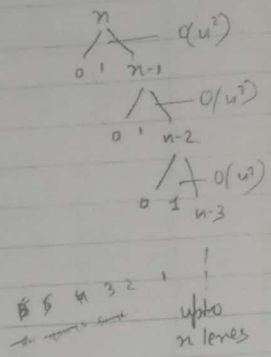
In quick sort the sorting of n numbers, the $n/10^{th}$ element is selected as pivot using $O(n^2)$ time complexity algo, then what will be the worst case time complexity of algo

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

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

$= \frac{n(n+1)(2n+1)}{6}$

$= O(n^3)$



In quick-sort the sorting of n no., the $n/5^{th}$ largest element is selected as pivot using $O(\log n)$ time complexity algo, Then what will be the the best case time complexity

$T(n) = O(\log n) + O(n) + T(n - n/5) + T(n/5)$

$\Rightarrow O(n \log n)$

Median = $\frac{n}{2}$ smallest element in sorted array

Randomized Quick-sort

ex 10 40 20 30 10 50 60 70

$i = RG_1(1,7)$

swap(1,4)

(20 30 10) 40 (70 60 50)

$i = RG_1(1,3)$ $\quad$ $i = RG_1(5,7)$

swap(1,1) $\quad$ swap(5,7)

10 (20) 30 $\quad$ (60,50) 70 ( )

(nothing but best, avg case)

Randomized Quick-sort

choosing pivot-element randomly is called randomized Quick-sort (in case of sorted array)

```
RQS (a, P, q)
{
  if (p == q)
    return (a[p]);
  else
  {
    r = RG1(P, q)
    swap(a[P], a[r])
  }
  m = Partition(a, P, q);
  RQS (a, P, m-1);
  RQS (a, m+1, q);
}
  return(a);
```

in randomized Quick-sort all three complexities are same i.e. $O(n \log n)$

note | for some cases randomized quick-sort will give worst-case performance of $O(n^2)$ these are

i) All elements are same
ii) if array is not sorted then also worst case performance will come.

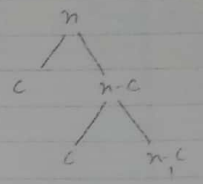
i) 7 ⇒ 1st smallest element
ii) 4 ⇒ 2nd smallest no.
iii) 1 ⇒ 7th.

60	20	50	30	10	90	80
1	2	3	4	5	6	7

suppose random no. generated = 5

at 5th partition no. = 10
so again left hand side no element & right hand side n-1
so this is also worst case.

```
QS(a,P,q)
{
  while(P<q)
  {
    m = partition(a,P,q)
    if(m-P > q-m)
    {
      QS(a,m+1,q);
      q=m-1
    }
    else
    {
      QS = (a,P,m-1)
      P=m+1
    }
  }
}
```



stack
extra space = max min
$O(\log n)$ $O(1)$

after partition call recursion on smaller size side & on larger side use while loop. while loop, do not take stack space, it will take only constant time

In array random access is possible

Selection procedure

i/p an array of n-elements and integer K
o/p return K^{th} smallest element

Algo(1)
if array is sorted and sorting ordering — 1
return(a[K]) $\Longrightarrow O(1)$

Algo(2)
i) if array not sorted
apply K passes selection sort $\Rightarrow O(Kn)$ } $O(Kn)$
ii) return (a[K]) $\Rightarrow O(1)$

diff b/w selection & bubble sort
⇓ ⇓
for min element for max element (at the end of the array)

after 1st pass of bubble sort, it gives at 1st largest element at the end of array while for min element apply selection procedure.

With divide & conquer

	1	2	3	4	5	6	7	8	9	10
A[	25	92	15	10	65	88	21	19	14	40]

~~K=6~~ if K=3.
apply partition algo

m = (14 15 10 21 19) 25 (92 65 88 40)
1 2 3 4 5 6 7 8 9 10

⇓
(2) m = (10) 14 (15, 21, 19)
⇓
(10) 15 (21,19)
3 4 5
return [a[K]]

```
{
  if (p==q)
    return
  else
  {
    m = partition (a, p, q)
```

→ to find kth smallest element.

```
Selection (a, p, q, k)
{
  if (p == q)
    return a[p]
  else
  {
    m = partition(a, p, q)        ⟶ O(n)
    if (m == k)
      return a[k]                 ⟶ O(1)
    else if (k < m)               ⟶ O(1)
      to selection(a, p, m-1, k)  ⟶ O(m-p)
    else
      selection(a, m+1, q, k)     ⟶ O(q-m)
  }
}
```

Let $T(n)$ be the amount of time required for the selection algorithm on n elements. Then recurrance relation.

$$T(n) = \begin{cases} O(1) & \text{if } (p == q) \\ O(n) & \text{if } (m == k) \\ O(n) + T(n/2) & \text{else} \end{cases}$$

$$T(n) = \begin{cases} \Theta(n) & \text{if } (n=1) \\ O(n) + O(1) + O(1) + \Theta(m-p) & \text{if } n > 1. \end{cases}$$

$O(n)$ or $O(q-m)$

best case

$\Theta(n) + T(n/2)$

$\Rightarrow O(n)$

worst case

$\Rightarrow O(n) + T(n-1)$

$\Rightarrow O(n^2)$

3 ways to get kth smallest element:

⇒ partition
⇒ selection procedure ← (best soln)
⇒ first sort then return kth smallest

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

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

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

$T\left(\frac{n}{2^k}\right) + n\left[\frac{1}{2^0} + \frac{1}{2^1} + \cdots + \frac{1}{2^{\log_2 n}}\right]$

$\Rightarrow O(n)$

This is also avg case

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

$= n + n-1 + n-2 + \cdots + 1$

$= \Theta(n^2)$

Stack space = $O(n)$ ⇓ n-levels.

Stack space = $O(\log n)$ ⇓ $\log n$ extra space

Stack space = $O(n)$

Strassen's matrix multiplication

without DAC

matrix-addition ⇓

$A_{m \times n}$, B_{mn}

$C_{m \times n} = A + B$

C contain mn elements ⇓ ⇓

$mn * 1$ ⟶ addition ⇓

$mn * O(1)$ ⇓

$O(mn) \Rightarrow O(n^2) \Rightarrow$ if $(m=n)$

- Space = O(mn)
Extra space = O(mn) (for C matrix)

matrix multiplication (without divide & conquer)

Condition

$A_{m \times n}$, $B_{n \times p}$

$C_{m \times p} = A * B$

C contains mp elements

So no. of multiplications = $mp * n$

$= mnp * O(1)$

$= mnp = O(n^3)$ if $(m = n = p)$

Algo for multiplication

```
for (i=1 to n)
  for (j=1 to n)
    for (k=1 to n)
{
  c[i][j] = c[i][j] + a[i][j] * b[k][j];
}
```

using DAC

matrix multiplication

With divide & conquer, matrix multiplication

1- Matrices sizes are $\leq 2 * 2$
Small problem

2- Matrices sizes should be of power of 2

3- Matrices should be square matrices

ex

$$A = \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right] \quad B = \left[\begin{array}{cc|cc} a & b & c & d \\ e & f & g & h \\ \hline i & j & k & l \\ m & n & o & p \end{array}\right]$$

A_{11} A_{12} A_{21} A_{22} B_{11} B_{12} B_{21} B_{22}

$\frac{4 \times 4}{2} = 2 \times 2$ $\frac{4 \times 4}{2} = 2 \times 2$

$$C = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]$$

$C_{11} = A_{11} * B_{11} + A_{12} B_{21}$

$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$

$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$

$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$

1 → matrix 4×4

⇓

8 → matrix 2×2

+

4 → additions 2×2

1 → MM → 64×64

⇓

8 → MM → 32×32

⇓

4 → MM → 16×16

⇒ 1 → MM → $n \times n$

⇓

8 → MM → $\frac{n}{2} \times \frac{n}{2}$

+

4 → additions → $\frac{n}{2} \times \frac{n}{2}$

Let $T(n)$ be the amount of time required to multiply 2 matrices of size $n \times n$

Then recurrence relation

$$T(4) = 8T(4/2) + 4 * (4/2)^2$$
$$T(64) = 8(64/2) + 4 * \left(\frac{64}{2}\right)^2$$

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

$$T(n) = \begin{cases} O(1) & \text{if } n \le 2 \times 2 \\ 8T(n/2) + 4 * (n/2)^2 & \text{if } n > 2 \times 2 \end{cases}$$

⇓

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

$$= 8\left[8T(n/2^2) + \left(\frac{n}{2^2}\right)^2\right] + n^2$$

$$= 8^2 T(n/2^2) + \frac{n^2}{2} + n^2$$

$$= 8^2\left[8T(n/2^3) + \left(\frac{n}{2^2}\right)^2\right] + \frac{2n^2}{2} + n^2 \Rightarrow 8^3 T(n/2^3) + n^2 + 2n^2 + \dots$$

$$= 8^3\left[T(n/2^3)\right] + n^2 + \frac{n^2}{2} + n^2 - \dots$$

k times

$$8^k T(n/2^k) + 2^2 n^2 + 2n^2 + 2^0 n^2$$

$$\frac{n}{2^k} = 2$$

$$n = 2^k \cdot n = 2^{k+1}$$

$$8^{\log_2 k} \quad \log_2 n = k+1$$

$$k = \log_2 n - 1$$

$$\frac{n^3}{8} + n^2\left[1 + \frac{2^{\log_2 n}}{2-1}\right]$$

$$= \frac{n^3}{8} + n^2 \log n$$

$$= O(n^3)$$

google → DAC problem

⇒ with & without DAC, matrix multiplication is order of $O(n^3)$, will takes same time of $O(n^3)$, so with DAC is ~~best~~ worst (or) need extra space), no need of extra stack space is req.

According to strassen's

$$T(n) = \begin{cases} O(1) & \text{if } n \text{ is } 2 \text{ or } 2 \\ 7T(n/2) + 18(n/2)^2 & \text{if } n > 2 \end{cases}$$

$$= 7T(n/2) + 4.5n^2$$

$$= 7T(n/2) + n^2$$

$$= O(\log^{\frac{7}{2}})$$

extra space is $O(\log n)$ $= O(n^{2.8})$

space $= O(\log^{32.29})$

Problem I

i/p An array of n-elements in which until some place-n elements are in increasing order and afterwards decreasing order.

o/p :- find n.

LS
BS

10 20 30 40 50 (60) 55 45 35 25

$O(\log n)$

② i/p An array of n-elements
o/p find no. of inversions.
without - DAC — n^2
with DAC — $n \log n$.

20	10	
1	2	

Position $1 < 2$.
but value $a[1] > a[2]$.
So this is inversions

$\log n$ → generally means Binary Search

③ i/p an array of n-elements in (x,y) plane

o/p :→ find closest plane ~~without~~ DAC

without DAC ⇒ $O(n^2)$

with DAC ⇒ $O(n \log n)$

④ i/p : 2 sorted arrays of size each of n.

o/p : find n^{th} smallest element in union array.

1) with merge algorithm ⇒ $O(2n) \Rightarrow O(n)$

without merge ⇒ $O(\log n)$ [binary search]

Masters theorem

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

$a \geq 1$, $b > 1$, $f(n)$ is a positive function.

① if $f(n) = O(n^{\log_b a - \epsilon})$ where $\epsilon > 0$ and ϵ is constant

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

if $f(n) = \Omega(n^{\log_b a + \epsilon})$ where $\epsilon > 0$ and ϵ is constant

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

③ if $f(n) = \Theta(n^{\log_b a})$

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

~~co~~ add $\log n$ coz of recursive problem.

1st problem

$$T(n) = 8\,T(n/2) + n^2$$

$a = 8$

$b = 2$

$n^{\log_b a} = n^3$ | $f(n) = n^2$

case (i)

$O(n^3)$

$f(n) = n^{\log_b a - \epsilon} = \frac{n^3}{n}$

$f(n) = n^{3-1}$

where $\epsilon = 1$ & constant

so ~~left~~ right hand side is greater by i.e. polynomial times greater

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

$a = 2 \quad b = 2$

$n^{\log_b a} = n^1 \mid f(n) = n^2$

case II

$\Theta(n^2)$

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

where $f(n) = n^2$

$n^{\log_b a} = n$

so ~~f(n) is~~ $f(n)$ is greater than $T(n)$ by polynomial n

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

$a = 2$

$b = 2$

$n^{\log_b a} = n \mid f(n) = n$

case (3)

$\Theta(n \log n)$

ex IV $T(n) = 64T(n/8) + n^3$

$a = 64 \quad b = 8$

$n^{\log_8 64} = n^2 \mid f(n) = n^3$

case (III)

$T(n) = \Theta(n^3)$

ex 3 $T(n) = 2T(n/2) + n \log n$

$a = 2 \quad b = 2$

$n^{\log_n a} = n \mid f(n) = n \log n$

Note master thm dont work bcoz $n^{\log_b a}$ is logarithmic times smaller than $f(n)$ but not polynomical times.

note 1 ⇒

1⇒ in case (I) $n^{\log_b a}$ is polynomical times greater than $f(n)$ (n^ϵ where $\epsilon > 1$

2⇒ in case (2) $n^{\log_b a}$ is polynomical times smaller than $f(n)$

3⇒ in case (3) $n^{\log_b a}$ and $f(n)$ are asymptotically equal.

special method

of the recurrence relation contain root operation

1) a) $T(n) = T(\sqrt{n}) + c$

⇓

i) assume $n = 2^k$

$T(2^k) = T(2^{k/2}) + c$

ii) Assume $T(2^k) = S(k)$

$T(2^k) = S(k)$

$S(k) = S(k/2) + c$

now it is in the form of master theorem

here $a = 1$ $b = 2$ $f(k) = c$

$k^{\log_b a} = k^0 = 1$ | $f(k) = c = 1$

$S(k) = \theta(\log k)$

iii) $T(2^k) = \theta(\log k)$

(iv) $T(2^{\log_2 n}) = \theta(\log \log n)$

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

2) $T(n) = 2T(\sqrt{n}) + \log n$

$n = 2^k$

$T(2^k) = 2T(2^{k/2}) + \log(2^k)$

assume $2^k = S(k)$

$T(2^k) = S(k)$

$S(k) = 2S(k/2) + \log(2^k)$ in k

$a = 2$ $b = 2$ $f(k) = \log 2^k$

$f(k) = k$

$k^{\log_2 2} = k$ | $f(k) = \log_2 2^k = k$

$S(k) = \theta(k \log k)$

log n is not polynomial while n log n is.

(iii) $T(2^k) = \theta(k \log k)$

iv) $T(2^{\log_2 n}) = \theta(\log n \log \log n)$

Q2. Consider the following C programme

```
A(n)
{
  if (n ≤ 1)
  return 1
  else
  return ( A(n/2) + A(n/2) + n)
}
```

↳ Here it is neat algorithm but a big problem.

Let T(n) = time complexity, then

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

$= 2\left[2T(n/2^2) + n/2\right] + n$

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

$\vdots$ K times

$2^K T(n/2^K) + Kn$

$= n = 2^K$

$K = \log n$

$= nT(1) + n\log n$

$= O(n \log n)$

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

$2T(n/2) + C$

i) $O(n \log n)$ ✓
ii) $O(n)$
iii) $= O(n^2)$
iv) $O(\log n)$

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

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

$= O(n)$

b = A(n/2) ⟹ T(n/2)

c = A(n/2) ⟹ T(n/2)

d = b + c ⟹ O(1)

e = d + n ⟹ O(1)

return(e) ⟹ O(1)

}

return T(n/2) + T(n/2) + n.

Consider the following

```
A(n)
{
  if ( n ≤ 1) return(n)
  else
  return ( 5 A(n/2) + 3 A(n/2) + n)
}
```

$T(n) = 5T(n/2) + 3T(n/2) + n$

$8T(n/2) + n \quad 2T(n/2) + n$

$= n^3 + n$

$= n^{\log_2 8}$ by master theorem

$n^{\log_2 2} = n$

$= O(n)$

b = A(n/2) → T(n/2)

c = 5 * A → O(1)
d = A(n/2) → T(n/2)
e = 3 * d → O(1)
f = c+e → O(1)
g = f+n → O(1)

⇒ 2T(n/2) + C

$$T(n) = \begin{cases} O(1) & \text{if } n \le 1 \end{cases}$$

Consider

n . n . n - - - n = $O(n^n)$ → value

s = 1
for (i=1 to n) ⇒ O(n) → time complexity
s = s * n

A(n)
{
if (n ≤ 1)
return (LS(n)) → it will not give O(1) but O(n) coz of function calling.

⇒ n+n+n+n+n+n = O(1) → time complexity.
O(n) → value.

⇒ 1+2+3+ - - - - - +n = $\frac{n(n+1)}{2}$
= $O(n^2)$ → value.

{ s=0
for (i=1 to n) = O(n) → time complexity.
s = s+i
}

Consider the following C-program.

A(n)
{
if(n ≤ 1) return 1
else
return (n * A(n-1)).
}

T(n) = A(n-1) time complexity

n(n-1)

$T(n)$ = for time complexity

$T(n) = O(n)$

$T(n)$ for no. of multiplication.

$T(n) = O(n)$

~~no of v~~ $T(n)$ = value

$T(n) = n^2$

$T(n)$ for time complexity

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

$T(n-1) + C$

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

$T(n-3)+C+C+C$

{ } k time

$T(n) + kC$

$= O(n)$.

$T(n)$ = no. of multiplication

$$T(n) = \begin{cases} 0 & \text{if } n \le 1 \\ T(n-1)+1 & \text{else} \end{cases}$$

$O(n)$

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

```
else
{
b = A(n-1) => T(n-1)
c = n * b => 1
return (c)
}
```

as it is an array, then it is value.

$T(n-3) + 1 + 1 + 1$

upto $n-1$ times

so no. of multiplication = $n-1$.

$T(n)$ = value

$$T(n) \begin{cases} 1 & \text{if } n = 1 \\ n * T(n-1) & \text{else } (n > 1) \end{cases}$$

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

$T(n-3) * n-2 * n-1 * n$

$n-1$ times

$1 \cdot 2 \cdot 3 \cdots (n-2)(n-1) \cdot n$

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

Bubble Sort

s/o	A[	1	2	3	4	5	6	7	8]
		85	15	26	75	19	40	20	99

Pass ① 15 26 75 19 85 40 26 85) 99 → 7

largest element at the last

Pass ② 15 26 19 75 40 26 75 75) 85 99 → 6 - c

second largest

second last position

2 - quest from DAC
2 - quest from greedy
total = 4

Pass ③ 15 19 26 36 40 75 85 99
15 26 19 40 36
5 - comparisons
5 swaps / 0 - best

④ 15 19 26 36 40 75 85 99
4 - comparisons
4 - swaps / 0 - best

15 19 20 36 40 75 85 99
3 comparisons

and so on

final bubble sort
15 19 26 36 40 75 85 99

15 19
15 19

total = (n-1) passes

ok
① Bubble sort requires n-1 passes

total comparisons = ~~1+2+3+...(n-1)~~ + n-2+ n-3 + ... +2+1
= ~~n(n+1)~~ $\frac{n(n-1)}{2} = O(n^2)$ [BC, WC, AC]

total swaps
BC → 0
WC → 1+2+3+.... +n-1 = $\frac{n(n-1)}{2} = O(n^2)$

Avg case
$= \frac{n}{2} * \left(\frac{n}{2} - 1\right) = O\left(\frac{n^2}{4}\right) = O(n^2)$

④ total time complexity = total comparisons + no. of swaps
$= O(n^2)$ $n^2 + n^2$ [BC, WC, AC]

⑤ total comparisons will be greater or equal to no. of swaps

⑥ in place, stable

Note ||

if in two passes of which are consecutive of bubble sort, that means n array elements are sorted, no need to go for more passes.

In bubble sort at any place 2 consecutives passes outputs are same then stop the bubble sort bcoz array is already sorted

so

BC	WC, AV	for bubble sort
⇓	⇓	
O(n)	O(n²)	

I/P 80 25 45 66 77 80
Pass ① 25 45 66 77 80
Pass ② " " " " "

Selection Sort (will give min element after each pass)

i/p A[	85 (15)	45	26	70	19	50	80	15 (89)]
	1	2	3	4	5	6	7	8

Pass ①

$i = 1$

min = ~~1~~ ~~2~~ ~~3~~ ~~5~~ 8

So minimum is at 8th position

so swap(a[i], a[min])

15	45 (19)	26	70	19 (45)	50	80	89
	4	5	6	7	8		

here comparisons = n-1

swap = 1 (in every case)

in selection, each pass contain only one swap at most one swap

Pass ② $i = 2$

min = ~~2~~ ~~3~~ 5 (n-2) comparisons

15	19	26	~~45~~ 70	45 (70)	50	80	89
1	2	3	4	5	6	7	8

Pass ③ $i = 3$ (n-3) comparisons

min = 3

Pass ④ $i = 4$ (n-4) comparisons

min = 5

15	19	26	45	~~70~~ 50	70	80	89

Pass ⑤ 15 19 26 45 50 70 80 89

note (1) it requires (n-1) passes and 1 swaps at each passes at most one

(2) total comparisons = $1+2+3+\cdots+n-1$

$= \frac{n(n-1)}{2} = O(n^2)$ [NC, BC, WC]

total no. of swaps = $n-1 = O(n)$

Selection sort is better in terms of swaps operation on the worst case than that of bubble sort

~~Time Complexity~~

Total comparisons = n^2

total swaps = $n-1$

then time complexity = $n^2 + n - 1$

$= O(n^2)$

inserting sort

i/p : A[	10	20	30	25	40	50	15	70	5]
	1	2	3	4	5	6	7	8	9

Pass ① [10] → 0 comparisons

Pass ② [10 | 20] → 1 comp

Pass ① [50 | 40 | 30]

Pass ② [40 | 50 | 30] 1-c

[40 | 50 | 25 | 30]

Pass ③ [40 | 50 | 25 | 30 | 40]

Pass ④ [40 | 50 | 25 | 30 | 40 | 60]

[40 | 50 | 25 | 30 | 40 | 60 | 15]

Pass ⑤ [10 | 15 | 20 | 30 | 40 | 50]

Pass ⑥ 10, 15, 20, 25, 30, 40, 50, 15

[10 | 15 | 15 | 20 | 30 | 40]

Pass ⑦ 10, 15, 20, 25, 30, 40, 50, 70

Note:
1) It is inplace, stable

best

Best Case

			c	s
o/p: 10, 20, 30, 40, 50				
Pass ①	[10]	0 comb	0	
Pass ②	[10 | 20]	one comb	1	0
Pass ③	[10 | 20 | 30]	1-cmb	1	0
Pass ④	[10 | 20 | 30 | 40]	1 cmb	1	0
Pass ⑤	[10 | 20 | 30 | 40 | 50]	1-cmb		

So total comparisons = n-1
Swap = 0(for) 0
Time Complexity = d·c + f·s
= O(n)

1) of all the sorting algo, insertion sort has the best case i.e. O(n) (when array is already sorted)

when array is sorted quick sort gives worst case.

1) Insertion sort algo, best case will take n-1 comparisons and 0 swaps, so total time complexity is O(n)

2) In the given array most of the elements are sorted, then insertion sort will give best case, but quick-sort will give worst case

Worst Case - (when array is sorted in reverse order)

50 40 30 20 10

Pass ①	[50]		
Pass ②	[50 | 40]	1-cmp	1 swap
Pass ③	[40 | 50 | 30]	2-cmp	2 swap
	[30 | 40 | 50]		

Pass ③ | 10 | 20 | 30 | — 1-c

Pass ④ | 10 | 20 | 30 | 25 — 1-c

| 10 | 20 | 25 | 30 |

Pass ⑤ | 10 | 20 | 25 | 30 | 40 |

Pass ⑥ | 10 | 20 | 25 | 30 | 40 | 50 |

Pass ⑦ | 10 | 20 | 25 | 30 | 40 | 50 | 15 |

| 10 | 15 | 20 | 30 | 40 | 50 |

Pass ⑧ 10, 15, 20, 30, 40, 50, 15

| 10 | 15 | 15 | 20 | 30 | 40 | 50 |

Pass ⑨ 10, 15, 10, 25, 30, 40, 50, 70, 5

note:-
⇒ It is inplace, stable

~~best~~

Best Case

		c	s
o/p: 10, 20, 30, 40, 50		0	
Pass ①	| 10 | — 0 comb	1	0
Pass ②	| 10 | 20 | — one comb	1	
Pass ③	| 10 | 20 | 30 | — 1-cmb	1	0
Pass ④	| 10 | 20 | 30 | 40 — 1 cmb	1	0

Pass ⑤ | 10 | 20 | 30 | 40 | 50 — 1-cmb

So total comparison = n-1
swap = ~~O(n)~~ 0
time complexity = c-c + f-s
= O(n)

of all the sorting algo, insertion sort has the best case i.e. O(n) (when array is already sorted)

when array is sorted quick-sort gives worst case.

1) Insertion sort algo, best case will take n-1 comparisons and 0 swaps, so total time complexity is O(n)

2) In the given array most of the elements are sorted, then insertion sort will give best case, but quick-sort will give worst case.

Worst Case - (when array is sorted in reverse order)

50 40 30 20 10

Pass ① | 50 |

Pass ② | 50 | 40 — 1-cmb | 1 swap

Pass ③ | 40 | 50 | 30 — 2-cmb | 2 swap

| 30 | 40 | 50 |

Pass ④ | 30 | 40 | 50 | 20 | 3-C | 3 swaps

| 20 | 30 | 40 | 50 |

Pass ⑤ | 20 | 30 | 40 | 50 | 10 | 4-C | 4 swaps

| 10 | 20 | 30 | 40 | 50 |

total comparison = 1+2+---+n-1
= n(n-1) = O(n²)

total swaps = 1+2+---+n-1
= n(n-1) = O(n²)

total time complexity = O(n²)

Question: while applying insertion sort, to find a correct place of a particular element we apply binary search, what will be the time complexity of insertion sort algo, if we replace linear search by binary search for n elements

worst case

| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 25 |

total inversion = 5

30, 25
40, 25
50, 25
60, 25
70, 25

in B.S. no. of comp decreases, but no. of swaps increases.

B.S.
for 1 element = log n — comp
n — swaps

n elements = n(n+log n)
= n² + n log n
= O(n²)

Note:
The no. of inversions in the given array is equal to the no of right shifts in insertion sort

2- if the no inversions are less than n, then the array is almost sorted

in the given array, if at all max n inversions are there, in the insertion sort, then the insertion sort will give best case algo

eg
10 20 30 40 50 60 70 5

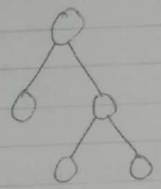
Average Case:
half best + half worst

(0 1 1 1 1 1) + (n n n n n)
n/2 * 1 n/2 * n

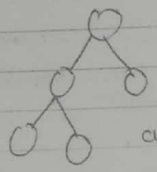
= n/2 + n²/2 = O(n²)

Heap-Sort

Almost complete binary Tree

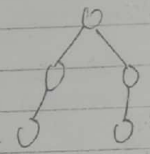


BT ✓
Almost BT ✓



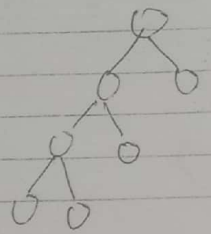
almost complete binary tree

BT ✓
Almost Complete BT ✓



BT ✓
ACBT ✗

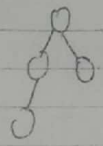
first complete left side



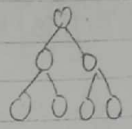
BT ✓
ACBT ✗

previous level gaps are not allowed,

greedy techniques completely uses heaps.



BT ✓
ACBT ✓

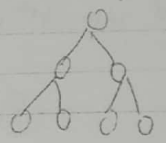


BT ✓
ACBT ✓

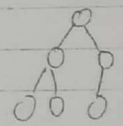
A Binary Tree is said to be a almost complete binary tree if and only if

i) without going left don't go to right (in case of every node)

ii) at every node, without completing the current level, don't go to the next level.



BT ✓
ACBT ✓
CBT ✓

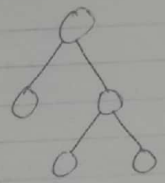


BT ✓
ACBT ✓
CBT ✗

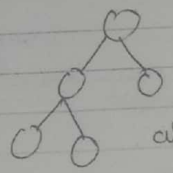
A maximal almost Complete BT is called Complete BT.

Heap-Sort

Almost complete binary Tree

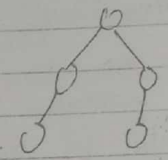


BT ✓
Almost BT ✓



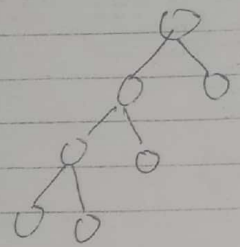
almost complete binary tree

BT ✓
Almost Complete BT ✓



BT ✓
ACBT ✗

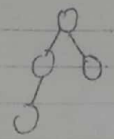
first complete left side



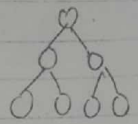
BT ✓
ACBT ✗

previous level gaps are not allowed,

greedy techniques completely uses heap.



BT ✓
ACBT ✓

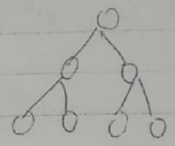


BT ✓
ACBT ✓

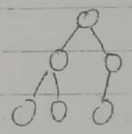
A Binary Tree is said to be almost complete binary tree if and only if

i) without going left don't go to right (in case every node)

ii) at every node, without completing the current level, don't go to the next level.

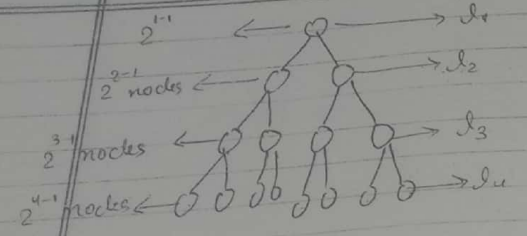


BT ✓
ACBT ✓
CBT ✓



BT ✓
ACBT ✓
CBT ✗

A maximal almost complete BT is called complete BT.



① in ACBT.

$15 = \lceil \frac{15}{2} \rceil = 8$ external nodes

$15 \Rightarrow \lfloor \frac{15}{2} \rfloor = 7$ internal nodes

① If the ACBT contain n nodes

no. of leaf node $= \lceil \frac{n}{2} \rceil$

no. of internal node $= \lfloor \frac{n}{2} \rfloor$

② The maximum no. of nodes present at K^{th} level of almost complete binary tree $= 2^{K-1}$

③ maximum nodes present in K^{th} level of ACBT $= n$.

$n = 2^K - 1$.

$2^K = n+1$

$K = \log_2(n+1)$

maximum no. of levels in BT with 15 nodes

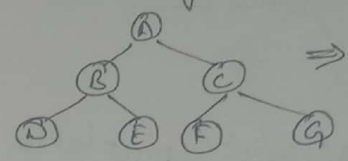
$= 15$

Balanced trees have less no. of levels than unbalanced B.T so it balanced trees will same time and will have less space required. i.e. in log.

Heap Sort

Q How binary tree will store in comp?

ex



⇒

1	2	3	4	5	6	7	
A	B	C	D	E	F	G	

N L R

If a node is stored in i^{th} place of the array then -

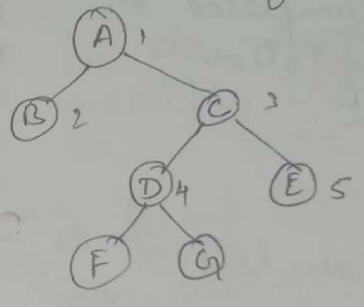
(i) parent of $i = \lfloor i/2 \rfloor$

(ii) left child of $i = i*2$

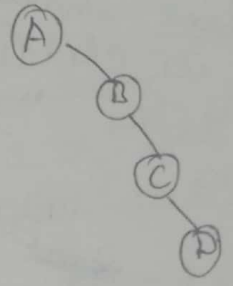
(iii) Right child of $i = (2i)+1$

$= (\text{left child of } i) + 1$

Q Store the following B.T in the form of array



1	2	3	4	5	6	7	8	9
A	B	C			D	E	F	G

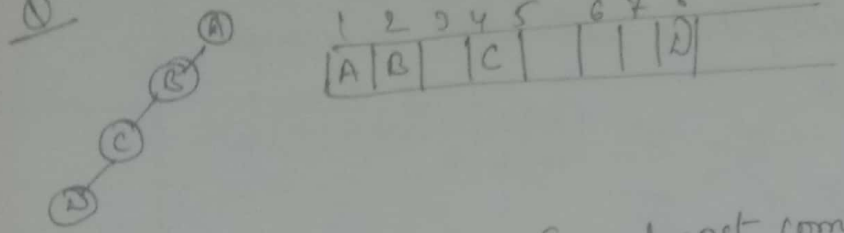


1	2	3	4	5	6	7	8	9	10	11	12
A	-	B	-	-	-	C	-	-	-	-	-

no. of nodes $= n$

no. of places $= 2^n - 1$

①



Ques Note:

n (nodes)

- Min size array → n-length array. ∴
- max. size array required to store n-nodes tree → $2^n - 1$ length array.

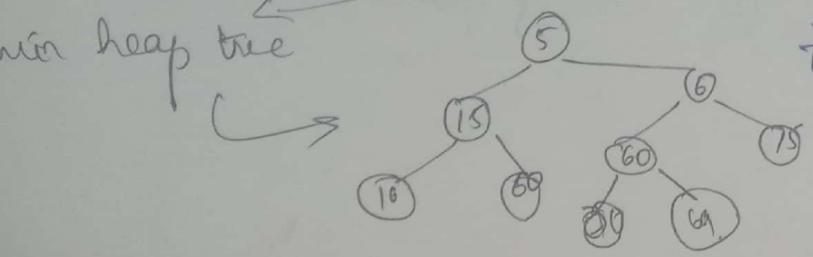
For almost completed binary tree → binary tree contains n-length array.

Note

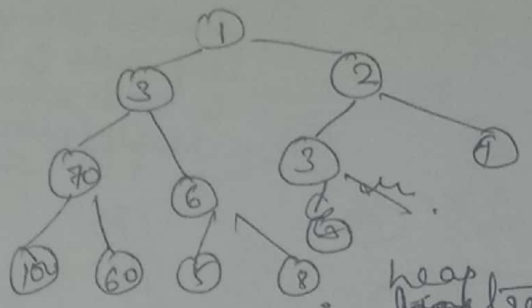
If the given binary tree is almost completed or completed then array representation is advisable otherwise link-list.

Heap Tree

In almost completed binary tree it is a min heap if root node is minimum comparing its children.



Time to find max = O(1)
find min = O(n)



In min heap tree → min heap tree root is small of array.

In almost completed binary tree in which at every tree root is min among most other children. or almost equal.

* In Max heap tree the uppermost root node is greater than all the other. or equal / not equal.

Min heap tree.

(min element can be found = O(1)
max " " " " = O(n)

Purpose of Data structure create and destroy nodes according to the require mem

Time to find min element if:
- sorted in ascending = O(1)
- sorted in descending = O(n)

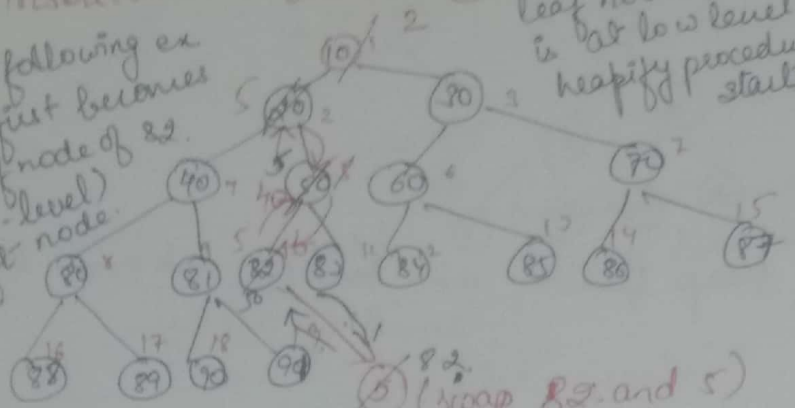
Time to find max element is O(n/2) i.e. O(n) min heap (n/2) becoz last level element would

Min-Heap

Insertion → Insert (5)

Inserted element becomes leaf node to the node which is at low level and then heapify procedure starts

In following ex 5 first becomes leaf node of 82. (4-level) left node.



(swap 82 and 5)

$\log_2 n + 1$

$\log_2 2^4$

Best Case → O(1)

worst case → log(n)

Average Case = $1 + 2 + 3 + \dots + \log n$

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

$= O(\log n)$

Note: Inserting 'n' element into min heap which already contain 'n' elements.

1) O(1) = best case
2) O(log n) = worst and average case

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
10	20	30	40	50	60	70	80	81	82	83	84	85	86	87	88	89

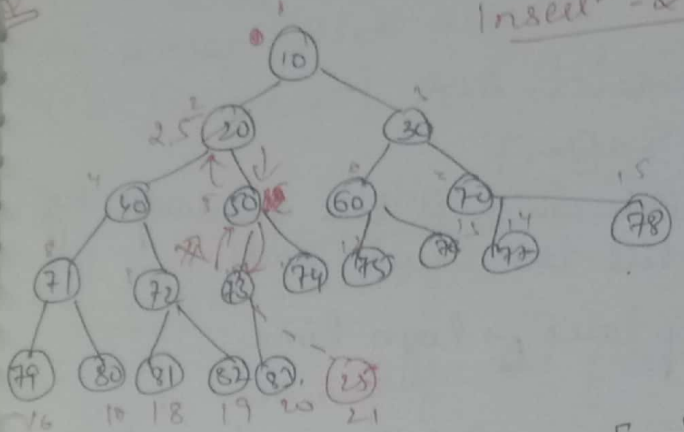
Parent of 5 = 20/2 = 10 (swap)

" " 10 = 5 = 5/2 = 2 (swap)

18	19	20
90	91	5

Parent of 2 = 1 = 5

Insert -25

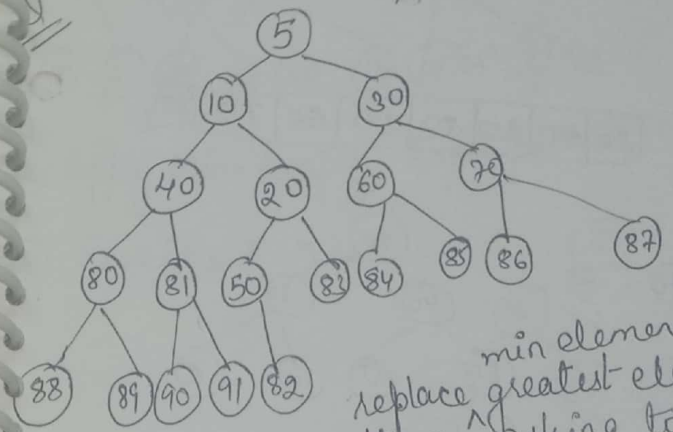


Heapify Top in case of insert

$21/2 = 10.5 = 10$ swap a[21] & a[10]

$10/2 = 5 \Rightarrow$ swap(10, 5)

$5/2 = 2.5 \Rightarrow$ swap(5, 2)



log n levels

Delete any one element from min-heap means delete min element from heap

replace min element by greatest element then min heapify bottom ↑ asking to lower levels.

5	10	30	40	20	60	70	80	81	50	83	84	85	86	87	88	89	90	91

$x = a[i]$ Total elements = n

$a[i] = a[n]$

now total elements = $n-1$

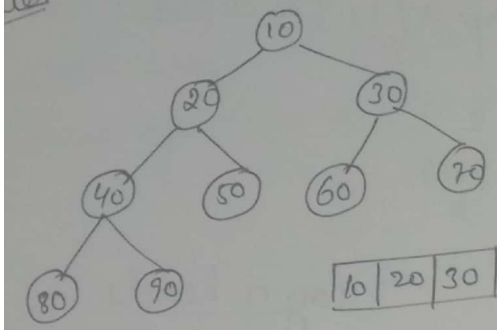
check if $a[i] > a[2*i+1]$

check from childs which is less and swap by less elements till the level ends.

min-heapify top take' → $\log n$ time

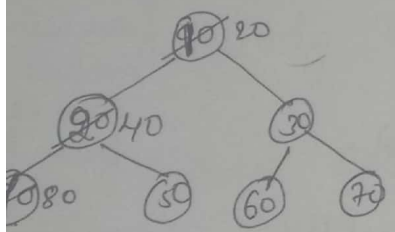
ues

Delete one element

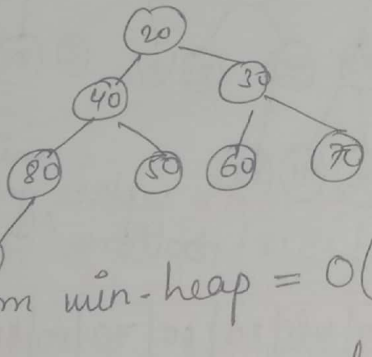


10	20	30	40	50	60	70	80	90

ultant tree



20	40	30	80	50	60	70	90



find smallest element from min-heap = $O(1)$

delete " " " " " = $\log n$.

To Note:

* To delete a element from min-heap or max heap which already contains elements $O(1)$ [best case] and $O(\log n)$ (worst case & average case)

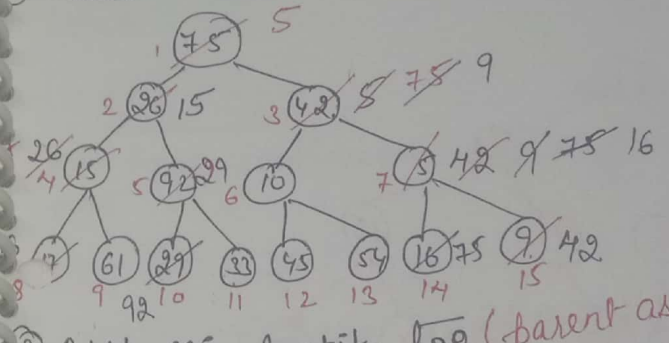
* In min-heap finding max element = $O(n)$ (in min heap max element is found in last level)

* BUILD-HEAP [creating a min heap tree ~~using~~ with $O(n)$ time]

ex: create min-heap for the following n-element array of

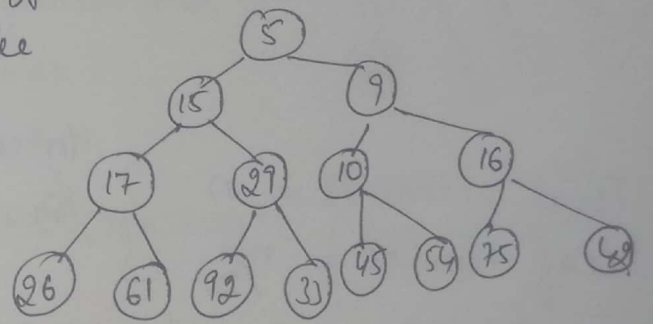
A [75, 26, 42, 15, 92, 10, 5, 17, 61, 29, 33, 45, 54, 60, 16, 9]

1) create B. Tree



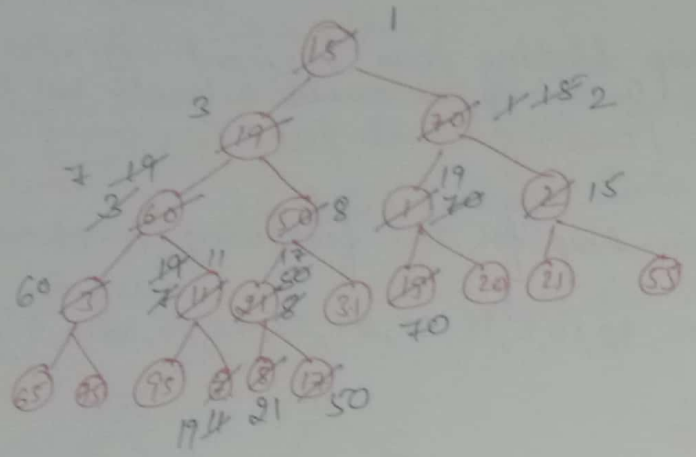
2) Apply Min-heapify top (parent ask child.) from bot

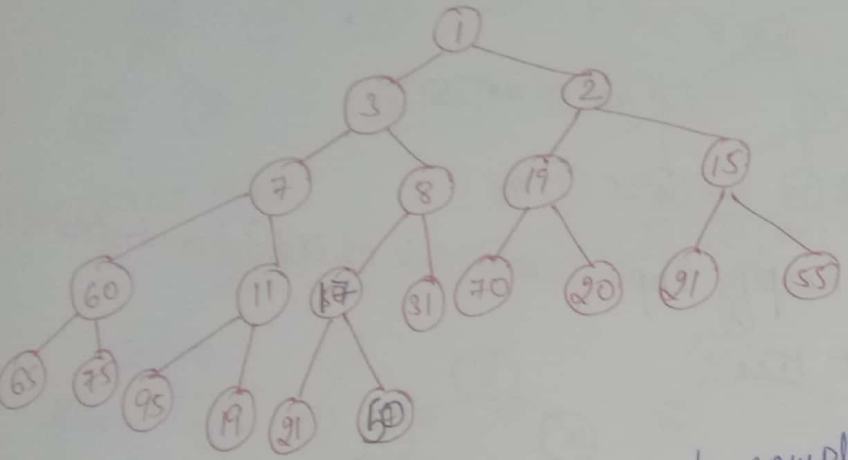
3) Resultant tree



ex2

15, 19, 70 60 50 1 2 3 11 21 31 19 20 21
55 65 75 95 7 8 17 O(n) time





if Total elements = n } In complete binary tree
then leaf nodes = n/2

Total swaps from bottom each level = $\frac{n}{2^0}\times 0 + \frac{n}{2^1}\times 1 + \frac{n}{2^2}\times 2 \cdots \frac{n}{2^{\log n}} \times \log n$.

$= n\left[\frac{0}{2^0} + \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots \frac{\log n}{n^{\log n}}\right]$

$= n\left[\left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \cdots + \left(\frac{1}{2}\right)^{\log n}\right]$

$= n * O(1)$

$= n \rightarrow$ swaps.

$= n \rightarrow$ swaps $+ 2n \rightarrow$ comparisons → each node is compared with left & right child so $2 * n$

$\Rightarrow 3(n)$

$\Rightarrow O(n)$

* create a min-heap using brute force = O(n
* create " " " " build heap = O(n)
* HEAP-SORT
min heap continuous deletion gives one by one element as descending order
max " " " " " descending order

Algo heap sort

* create min heap using Build-heap ⇒ O(n)
* Delete one by one and store from Right Hand
continue n times → n

O(log n) → O(n log n) [worst case / Average case]

if best case → O(n * 1) ⇒ O(n)

It is in-place

* It is not stable.

* To find 1st min. ⇒ 1 fighting — 0 comparison
  " " 2nd min. ⇒ 2 " — 1 —
  " " 3rd " ⇒ 3 fighting — 2 —
  To find nth min ⇒ n fighting — (n-1) comparison.
  50th min-finding ⇒ 50th fighting — 49 comparison
  To find 50th min we have to find 1st min, 2nd min, 3rd min, 4th min. ... upto 49th min. ⇒ $49 \times \frac{50}{2}$

$\Rightarrow n \times \frac{(n-1)}{2} \Rightarrow O(n^2)$

$O(n^2)$

① Selection sort n pass.
② Bubble sort 1 pass

Min-Heap Tree

1. Creating min-heap ⇒ O(n) Build heap
2). Insertion ⇒ O(1) [Best case)
   O(logn) [worst / Average case)
3) Deletion ⇒ O(1) [B.C)
   O(logn) (WC, A C]
4) @ 10th (10 ways) min. finding ⇒ $\frac{9 \times 10}{2} = 45 = O(1)$
⑤ → Delete 9 min ⇒ 9 logn.
⑥ 10 pass of selection sort = O(n)

3) max-element finding ⇒ $\frac{(n-1)n}{2} = O(n^2)$
* Bubble sort 1st pass ⇒ O(n)

GREEDY TECHNIQUE

Sorting techniques

Comparison based Sorting techniques

	B.C	W.C (upper bounds)	A.C
* Merge Sort	nlogn	nlogn	nlogn.
* Quick Sort	nlogn	n^2	nlogn
* Bubble Sort	n^2	n^2	n^2
* Selection sort	n^2	n^2	n^2
* Insertion Sort	n	n^2	n^2
* Heap Sort	n	nlogn	nlogn

Non-comparison based S.T (Assumptions are required) — Linear time complexity of O(n). Read from coleman

- Radix sort
- Counting Sort
- Bucket Sort

} O(n)

In all comparison based S. Tech. upper bounds what will be lower to bound
nlogn

In all comparison based Sorting Tech. Lower bounds, lower bound is (n) → ans. (Best case)

(4 marks)

GREEDY TECHNIQUES

Note: In greedy Technique most of the problems contain n-i/p and our objective is finding a subset which will satisfy our conditions and optimize our goal.

1. Solution Space:
Set of all possible solutions over the given n- no. of i/p's is called solution space.

2. Feasible Solution:
Set of all possible solutions which will satisfy our conditions.

3. Optimal Solution.
Those feasible solutions which will optimize our goal is called optimal solution. Need not be unique.

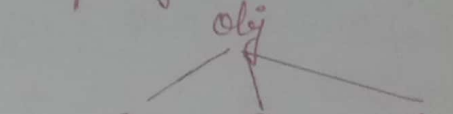
Applications of Greedy
1. Job sequencing with deadlines
2. Knapsack problem.
3. Huffman coding
4. Optimal Merge pattern
5. Minimum Cost spanning tree
   (i) Kruskal
   (ii) Prims

6. Single Source shortest path
   (i) Dijkstra's Algo.
   (ii) Bellman-ford algo
   (iii) Breadth first traversal

KNAPSACK PROBLEM (Real/fractional)

i/p : n - objects — weight, Profit

knapsack capacity = M



$0 \le x_i \le 1$

1) Problem: $\sum_{i=1}^{n} w_i > M$

2) Feasible: $\sum_{i=1}^{n} w_i x_i \le M$

3) Optimal $\sum_{i=1}^{n} x_i * P_i$ max

Ex-1

$n = 3$ $\quad m = 20$

Object	ob_1	ob_2	ob_3
Profit	25	24	15
weight	18	15	10

w_2 P_i

$15 * \frac{24}{15} + 5 * \frac{15}{10} \Rightarrow 24 + 7.5 = 31.5$ Profit max

$\sum_{i=1}^{n} w_i x_i = 0 \times 18 + 1 \times 15 + \frac{5}{10} \times 15 = 15 + 7.5 = 22.5$

(x_1, x_2, x_3)

$\frac{25}{18}$ 1.3 $\quad \frac{24}{15}$ 1.6 $\quad \frac{15}{10}$ 1.5

Note:
In greedy knapsack we will always get optimal ~~knapsac~~ solution by giving priority to both profit and weight.

Q $n=7$ $M=25$

Objects	ob_1	ob_2	ob_3	ob_4	ob_5	ob_6	ob_7
Profits	10	7	15	22	16	50	40
Weights	2	1	4	6	3	8	6
P_i/w_i	5	7	3.75	3.63	5.33	6.25	6.66

$\sum_{i=1}^{n} w_i \times p_i = 2\times\frac{10}{2} + 1\times\frac{7}{1} + 4\times\frac{15}{4} + 1\times\frac{22}{6} + 3\times\frac{16}{3} + 8\times\frac{50}{8} + 6\times\frac{40}{6}$

$\sum_{i=1}^{n} w_i = 2 + 1 + 4 + 1 + 3 + 8 + 6$

$\sum_{i=1}^{n} w_i p_i = 10 + 7 + 15 + \frac{22}{6} + 16 + 50 + 40$

$= 141.66$ (Max. Profit)

KNAPSACK ALGO (Fractional)

1). for ($i=1$ to n) $A[i] = P_i/w_i$ } $\Rightarrow O(n)$

2). Sort array A in decreasing order } $\Rightarrow O(n \log n)$ By merge sort/ heap sort.

3). Take one by one object until ~~k~~ capacity of knapsack becomes zero } $O(n)$

Final $\Rightarrow n + n\log n + n \Rightarrow O(n \log n)$

Q what is the time complexity of knapsack problem if objects are already arranged in P_i/w_i decreasing order / increasing order (sorted)

$O(n)$

JOB SEQUENCING WITH DEAD LINES

1). Single CPU available → fixed fraction of time alloted to every job
2). No - interleaving. (Round-Robin fail)
3). Arrival time of every job is same (FCFS fail)
4). Running time of every job is 1-unit (SJF fail)

ex1 $n = 4$

jobs:	j_1	j_2	j_3	j_4
Profit:	200	150	300	250
Deadline:	2	1	2	1

optimal soln := (J_4, J_3)

(j_4, j_3) 250 300 = 550 | (j_4, j_1) | (j_2)
(j_1, j_3) | (j_3)
(j_2, j_1) | (j_3, j_1) | (j_4)
(j_2, j_3) | (j_1) | $( )$

} 11 feasible solutions.

ex2 $n = 7$

jobs	j_1	j_2	j_3	j_4	j_5	j_6	j_7
Profit	(25)	(75)	15	(89)	(91)	(60)	(55)
Deadline	5	3	4	2	6	5	3

1	2	3	4	5	6
55	89	75	25	60	91
j_7	j_4	j_2	j_1	j_6	j_5

max profit = 395 penalty ⇒ j_3 (15)

1) Find the max. deadline
2) Take an array of size max. deadline
3) Fill it from right side considering their deadlines & profits

} Apply this procedure in exams

ex3 $n = 9$

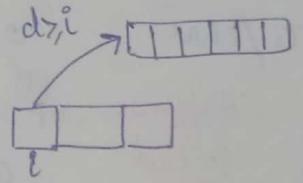
jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
profit	(25)	(15)	35	10	(45)	(55)	5	(16)	(92)
deadline	7	5	2	5	3	2	1	4	3

1	2	3	4	5	6	7
J_5	J_6	J_9	J_8	J_2		J_1
45	55	92	16	15		25

= 248

1) Sort in decreasing order of profit = $O(n \log n)$ merge sort
Job left = J_3, J_4, J_7
Penalty 35 + 10 + 5 = 50
2) Find max deadline
3) Take array of size of max deadline and start } O(
4) from R.H.S
4). For every slot i find a job with a deadline $d \geq i$ using linear search → n

} O(



Doubt:
Is it necessary to sort according to profit? If yes then how it is helpful in further process

HUFFMAN CODING

1) Data Encoding technique
2) Data Compression technique

M = 100

a = 45, b = 15
c = 2, d = 30
e = 5, f = 3

Sender → Receiver

compress.

6 char used
So we can represent 6 char with the help of 3 bits coz $2^3 = 8$ diff. seq.

a = 000
b = 001
c = 010
d = 011
e = 100
f = 101

Uniform coding ⇒

a = 45 * 3
b = 15 * 3
c = 2 * 3
d = 30 * 3
e = 5 * 8
f = 3 * 8

100 char = 300 bits
3 bits / char.

⇒ Can you compress further.

ASCII uniform coding

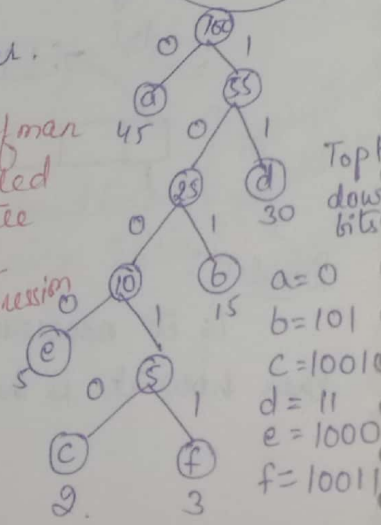
freq * bits
a = 45 * 8
b = 15 * 8
c = 2 * 8
d = 30 * 8
e = 5 * 8
f = 3 * 8

100 char = 800 bits
Avg 1 char = 8 bits

Huffmann coding
↓
Non uniform coding
↓
Ter Freq = ↓ bits
Ver Freq = ↑ bits

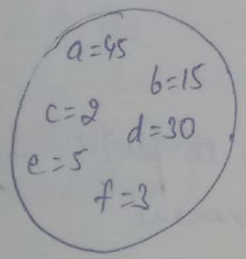
assign ⇒
left 0
right 1

Huffman coded tree for compression



Top to down bits

a = 0
b = 101
c = 1001
d = 11
e = 1000
f = 1001



left less freq
right more freq
in tree

Take 2 least frequency & add & return result

Total Bits = $a \atop 1*45 + b \atop 3*15 + c \atop 5*2 + d \atop 2*30 + e \atop 4*5 + f \atop 5*3$ (10)

= 45 + 45 + 10 + 60 + 20 + 15

100 char = 195 bits
1 char = ' '

merge sort Heap " O(n log n)

Job Scheduling with deadlines

1. Sort decreasing order of profit (n log n)
2. max deadline
3. Take array & start from R.H.S
4. for i ⇒ n find job with deadline d ≥ i using linear search (n × n) n^2

M
M_2
M_3
M_4
M_5
M_6 =

Total :

75 char
1 char

encoded =

60	
100	
101	
10	
11	
1	

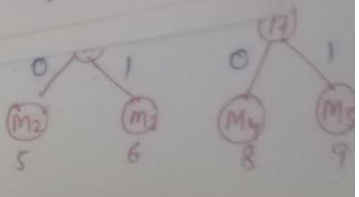
Huffman Coding

1. Data encoding
2. " compression

create min heap. n log n
Delete one min. log n
Delete one more min. log n
Add both min log n
Peek in heap (log n)
& create Huffman coding tree

M_6 ⇒ 3 log n * 2

TATA



1). Create Min-heap O(n)
2) Delete one Min from min heap we will get one min. O(logn)
3). Delete one more min. we will get another min O(logn)
4) Add both min.
5) insert in heap. O(logn)
O(3logn)
6). Follow the procedure to create huffman coding tree. Add min value on left side and higher value on right.
7). After formation of tree assign 0 to left hand and 1 to right hand (Top to bottom)

creation
Time = n + (n-1)3logn O(n²) using array
= n + 3nlogn
= O(nlogn)

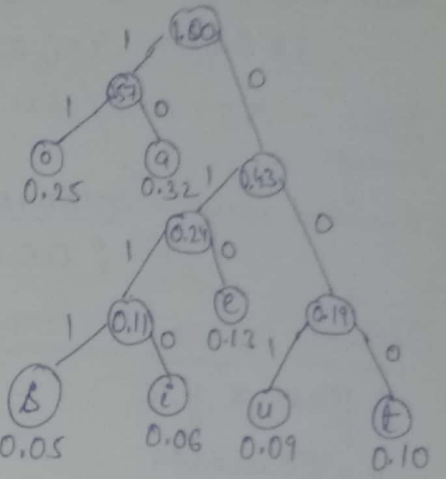
Min heap contains $n.2^n$ elements then one insertion will take ? time.
ie know in n elements min heap one insertion take logn time
so here. ⇒ $\log(n.2^n)$
$= \log n + \log 2^n$
$= \log n + n \log 2$
$= \log n + n$
$= O(n)$

Q M = (a, e, i, o, u, s, t)
0.32 0.13 0.25 0.05 0.10 0.06 0.09

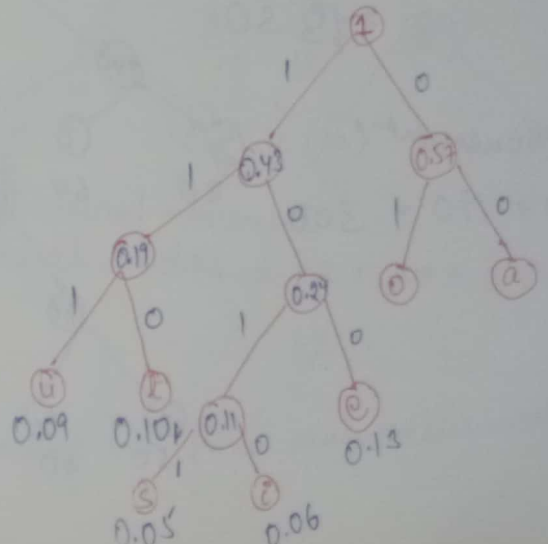
Assume left = 1 Right = 0

a = 10
e = 010
i = 0110
o = 11
u = 001
s = 0111
t = 000

Total bits = 2×0.32 + 3×0.13 + 4×0.06 + 2×0.25 + 3×0.09 + 4×0.05 + 3×0.10
= 0.64 + 0.39 + 0.24 + 0.50 + 0.27 + 0.20 + 0.30
=



a = 00
e = 100
i = 1010
o = 01
u = 111
s = 1011
t = 110



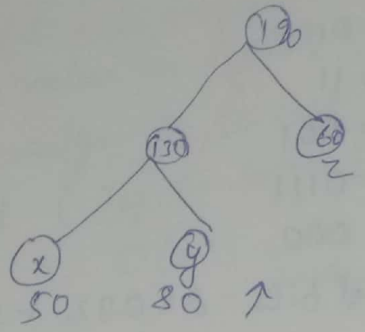
Total $= 2\times0.32 + 3\times0.13 + 4\times0.06 + 2\times0.25 + 3\times0.09 + 4\times0.05 + 3\times0.10$

1 char = 2.54 bits/char
Average

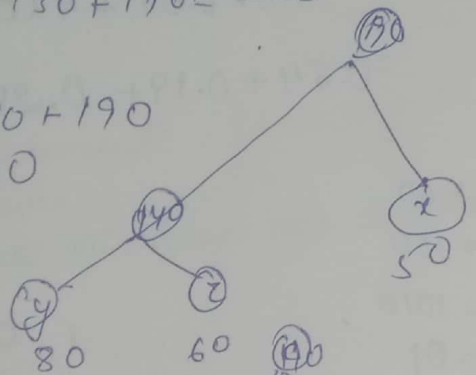
③ Encoded msgs = 10100110000111101111 10
i o e a u & u t

Optimal Merge Pattern

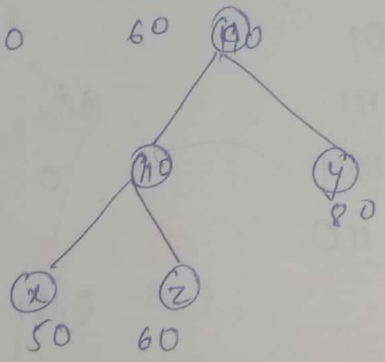
3 files x, y, z
x = 50 records
y = 80 records.
z = 60 records.



records movements = 130 + 190 = 320



records movements = 140 + 190
= 330

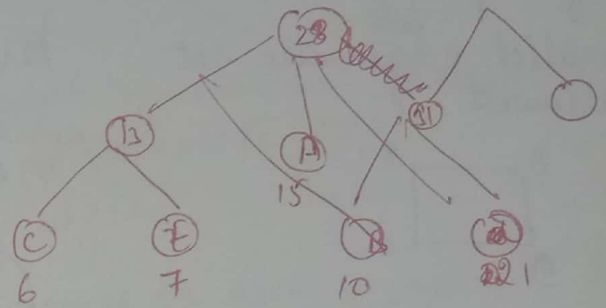


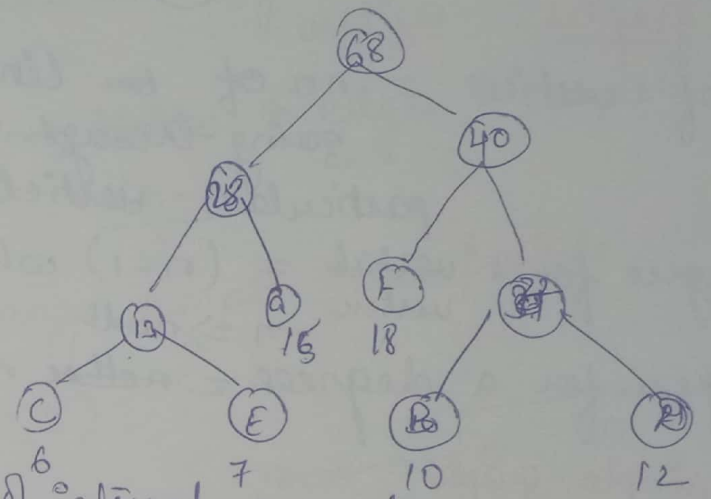
Records movement:
= 110 + 190 = 300

To merge three files. ∴ 3 merge patterns those are m_1, m_2, m_3,

Ques 6 files (2-way merge tree)

A → 15
B → 10
C — 6
D — 12
E — 7
F — 18





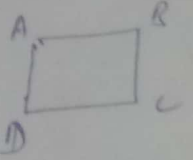
min no. of internal movements
add internal nodes. = 13 + 28 + 22 + 40 + 68
= 171

Note:
start making tree from same value nodes.

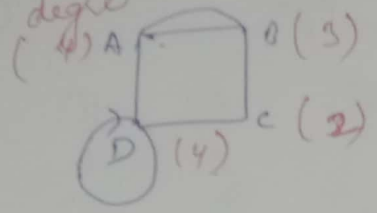
7 COST
MINIMUM SPANNING TREE (MST)

Types of Graph

Simple graph
Self loop is not allowed
&&
Parallel edges are not allowed



Self loop are allowed
OR
Parallel edges are allowed



Simple graphs

degree of vertices = no. of lines (edges) going through a particular vertex

max. degree for a vertex = (n-1) where n ⇒ node

min degree for a degree = nothing (0)

If the simple graph

max. degree of max degree = infinite (undefined)

min degree of max = 0

TYPES OF SIMPLE GRAPHS

In any simple graph any vertex deg contain no. edges = NULL graph

null graph because no edges

In any simple graph if vertex doesn't have maximum vertices → complete graph

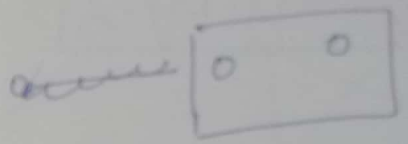
edges

no. of vertices × degree of each = faces
vertices

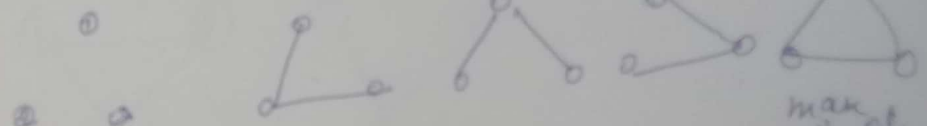
Total No. of edges in complete graph = $\frac{n(n-1)}{2}$.

How many simple graph possible with n vertices

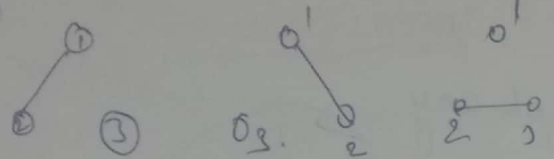
* n=2 (1,2)



* n=3 (1,2,3)



max simple graph



8 - graphs (2^n graphs)

~~For each edges~~ ~~have~~ ~~mode~~

max edges for n vertices = $\frac{n(n-1)}{2}$

Total no. of simple graphs with n vertices = $2^{n(n-1)/2}$

* Note

A simple Graph $G(V, E)$ (max. edges) $2^{\text{max edges}}$

$|E| \leq \frac{v(v-1)}{2}$

$|E| \leq c.v^2$

$|E| = O(v^2)$

$\boxed{\log E = O(\log V)}$

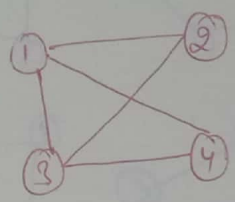
SPANNING TREE

A subgraph 'S' is said to be spanning tree after given graph 'G' iff —

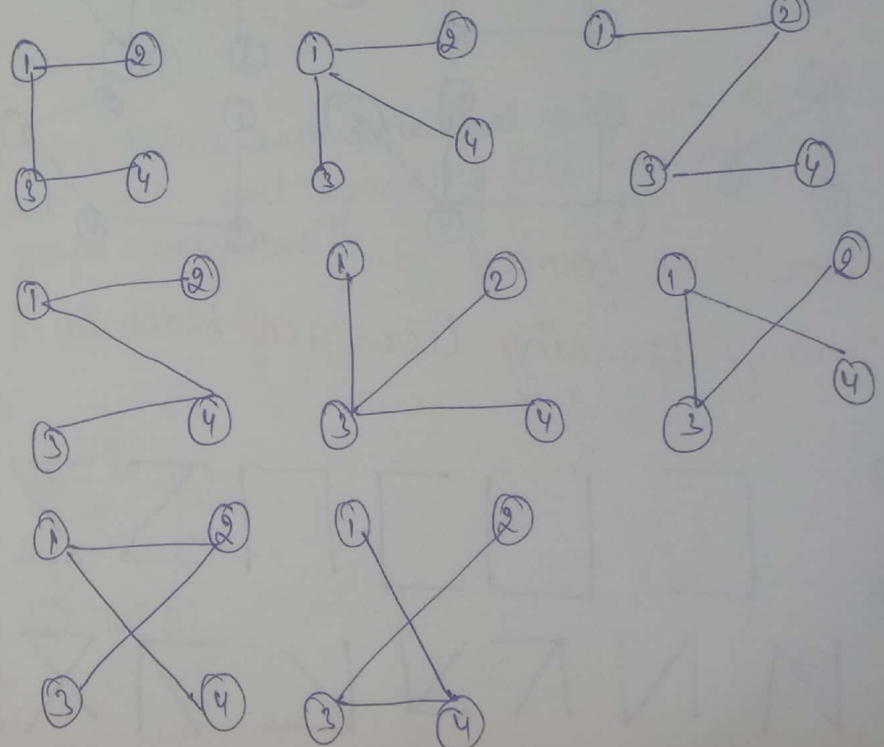
1) It should contain all vertices of G.
2) It should contain (n-1) edges. if G contains n vertices.
3) S cannot contain cycle. no cycle

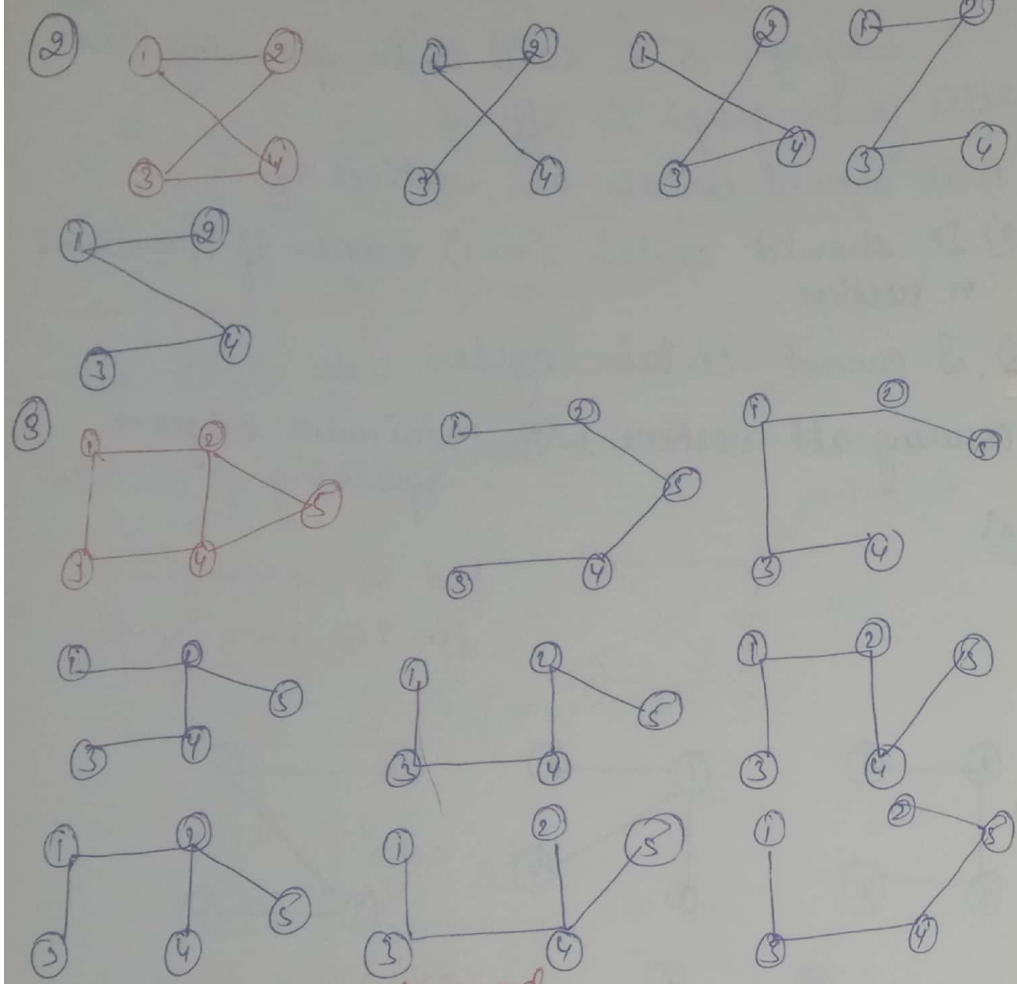
covering all vertices with minimum edges → Spanning.

ex1

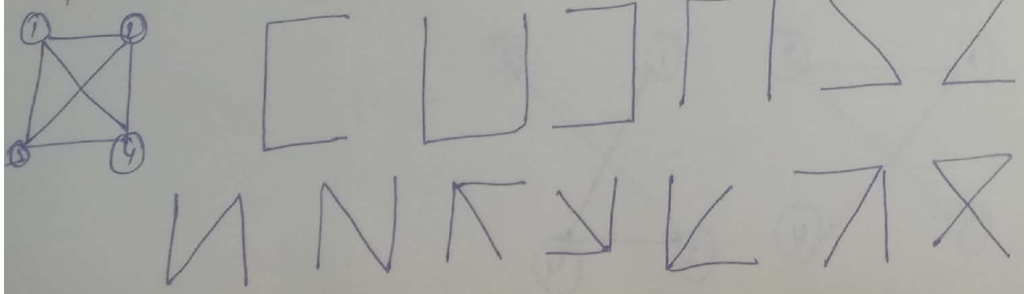


find all spanning tree for the given graph





How many spanning trees are there in K4



Spaning tree $(K_3) = 3$

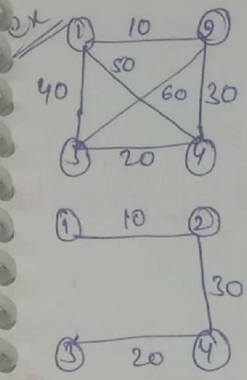
" " $(K_4) = 16$

" " $(K_5) = 125$

" " $K_n = n^{n-2}$

MINIMUM COST SPANNING TREE

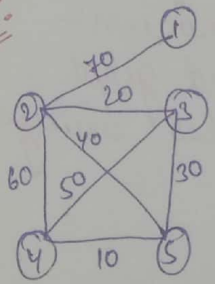
(min. edges and edges weights should also be minimum)



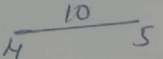
Using Prims and Krushkal algo we can find out min. cost spanning tree for the given graph very easily.

Kruskal's algo

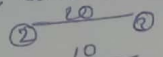
Ex:

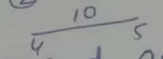


Step 1
* create min heap tree of edge weights } → O(E)
* Find min-cost-edge and add to MST } O(log(E))

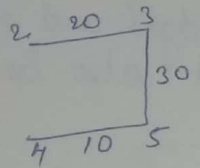


* step 2
Find next min-edge and add to MST } log(E)





* step 3.
Find next min-edge and add to MST
if no cycle



* step 4
Repeat step 3 — 3 log(E)

$40_{2,5}$ } forming cycle
$50_{4,3}$ } so rejected time wasted
$60_{2,3}$ }

$70_{1,2}$ } taken

In case of prims algo we will always get connected graph but in case of kruskal algo we may get disconnected graph in middle of algo but at last we will get connected graph

Note
In kruskal algo while generating minimum-spanning tree it can give disconnected graph in b/w while Prim's in between also always give connected graph.

assume $E \geq V-1$

Time Complexity

Best case
$E + (V-1)\log E$
$E + V\log E$
$E + V\log V$ (coz $\log E = O(\log V)$)

complete graph
$E > V\log V$
coz $E = V^2$

In null graph
$E < V\log V$

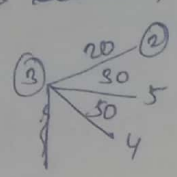
$O(E + V\log V)$

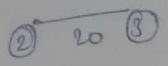
Worst case & AC
$E + E\log E$
$E < E\log E$
$E\log E$
$O(E\log V)$

In starting if we sort the array of edges cost then we will get → $E\log E$
so it would cost more.

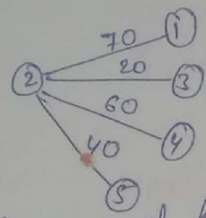
PRIMS ALGO

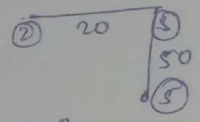
step 1
Choose any vertex and find adjacent of that vertex (how: chosen vertex will ask to all other vertexes by linear search) and ~~min~~ find min.





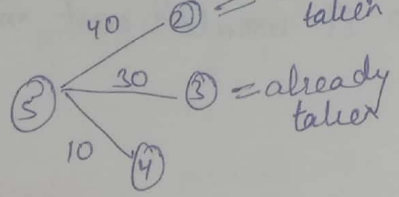
step 2
Find the adjacent of new vertex

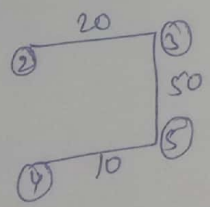




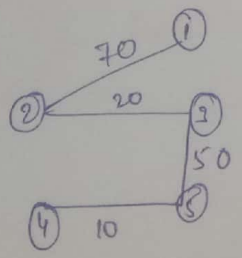
and take minimum of these and previous.
vertices

step 3
Find the adjacents of new vertex and take minimum of these and previous && add if no-cycle.

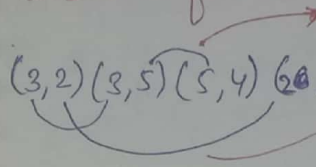




step 4
Repeat 3rd step

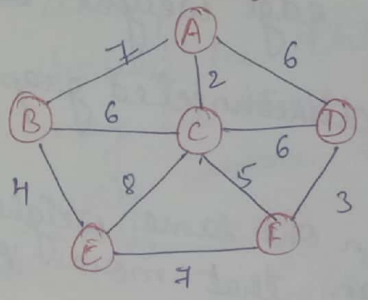


Note
The spanning tree found by kruskal and prims. may or may not be same (if 2 or more edges have same weight) but at last the sum of cost would be same for both the cases.

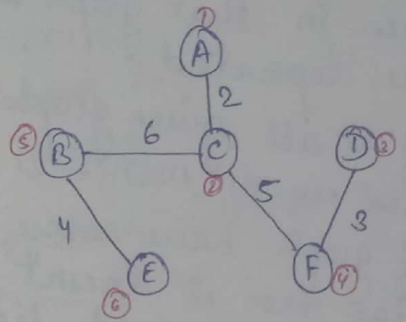


exam
① Always check first adjacency then min cost

Ques Consider the following graph.



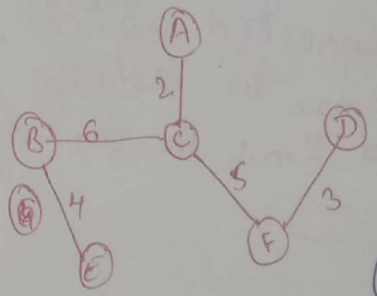
Kruskals



(A,C) (D,F) (B,E) (CF) (B,C

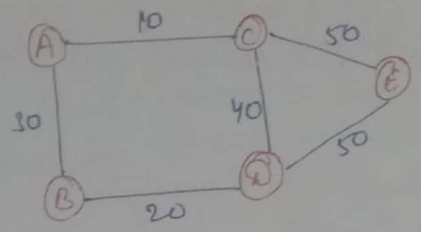
no connectivity property only cycle property

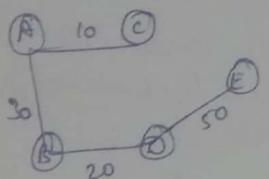
Prims

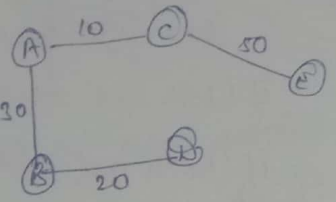


(A,C) (C,F) (F,D) (C,B) (B,E)

Ques Consider the following graph How many min cost spanning tree possible







Note:

* For the given graph more than one MST may be possible but the cost will be same
* If all e more than one MST is possible then in that graph some edge weight are repeated.
* If at all given graph is disconnected graph then no. of MST = 0
* If graph have more than one same weighted edge then it doesn't mean that more than one MST would be formed.

Ques Let G be a undirected connected weighted graph with n vertices and e_{max} be the edge with max edge weight and e_{min} be the edge with min edge weight.

True/False

(i) e_{min} should be there in MST of G. True
(ii) e_{max} should be there in MST of G False (coz of should be)
(c) e_{max} may be there in MST of G. True.
(d) G contain unique MST. True
(e) e. If e_{max} is in MST then its removal from G must disconnect G. True

Ques Let G be a undirected connected weighted graph with n vertices & w be the min. edge weight among all edge weights and e be a specific edge with weight w.

Soln True/False

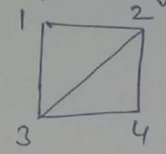
(i) e should be there in every MST of G. Fa
(ii) e may be there in some MST of G. Tru
(iii) Every ~~should~~ ~~be~~ ~~good~~ MST must contain an edge with weight w. True.
(iv) If MST not contain e then in that cycle all edge weight are w. True

Ques Consider the following graph

GRAPH REPRESENTATION

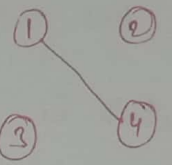
i) Adjacency Matrix
ii) Adjacency List

i) Adjacency Matrix



Best if
{ Dense graph
or
more edges

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



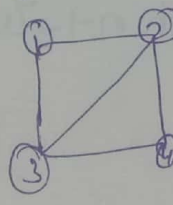
Sparse

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

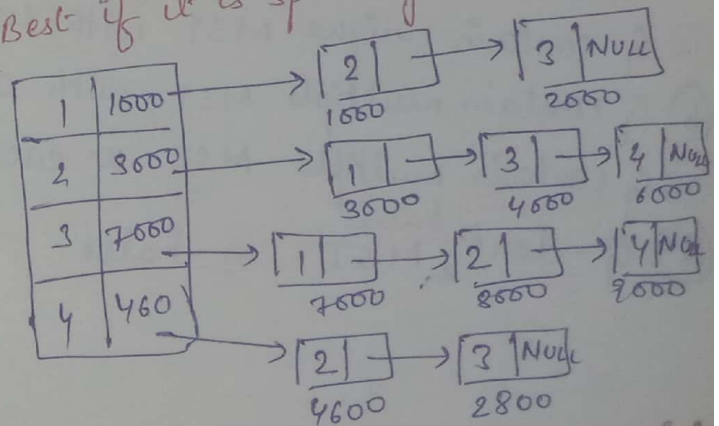
No best case in worst case to find degree of vertex = O(V)

If G(V, E) then to make matrix we will take $O(V^2)$ space

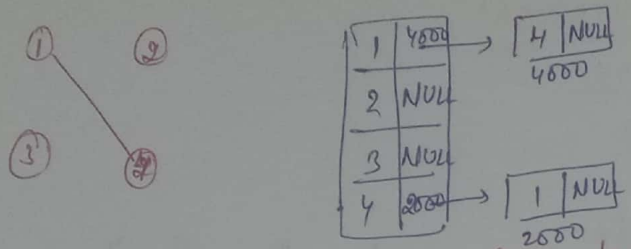
ii) Adjacency List (Best if it is sparse graph / less edges)



⇒



Best case to find the degree of any vertex is O(1)



Space taken by adjacency list for $G(V, E)$

$V + 2E$

↓ ↳ for linked list.

for array.

for complete graph $E > V$

for null graph $E < V$

Let G be a graph with n-vertices. whose adjacency matrix is given by $n \times n$ sq. matrices, in which

(i) All diagonal elements are 0's.

(ii) All non-diagonal elements are 1's.

Then check the following statements are true/false.

G contain unique MST with cost $n-1$ → False

G contain multiple MST with different cost → False

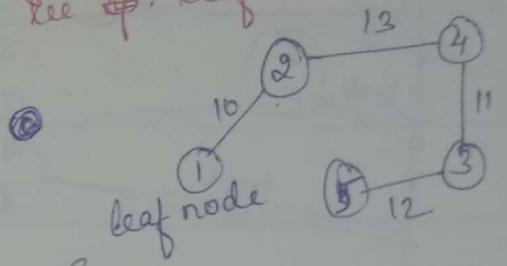
G contain multiple MST of each of cost $n-1$ → True

G - dont have MST → False

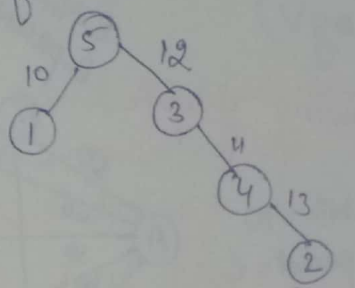
Ques Consider the following graph.

	1	2	3	4	5
1	0	10	10	10	10
2	10	0	15	13	17
3	10	15	0	4	12
4	10	13	11	0	14
5	10	17	12	14	0

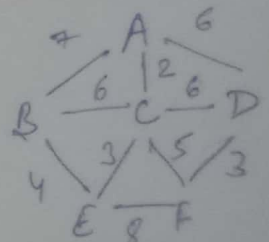
(i) What will be the cost of Min-Cost Spanning tree for the above graph where in that min-cost spanning tree will be $1-V$ leaf node.



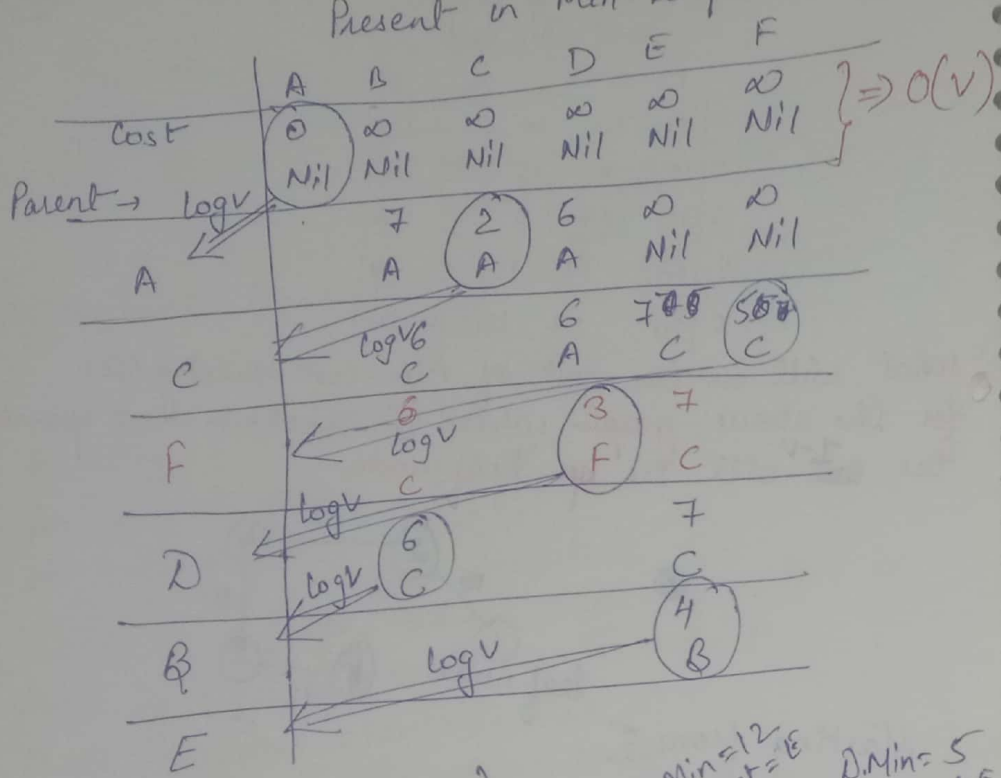
Starting from 5

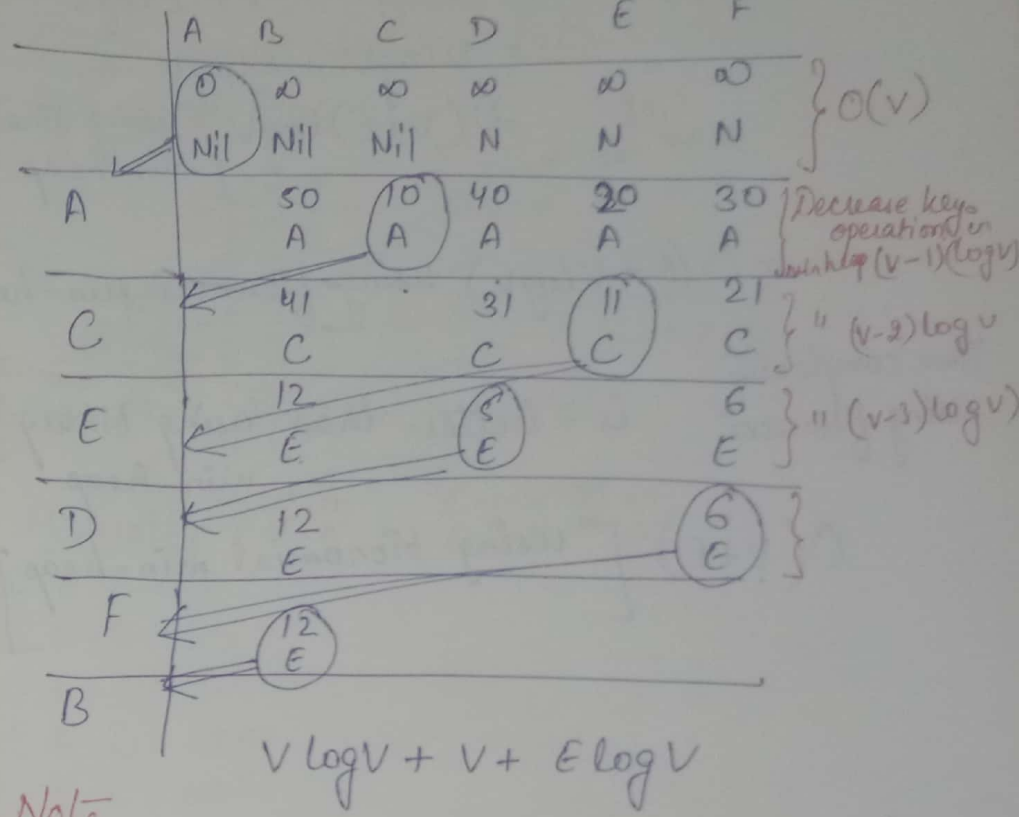


TIME COMPLEXITY OF PRIM'S ALGO [using min heap

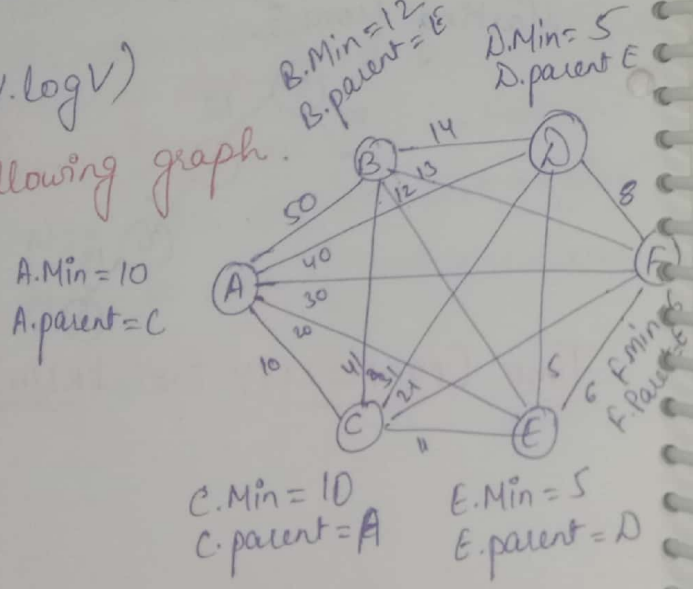


Present in Min-heap





Ques Consider the following graph.



Note

Increase key and decrease key operation in min-heap and max-heap will take O(logn) (worst case and Average case)

O(n) in best case

Also Priority Queue is also called Min-heap.

$(V-1)+(V-2)+(V-3)\ldots\ldots = E$ (Edges)

imp Time complexity (Prims) = $V\log V + V + E\log V$

$= V\log V + E\log V$

imp $= O(V+E)\log V$ [using binary min-heap]

$= O(E + V\log V)$ using fibnocci min-heap

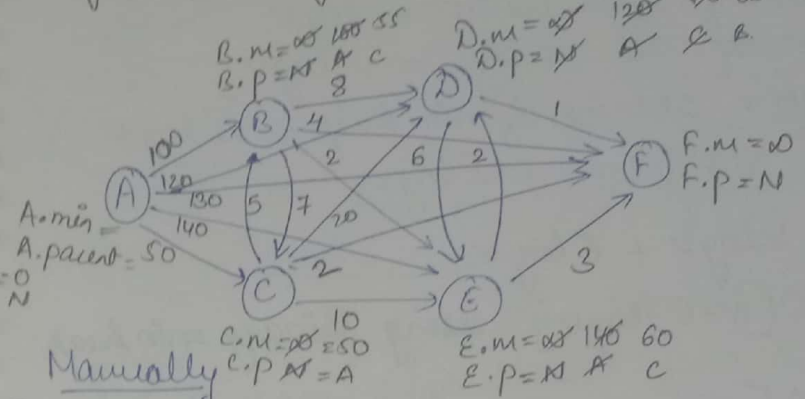
Time complexity using fibonacci is better than using binary min-heap.

$O(V+E)$ [using binomial min-heap]

ADA

Single Source Shortest Path

↳ Dijkstra's Algo



Manually

(a) A - A = 0

(b) A - B = 55

(c) A - C = 50

(d) A - D = ~~62~~ 59

(e) A - E = ~~60~~ 57

(f) A - F = 52

	A	B	C	D	E	F	
	0	∞	∞	∞	∞	∞	⇒ O(V)
logV	N	N	N	N	N	N	
		100	50	120	140	130	⇒ 5 logV
A		A	A	A	A	A	
logV		55		70	60	52	⇒ 4 logV
C		C		C	C	C	
logV		55		70	60		⇒ 0 logV
F		C		C	C		
logV				63	57		⇒ 2 logV
B				B	B		
logV					59		⇒ logV
E					E		
D							= E logV
V logV							

Time complexity of Dijkstra's algo.

* Initially to create min heap = $O(V)$
- Deletion of each vertex = $\log V$
- Deletion of all vertex = $V \log V$

After deletion ordering of min heap = $E \log V$

$T(Dij) = V + V\log V + E\log V$

$= V\log V + E\log V$

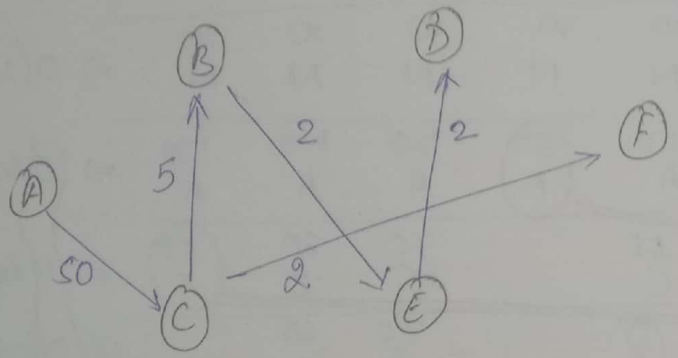
① $= O[(V+E)\log V]$ using Binary min heap.

② (not in syllabus) $= O[E + V\log V]$ using Fibonacci min heap

③ (not in syllabus) $= O(V+E)$ using binomial min heap.

④ By array.

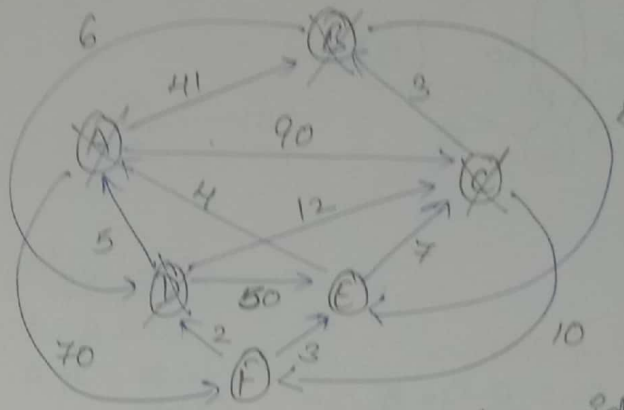
$O(V^2)$.



1) A-D (59)

2) A→C→B→E→D

Q Consider the following graph



A. min = ∞, 0 (1) P = N (0)

B. min = ∞, 90, 41 (2) P = N, A (A)

C. min = ∞, 90, 97, 47 (4) P = N, A (B)

D. min = ∞, 59 (3) P = N (A)

E. min = ∞, 121, 80, 77 (5) P = N (D)

F. min = ∞, 70, 69 (6) P = N, A (C)

(i) print the seq. of vertices identified by Dij's algo if when algo started from 'A'

(ii) A, B, D, C, F, E

(iii) what will be the cost of shortest path from A to E.

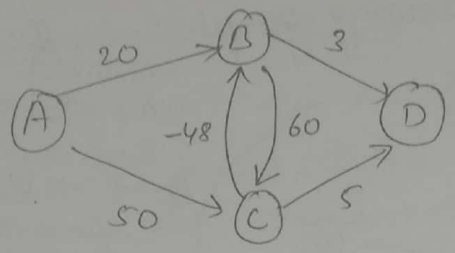
A - E (72)

(iv) what will be the shortest path from A - E

A → B → D → C → F → E

	A	B	C	D	E	F
	0 N	∞ N	∞ N	∞ N	∞ N	∞ N
A		41 A	90 A	∞ N	∞ N	70 A
B			90 A	47 B	121 B	70 A
D			59 D		97 D	70 A
C					97 D	69 C
F					72 F	

Ques



Manually

A - A = 0
A - B = 2
A - C = 50
A - D = 5 — A-C-B-D = 5
A-C-D = 55.

Dijkstra's

	A	B	C	D
	0 N	∞ N	∞ N	∞ N
A		20 A	50 A	∞ N
B			50 A	23 B
D				50 A
C				

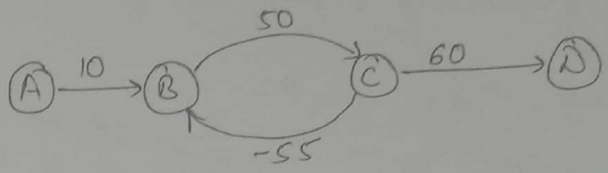
A-B-D-C
20+3+

A - B = 20
A - C = 50
A - D =

Note:

(a) If the graph contain negative edge wts Dijkstra's algo may give wrong ans.
(b) If the graph contains all positive edge wts then Dijkstra's algo always give correct ans.

Ques.



Manually

A - A = 0
A - B = 5, 0, -5, -10 ... $-\infty$ coz of negative wt edge.
A - C = 60 ... $-\infty$ (")
A - D = 120 ... $-\infty$ (")

By Dijkstra's

	A	B	C	D
	0 N	∞ N	∞ N	∞ N.
A		10 A	∞ N	∞ N
B			60 B	∞ N
C				120 C
D				

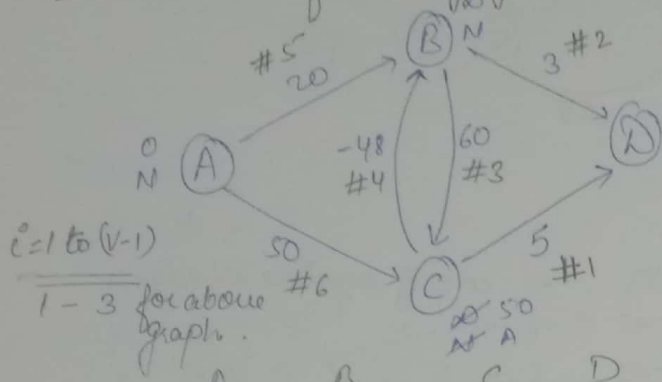
A - A = 0
A - B = 10
A - C = 60
A - D = 120

Note:

If the graph contains negative edge wt. cycle then Dijkstra algo always will fai

BELLMAN FORD ALGO

Consider the following graph



using array. Bellman ford is solved.

	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	20	50	∞
	N	A	A	N
i=2	0	2	50	23
	N	C	A	13.
i=3	0	2	50	5
	N	C	A	B

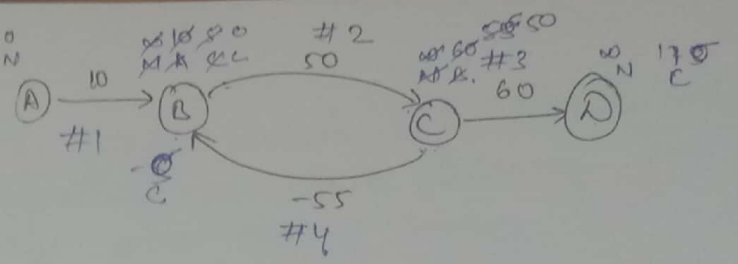
E, E, E → O(VE)

For all the graphs → O(VE)

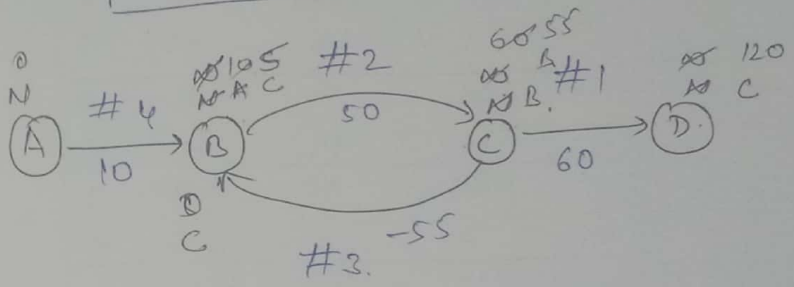
If graph is complete then the complexity $= O(V^3)$

Max complexity of Dijkstra algo $= VE \log V$
$= V^2 \log V$

While in case of Bellman ford algo the Max complexity $= O(V^3)$



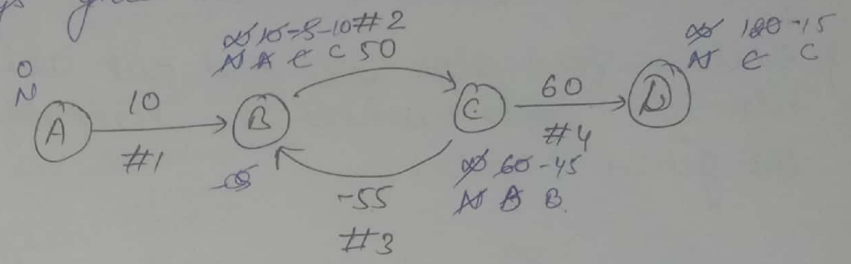
	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	5	60	120
	N	C	B	C
i=2	0	0	55	115
	N	C	B	C
i=3	0	-5	50	110
	N	C	B	C



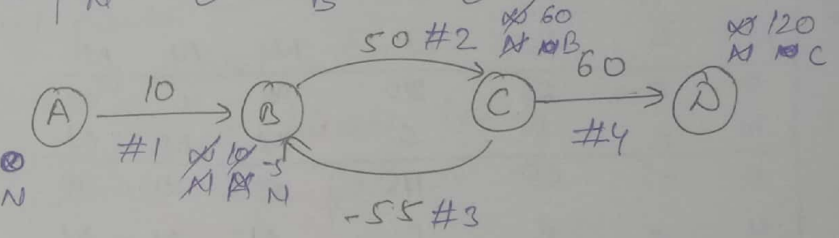
	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	10	∞	∞
	N	A	N	N
i=2	0	5	60	∞
	N	C	B	N
i=3	0	0	55	120
	N	C	B	C

Note:
If the graph contain all positive edge wt or negative wt. then Bellman ford will always give correct answer.

Ques



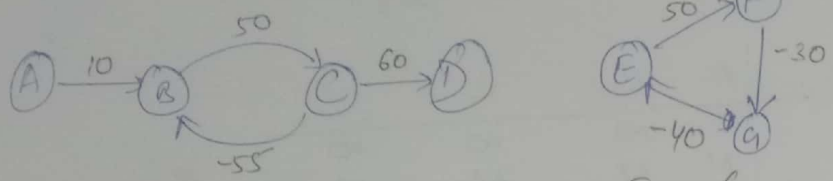
	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	10	∞	∞
	N	A	N	N
i=2	0	-5	60	120
	N	C	B	C
i=3	0	-10	-45	-15
	N	C	B	C



	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	-5	60	120
	N			N
i=2				
i=3				

Note — Bellman ford → Dynamic prog. (more time) (always give correct ans)

* If the graph contain negative edge wt. cycle then also bellman ford algo will give correct ans.
* Bellman ford algo will find out all negative edge wt. cycle which are reachable from the given cycle.



	A	B	C	D	E	F	G
	0	∞	∞	∞	∞	∞	∞
	N	N	N	N	N	N	N
i=1	0	10	∞	∞	∞	∞	∞
	N	A	N	N	N	N	N
i=2	0	5	60	∞	∞	∞	∞
	N	C	B	N	N	N	N
i=3	0	0	55	120	∞	∞	∞
	N	C	B	C	N	N	N
i=4	0	-5	50	115	∞	∞	∞
	N	C	B	C	N	N	N

decreasing with same diff. i.e. 5 means part of same cycle

decreasing with same diff. i.e. ∞ means part of same cycle

Dijkstra algo → Greedy algo (less time)
(some time may give wrong ans)

Dynamic Programming

Greedy Technique → Sometimes wrong answer.

Dynamic Prog → Always correct answer

G.T – few possibilities will be covered
D.P → All possibilities " " "

G.T – less time
D.P – More time

Applications of Dynamic Programming

1. Fibonacci series
2. longest common subsequence
3. Matrix Multi chain multiplication
4. Travelling salesman problem
5. 0/1 knapsack
6. All pairs shortest path
7. Sum of subset
8. Optimal cost binary search tree.
9. Optimal merge pattern

Fibonacci Series

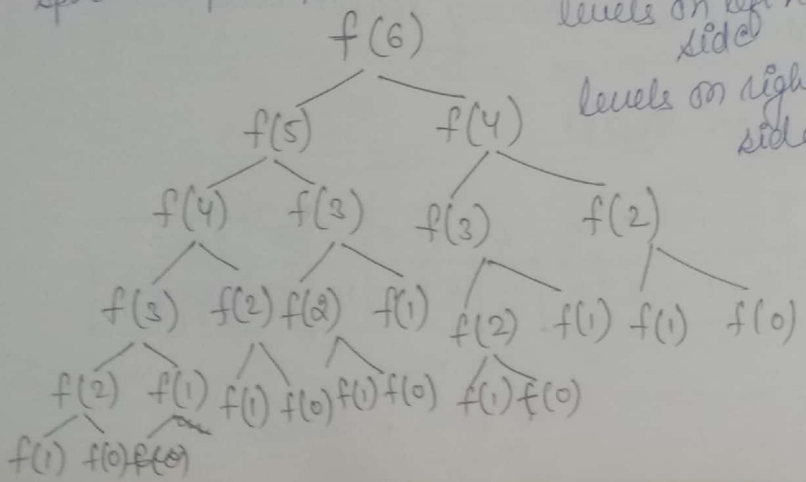
n	0	1	2	3	4	5	6	7	8	9	10
fib(n)	0	1	1	2	3	5	8	13	21	34	55

Recurrence Relation: $f(n) = \begin{cases} n & \text{if } n=0 \text{ or } n=1 \\ f(n-1)+f(n-2) & \text{if } n>1 \end{cases}$

Recursive prog

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

* For time complexity we need recursive relation of time of recursive prog.
* For a single recursive prog we can write many recursive prog like for time complexity, space complexity, value etc.



levels on left hand side = 6

levels on right hand side = 4

$f(n)$ = n-level complete binary tree (upper bound)

$= 2^n - 1$ nodes

$= 2^n - 1$ function call

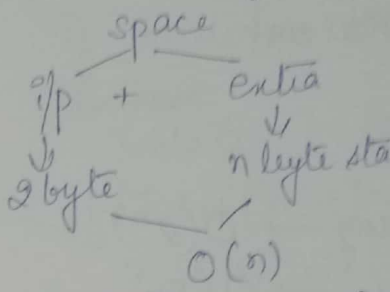
$= 2^n \times 1$ function calling cost $\Rightarrow 2^n \times 1$-addition

$= 2^n \times O(1) = O(2^n)$

Time complexity $= O(2^n)$ upper bound

Space: i/p (only one i/p i.e. 'n' of 2 bytes)

extra (stack = no. of level of tree



Brute force & Dynamic prog will always give correct answer

⇒ In the above recursive tree many function calls are repeated (overlapping subproblems) So what is the need of executing the same function again and again.

⇒ In the case of dynamic programming we will compute only distinct function calls. because as soon as we compute

any func. we will store its value in table so that we can re-use it further whenever it is needed.

imp Ques How many distinct func.n calls are there in fibanocci of 'n'?

Ans $f(6) \Rightarrow f(5), f(4), f(3), f(2), f(1), f(0)$

$= 6+1$ calls.

$f(n) = n+1$ distinct func. calls.

Time $= (n+1) * (\text{one func. call cost})$

$= n * 1$-addition cost

$= n * O(1)$

$= O(n)$.

without dynamic prog $\rightarrow O(2^n)$

with " " $\rightarrow O(n)$

Space using dynamic prog $\rightarrow$ i/p + extra

$= 2$ Byte + stack + Table

$= 2$ Byte + n Bytes + $(n+1)$ Byte (for storing distinct func. call)

$= 2n$ Bytes

$O(n)$

without dynamic prog $= O(n)$

Dynamic prog will give more level tree than Divide & conquer

In D&C we don't bother about distinct func. call whereas in Dynamic Prog we do.

Fibonacci prog with dynamic Prog

Table → global array.

```
fast-fib(n)
{
  if (n==0 || n==1)
    return (n)

  else
  {
    if (Table[n-1]==null)
      Table[n-1] = fast-fib(n-1);
    if (Table[n-2]==null)
      Table[n-2] = fast-fib(n-2);
    Table[n] = Table[n-1] + Table[n-2]
    return (Table[n]);
  }
}
```

LONGEST COMMON SUBSEQUENCE (LCS)

Subsequence of a given sequence is just the given sequence only. in which zero or more symbols are left out.

ex:

$S = (A, B, B, A, B, B)$
$\quad\quad 1\ 2\ 3\ 4\ 5\ 6$

$SS_1 = (B, B, B, B)$
$\quad\quad 2\ 3\ 5\ 6$

$SS_2 = (A, A)$
$\quad\quad 1\ 4$

$SS_3 = (B, A, B, A)$ X some other sequence
$\quad\quad 2\ 4\ 5\ ?$

$SS_4 = ( )$

$SS_5 = (A, B, B, A, B, B)$

$SS_6 = (A, B, A, B)$
$\quad\quad 1\ 2\ 4\ 5$

$SS_7 = (A, A, B, B)$ X

Common subsequence : Z is a common subsequence of ~~two~~ sequences x & y iff z is subsequence to x & Z is subsequence to y only.

ex:
$\quad\quad 1\ 2\ 3\ 4\ 5\ 6$
$x = (A, B, B, A, B, B)$

$y = (B, A, A, B, A, A)$
$\quad\quad 1\ 2\ 3\ 4\ 5\ 6$

$Z_1 = (A, A)$

$Z_2 = ( )$

$Z_3 = (A, B, A)$

$Z_4 = (B, A, B, A)$ X

$Z_4 = (A)$

4 length common subsequence is not possible
length is longest
so 3 is longest common subsequence.

Ques
$\quad\quad 1\ 2\ 3\ 4\ 5\ 6\ 7$
$x = (A, B, B, A, B, B, A)$

$\quad\quad 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
$y = (B, A, B, A, B, A, B, A)$.

0 - length
$Z_1 = ( )$

1 - length
$Z_2 = (A)$

2 - length
$Z_3 = (AB)$

3 - length
$\quad\quad 1\ 2\ 4$
$Z_4 = (A\ B\ A)$
$\quad\quad 2\ 3\ 4$

~~5~~ - length
$Z_5 = (A, B, A, B, A)$

~~4~~ - length
$Z_6 = (A, B, AB)$

6 - length
$Z_7 = (B, B, A, B, B, A)$

longest common subsequence is of length 6

How to find out LCS using Dynamic programming

ex1 $x = (A, A, B, A, B)$ $\quad 1\ 2\ 3\ 4\ 5$

$y = (B, A, A, B, A)$ $\quad 1\ 2\ 3\ 4\ 5$

$LCS(4,5) = 1 + LCS(3,4)$

$= 1 + LCS(2,3)$

$= 1 + LCS(1,2)$

$= 1 + LCS(0,1)$

$= 1 + 0$

$= 1$

4 Ans

ex2 $x = (A\ B\ A\ A\ B\ A\ B)$ $\quad 1\ 2\ 3\ 4\ 5\ 6\ 7$

$y = (B\ B\ A\ A\ B\ B\ A\ B)$ $\quad 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$

$LCS(7,8) = 1 + LCS(6,7)$

$= 1 + LCS(5,6)$

$= 1 + LCS(4,5)$

$\max \{ LCS(3,5), LCS(4,4) \} \Rightarrow 4$ (3, 4)

$LCS(7,8) = 7$ Ans

Note

LCS of (m, n)

LCS(m, n) = length of the longest common subsequence possible with two seq. (x, y) where x contains m symbols and y contains n symbols.

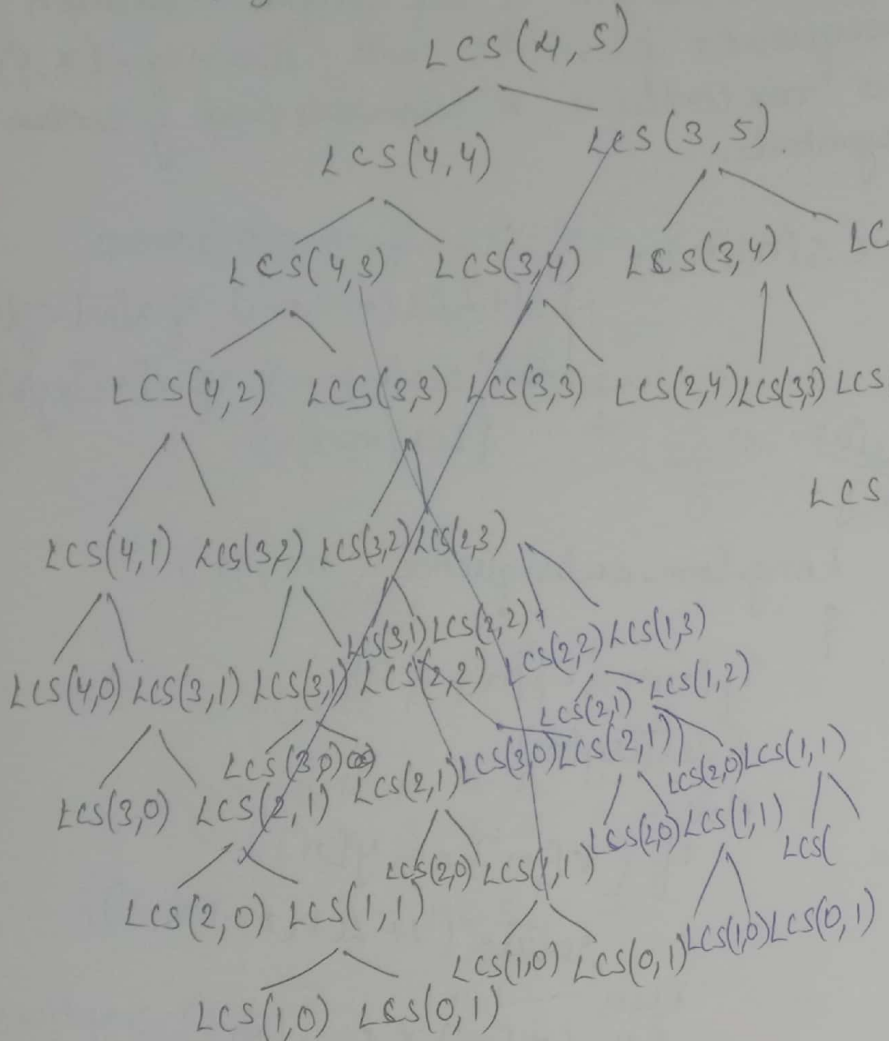
$$LCS(m,n) = \begin{cases} 0 & \text{if } m=0 \text{ or } n=0 \\ 1 + LCS(m-1, n-1) & \text{if } x[m] == y[n] \\ \max \begin{cases} LCS(m, n-1) \\ LCS(m-1, n) \end{cases} & \text{if } x[m] \neq y[n] \end{cases}$$

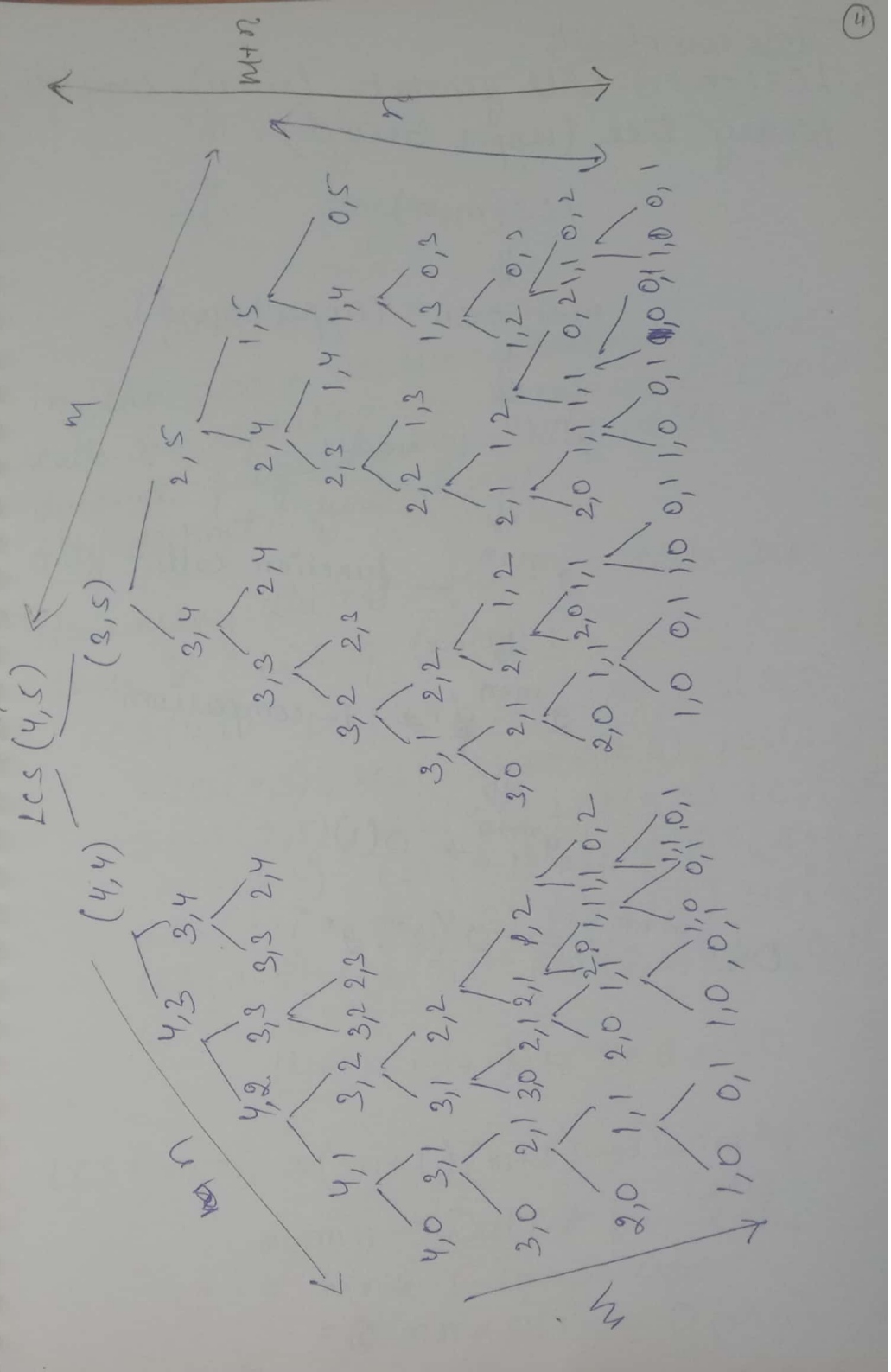
Recursive Prog

```
Long-Com-subsequence (m, n)
{
  if(m==0||n==0)
    return(0)
  else
    if (x[m]==y[n])
      return (1+LCS(m-1,n-1))
    else
    {if (x[m]!=y[n]) => optional
      a = LCS (m, (n-1));
      b = LCS (m-1,(n-1));
    } c = max(a,b)
    return c;
}
```

x = A A A A
    1 2 3 4

y = B B B B B
    1 2 3 4 5





Time complexity

LCS(m,n) will generate (m+n)-complete binary tree (upper bound)

LCS(m,n)

⇓

m+n → CBT (upper bound)

level

⇓

$2^{m+n}-1$ nodes.

⇓

2^{m+n} — function calls.

⇓

2^{m+n} * 1-comparison

⇓

2^{m+n} * O(1)

$O(2^{m+n}) \Rightarrow O(2^m \cdot 2^n)$

Space —

i/p + extra

⇓ ⇓

m & n stack

⇓

m+n

$\Rightarrow O(m+n)$.

In the above recursive tree many func^n calls are repeating so ~~many~~ we will go to dynamic programming which will solve only distinct function call.

imp Try How many distinct function calls are there in LCS of (m,n)?

LCS(4,5) = LCS(4,4), LCS(3,4), LCS(3,5),

= LCS(4,3), LCS(3,3) LCS(2,3), LCS(3,4)

= LCS(2,5), LCS(4,2) LCS(3,2), LCS(1,5)

LCS(2,4), LCS(4,1) LCS(3,1) (LCS(2,4))

LCS(1,4) LCS(1,3) (LCS(1,2) LCS(0,5)

LCS(0,4) LCS(0,3) LCS(0,2) LCS(0,1)

LCS(0,0)

(4+1) * (5+1) = 5 × 6 = 30

LCS(m,n) = (m+1)(n+1) → Distinct func. calls.

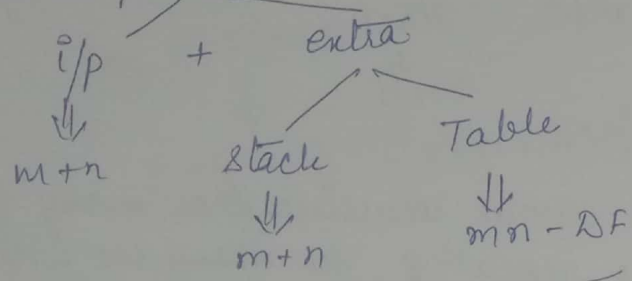
= mn → Distinct func calls.

= mn * 1-comparison

= mn * O(1) = O(mn)

$O(mn)$ → with Distinct func. call.

Space complexity



3. 0/1 KNAPSACK

ex: M=10

obj:	ob_1	ob_2	ob_3
profits:	70	38	58
weights:	7	4	6

For 0/1 knapsack problem greedy algo will fail because of this reason we are going for dynamic programming. which will cover every possibility exactly one time.

ex2 n=5

ob:	ob_1	ob_2	ob_3	ob_4	ob_5
profit	25	75	15	45	35
wt	5	3	2	1	2.

⇓

1) $0/1ks(n,m) = 0$ if $0,m = 0$, $n,0 = 0$, $0,0 = 0$

2) $0/1ks(n,m) = 0/1ks(n-1,m)$ if $w_n > m$.

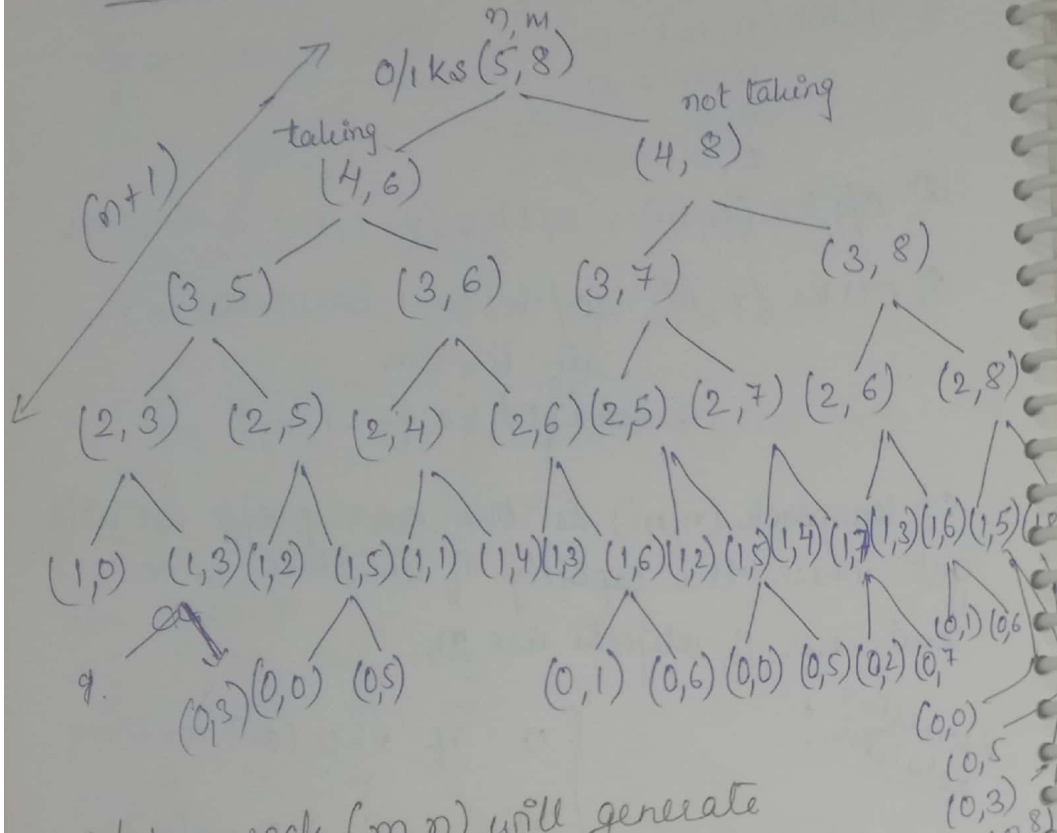
3) $0/1ks(n,m) = \begin{cases} 0/1ks(n-1, m-w_n) + P_n \\ \quad \text{if } w_n \le m \\ \max\{0/1ks(n-1,m)\end{cases}$

0/1 knapsack (n,m) is the max. profit we will get where the capacity of knapsack is m and no. of objects are n.

Recursive Relation:

$$0/1ks(n,m) = \begin{cases} 0 & \text{if } n=0 \text{ (or) } m=0 \\ 0/1ks(n-1,m) & \text{if } w[n] > m \\ \max\begin{cases} 0/1ks(n-1, m-w[n]) + P[n] \\ \qquad \text{if } w[n] \le \\ 0/1ks(n-1,m) \end{cases} \end{cases}$$

Recursive Tree



0/1 knapsack (m,n) will generate

⇓

n+1 — complete Binary Tree (upper bound)

⇓

2^{n+1} — 1 nodes.

⇓

2^n — function.

⇓

2^n * 1 - comparison

⇓

$2^n * O(1) \Rightarrow O(2^n)$ without Dynamic prog.

Here Structure is used to store the information.

Space:

i/p + extra

⇓

n + stack

⇓

n+1

O(n) without dynamic prog.

In the above recursive tree some (very less) function calls are repeating. because of this reason we are going for dynamic prog. which will call only distinct func call

imp How many distinct func.n call are there in 0/1 knapsack? with Dynamic prog.

Ans 0/1 knapsack (n, m)

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

(n+1) (n+1)

(n+1).(n+1) — Distinct func. call

⇓

mn — Distinct Func call.

⇓

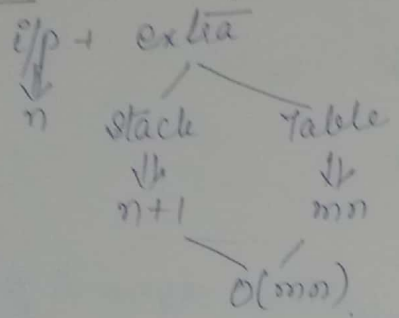
mn * O(1)

⇓

O(mn)

Time:

Space: (with Dynamic Prog)



In 0/1 knapsack problem because of very less repeatations time complexity O(mn) is almost equal to $O(2^n)$ so, it is one of the N-P complete problem.

problem taking more the polynomial time is called NP complete.

Note-2

Sum of subset problem is polynomialy reducable to 0/1 knapsack pro

Note-2

0/1 knapsack is polynomial time reduce-able to sum of subset. So sum of subset is also N.P complete problem.

(If one problem is reducable to another type of problem means that they are similar)

MATRIX CHAIN MULTIPLICATION

matrices

Ex: $A_{2\times4}$ $B_{4\times3}$

$C_{2\times3} = A_{2\times4} \times B_{4\times3}$

$A = \begin{bmatrix} - & - & - & - \\ - & - & - & - \end{bmatrix}_{2\times4}$ $B \begin{bmatrix} - & - & - \\ - & - & - \\ - & - & - \\ - & - & - \end{bmatrix}_{4\times3}$.

multiplication

$C \begin{bmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \end{bmatrix}_{2\times3} \Rightarrow$ 2×3 elements

each element ⇓ multiplication

2 * 3 * 4 ⇓

24 multiplication

Ex 2

$A_{10\times2}$ $B_{2\times5}$

C = AB

⇓

10×5 element

⇓

10×5×2 multiplications

= 100 mul

Ex 3

$A_{2\times4}$ $B_{4\times5}$ $C_{5\times3}$.

$AB_{2\times5} \times C_{5\times3} \Rightarrow ABC_{2\times3}$

2×5×4 + 2×5×3.

40 + 30 = 70 multiplication.

ex 4

$A_{2\times3}\ B_{3\times5}\ C_{5\times2}\ D_{2\times4}$

$E = ABCD$

$((AB)C)D$

$AB_{2\times5} = 2\times5\times3 = 30$ mul

$(ABC)_{2\times2} = 2\times5\times2 = 20$ mul.

$(ABCD)_{2\times4} = 2\times2\times4 = 16$ mul

66 mul.

$(A(B(CD)))$

$(CD)_{5\times4} = 5\times4\times2 = 40$ mul

$(BCD)_{3\times4} = 3\times4\times5 = 60$ mul.

$(ABCD)_{2\times4} = 2\times4\times3 = 24$ mul

124 mul.

$(A((BC)D))$

$BC_{3\times2} = 3\times5\times2 = 30$ mul

$BCD_{3\times4} = 3\times2\times4 = 24$ mul.

$ABCD_{2\times4} = 2\times3\times4 = 24$ mul

78 mul.

$((A(BC))D)$

$BC_{3\times2} = 3\times2\times5 = 30$ mul

$ABC_{2\times2} = 2\times3\times2 = 12$ mul.

$ABCD_{2\times4} = 2\times2\times4 = 16$ mul.

~~58~~ 58 mul

$((AB)(CD))$

$AB_{2\times5} = 2\times3\times5 = 30$ mul

$CD_{5\times4} = 5\times2\times4 = 40$ mul.

$ABCD_{2\times4} = 2\times5\times4 = 40$ mul

110 mul.

ex 5

$A_{1\times2}\ B_{2\times1}\ C_{1\times3}\ D_{3\times1}$ ABCD

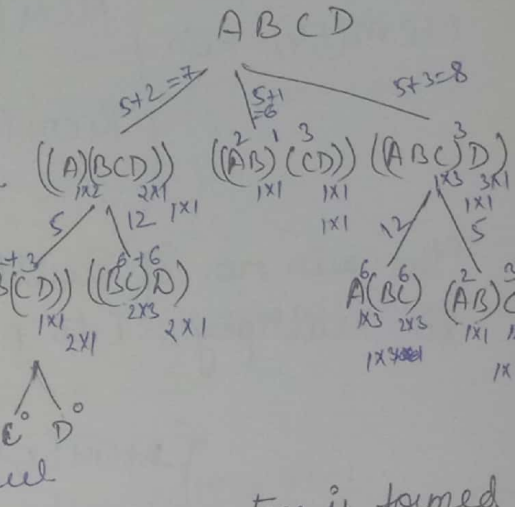
$((AB)C)D$

$AB_{1\times1} = 1\times2\times1 = 2$

$ABC_{1\times3} = 1\times3\times1 = 3$

$ABCD_{1\times1} = 1\times1\times3 = 3$

8 mul



$(A(BC))D$

$BC_{2\times3} = 2\times3\times1 = 6$

$ABC_{1\times3} = 1\times3\times2 = 6$

$ABCD_{1\times1} = 1\times1\times3 = 3$

15 mul

$((AB)(CD))$

$AB_{1\times1} = 1\times2\times1 = 2$

$CD_{1\times1} = 1\times3\times1 = 3$

$ABCD_{1\times1} = 1\times1\times1 = 1$

6 mul.

$(A((BC)D))$

$BC_{2\times3} = 2\times3\times1 = 6$

$BCD_{2\times1} = 2\times1\times3 = 6$

$ABCD_{1\times1} = 1\times1\times2 = 2$

14 mul

$A(B(CD))$

$CD_{1\times1} = 1\times3\times1 = 3$

$BCD_{2\times1} = 1\times2\times1 = 2$

$ABCD_{1\times1} = 1\times1\times2 = 2$

7 mul

Multiplication of 4 matrices (ABCD) $\rightarrow P_1 * P_4$

$$MCM(1,4) = \min \begin{cases} MCM(1,1) + MCM(2,4) + P_0 * P_1 * P_4 \\ MCM(1,2) + MCM(3,4) + P_0 * P_2 * P_4 \\ MCM(1,3) + MCM(4,4) + P_0 * P_3 * P_4 \end{cases}$$

($P_0 * P_1$, $P_1 * P_2$, $P_3 * P_4$; $P_0 * P_2$, $P_2 * P_4$)

The min no. of multiplications required to multiply i to j matices :-

$$MCM(i,j) = \begin{cases} 0 & \text{if } (i=j) \\ \min_{i \le k \le j-1} \begin{cases} MCM(i,k) + MCM(k+1,j) + P_{i-1} * P_k * P_j \end{cases} & \text{if } i<j \end{cases}$$

($P_{i-1} * P_k$, $P_k * P_j$)

'MCM(1, n) will generate n-level n-array tree (upperbound)

Total func^n calls $= (n-1)(n-2)(n-3)\ldots 1$

$= (n-1)!$

Time complexity $= (n-1)! \times$ cost of 1 func call

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

$= n! = O(n^n)$

To find the min b/w 23 $\rightarrow$ 2-1 comparison

To find the min b/w 100 = 100-1 "

So % " " " " n = n-1

coz of this cost of 1 func call = $n-1$

$= O(n-1) \equiv O(n)$

Space : without

i/p + extra

n n level tree

n n

$O(n)$

imp

In the above recursive tree many func calls are repeating. so we will go to dynamic prog.

Ques How many distinct func calls are there in MCM(i, j)?

Soln MCM(i, j)

change n way change n ways

n^2 – DFC.

Time $= n^2 \times n = O(n^3)$

Space (with dynamic prog)

i/p + extra

i/p → n

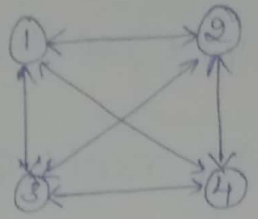
extra → stack → n

extra → Table → n^2 – Distinct Function call

$O(n^2)$

TRAVELLING SALESMAN PROBLEM

ex



	1	2	3	4
1	0	10	20	30
2	5	0	70	80
3	7	60	0	50
4	6	15	25	0

$$TSP(1,\{2,3,4\}) = \min \begin{cases} C(1,2)+TSP(2,\{3,4\}) \\ C(1,3)+TSP(3,\{2,4\}) \\ C(1,4)+TSP(4,\{2,3\}) \end{cases}$$

$$TSP(1,\{2,3,4\}) = \min \begin{cases} C(1,2)+C(2,3)+C(3,4)+C(4,1) \\ C(1,2)+C(2,4)+C(4,3)+C(3,1) \\ C(1,3)+C(3,2)+C(2,4)+C(4,1) \\ C(1,3)+C(3,4)+C(4,2)+C(2,1) \\ C(1,4)+C(4,2)+C(2,3)+C(3,1) \\ C(1,4)+C(4,3)+C(3,2)+C(2,1) \end{cases}$$

$$\min \begin{cases} 10+30+50+6 = 96 \\ 10+80+25+7 = 122 \\ 20+60+80+6 = 166 \\ 20+50+15+5 = 90 \\ 30+15+70+7 = 122 \\ 30+25+60+5 = 120 \end{cases}$$

$$TSP(2,\{3,4\}) = \min \begin{cases} C(2,3) + TSP(3,\{4\}) \\ C(2,4) + TSP(4,\{3\}) \end{cases}$$

$TSP(3,\{4\}) = C(3,4) + TSP(4,\phi)$

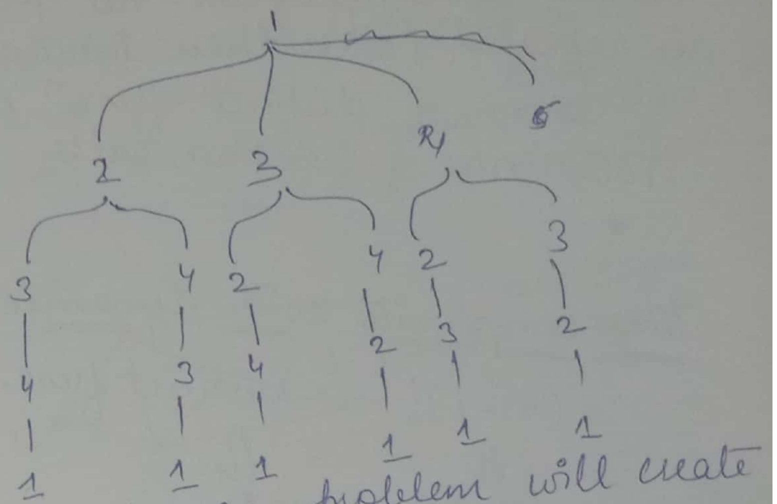
$TSP(4,\{3\}) = C(4,3) + TSP(3,\phi)$

$TSP(4,\phi) = C(4,1)$

$TSP(A,R)$ = Min. cost required to go from 'A' to all other remaining vertices in R exactly once and comoning back to source S'.

Recurrence Relation

$$TSP(A,R) = \begin{cases} C(A,S) & \text{if } R = \phi \\ \min \{ C(A,K) + TSP(K, R-K) \} & \forall K \in R \end{cases}$$



Travelling salesman problem will create n-level n-array tree (uperbound)

Total no. of function calls $= (n-1)(n-2)\ldots 1$

$= (n-1)!$

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

$= n! \Rightarrow O(n^n)$

Space = i/p + extra

i/p ↓ n^2 (matrix

extra ↓ stack ⇒ n level

$O(n^2)$

In the above problem no. function call is repeated (other than terminations)
So no. of distinct func^n call = Total no. of function calls.

Time complexity with dynamic Prog

$(n-1)! \longrightarrow$ Distinct func. call.

$(n-1)! * n$

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

Space $\Rightarrow$ i/p + extra

i/p $\Rightarrow n^2$

extra $\Rightarrow$ stack + Table

stack $\Rightarrow n$

Table $\Rightarrow (n-1)! = O(n^n)$

no repeation so elements just storing but not used again

$\Rightarrow O(n^n)$.

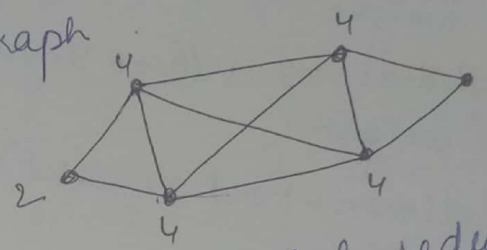
Note:

TSP is one of the NP-complete problem.

Hamiltonian graph/cycle is also a NP-complete problem. $\longrightarrow$ traversing every vertices edge of a graph exactly once & coming back to source

Euler graph $\longrightarrow$ cover every edge of a graph traversed exactly once and come back to source point

If a graph have only positive even (degree) edges. of every vertices. then it is definitly a eular graph

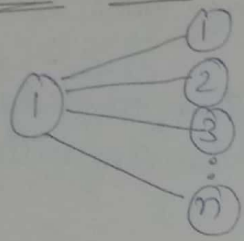


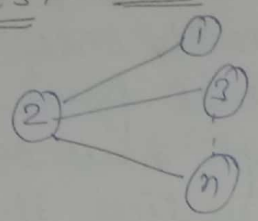
even degree edges.

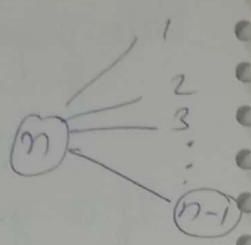
TSP is polynomial reduced to Hamiltonian graph. So checking Halmiltonian graph is a Hamiltonian graph

Checking given graph is Eular graph or not? $O(v^2)$ algo is possible so it is P-class problem.

6. ALL PAIRS SHORTEST PATH







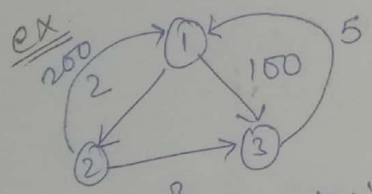
① +ve edge weight
$|V| *$ Dijkstra's $\Rightarrow V((V+E)\log V)$

② -ve edge weight
$|V| *$ Bellman-Ford $\Rightarrow V(VE)$

③ Unweighted Graph
$|V| *$ BFT $\Rightarrow V(V+E)$

④ Floyd warshall Algo
$\Rightarrow O(V^3)$ for +ve wt or for -ve wt

Floyd-Warshalls Algo



$\Rightarrow$

A^0 → directly.

A^0	1	2	3
1	0	2	100
2	200	0	3
3	5	∞	0

↑ no edge b/w (3→2)

3 → b/w 2 vertex, '1' no. of vertex

A'	1	2	3
1	0	2	100
2	200	0	3
3	5	7	0

$A'(3,2) = \min\{A^0(3,2), A^0(3,1)+A^0(1,2)\}$
$= \min\{\infty, 5+2=7\}$
$A'(3,2) = 7$

A^2	1	2	3
1	0	2	5
2	200	0	3
3	5	7	0

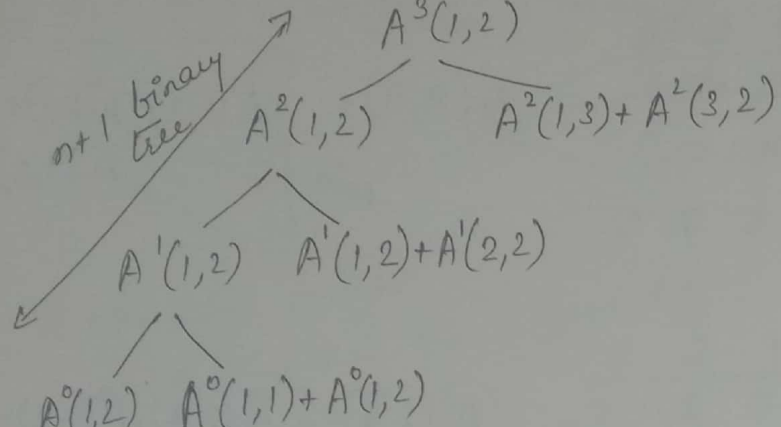
$A^2(2,3) = \min\{A^1(2,3), A^1(2,1)+A^1(1,3)\}$

A^3	1	2	3
1	0	2	5
2	8	0	3
3	5	7	0

$A^3(2,3) = \min\{A^2(2,3), A^2(2,3)+A^2(3,1)\}$
$= \min\{\underbrace{200}_{A^2(2,3)}, \underbrace{5+3}_{A^2(2,3)+A^2(3,1)}\}$
$= 8$

Let $A^k(i,j)$ = the min. cost required to go from vertex i to vertex j to which all intermediate vertices present in the set $\{0 \ldots k\}$

$$A^k(i,j) = \begin{cases} 0 & \text{if } (i=j) \\ \min\{A^{k-1}(i,j),\ A^{k-1}(i,k)+A^{k-1}(k,j)\} & \end{cases}$$



Floyd Warshalls algo on n-vertices generates $(n+1)^{th}$ level binary tree Then no. of func? call

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

$= 2^n * O(1)$

Time $= O(2^n)$

Space = i/p + extra

n^2 , $n+1$ tree

$O(n^2)$

In the above recursive tree some repeatition is there so we will go for dynamic prog. which will call distinct func call.

Ques How many distinct func call are there in $A^k(i,j)$

Soln $A^k(i,j)$

n n n

n^3 Distinct func. calls.

$n^3 * O(1) = O(n^3)$

Space = i/p + extra

n^2 , stack n^2 , Table

n tables of size n^2

$= n \times n^2 = n^3$

$O(n^3)$

Note

In floyd warshell., instead of storing every time in the new table, store in the same table, with this modification

space = i/p + extra

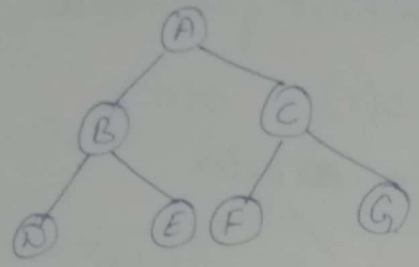
n^2 , only stack n

$O(n^2)$

$A^k(i,j) = \begin{cases} 0 & \text{if } i==j \\ \min\{A^0(i,j),\ A^0(i,k)+A^0(k,j)\} \end{cases}$

TREE TRAVERSAL & GRAPH TRAVERSAL

1. Preorder (Root, Left subtree, Right subtree)
2. Postorder (LST, RST, Root)
3. Inorder (LST, Root, RST)



Preorder : A B D E C F G

Postorder : D E B F G C A

Inorder : D B E A F C G

preorder (root) $\Longrightarrow T(n)$

```
{
    pf (root → data)              ⟹ O(1)
    preorder (root → left)        ⟹ T(n/2)
    preorder (root → right)       ⟹ T(n/2)
}
```

$$T(n) = 2T(n/2) + \text{Constant}$$
$$= O(n)$$

```
postorder (root)
{   postorder (root → left)
    postorder (root → right)
    pf (root → data)
}
```

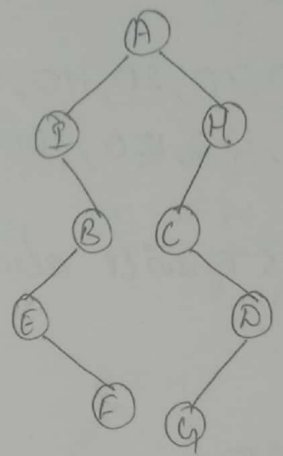
```
Inorder (root)
{   Inorder (root → left);
    pf (root → data);
    Inorder (root → right);
}
```

Note

Inorder, preorder and postorder on a binary tree takes $O(n)$ time always. (Best case, Worst case, Average case)

n nodes

Ques

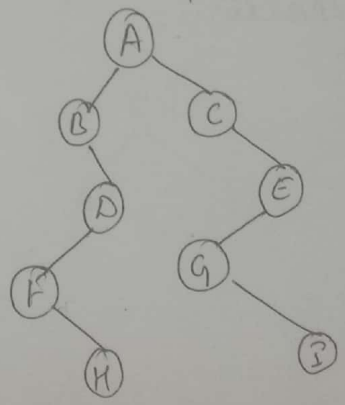


Inorder → I E F B A C G D H

preorder → A I B E F H C D G

postorder → F E B I G D C H A

Ques

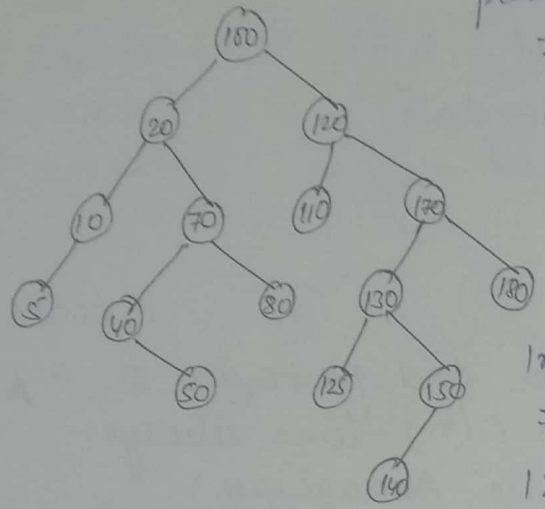


preorder = A B D F H C E G I

postorder = H F D B I G E C A

Inorder = B F H D A C G I E

(3)



preorder – 100, 20, 10, 5,
70, 40, 50, 80,
120, 110, 170, 130, 125,
150, 140, 180

Inorder = 5, 10, 20, 40, 50,
70, 80, 100, 110, 120,
125, 130, 140, 150, 170,
180

postorder = 5, 10, 50, 40, 80, 70, 20, 110, 125,
140, 150, 130, 180, 170, 120, 100

Note

* Inorder traversal of a BST will always produce ascending order.

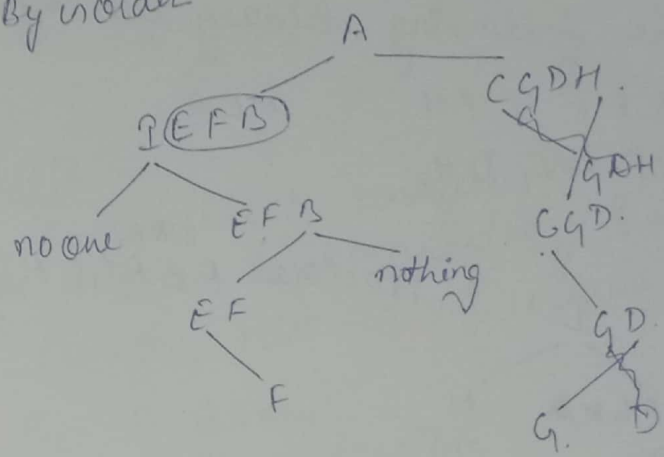
Left Root Right
small middle Greater

* BST with n nodes,

Q Consider the following Binary tree info?

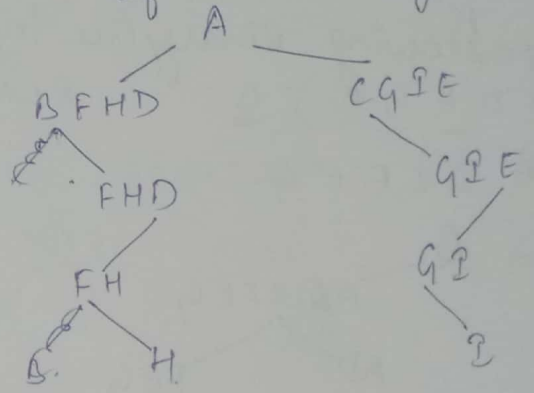
Preorder – (A) I B E F H C D G
Inorder – I E F B (A) C G D H.
Postorder ?

By inorder



Ans Post H F D B I G E C A — start from last

In B F H D A C G I E
left Right



Note

To construct binary tree for the given pre-order, post order and inorder: ~~Inord~~ $O(n^2)$ time is req. (n-times linear search for finding.

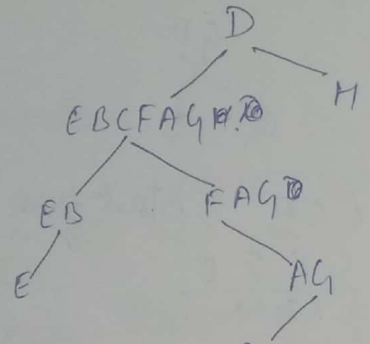
Q Consider the following binary tree info.

Pre: [D] C B E F G A H — NLR.

In: E B C F A G D H.

post order? — LRN

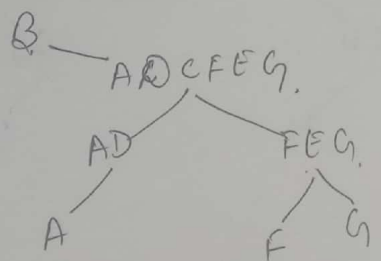
postordu E B A G C H D



Q Consider the following Binary tree inf. — RLN

Post: A D F G E C B

In: B A D C F E G.

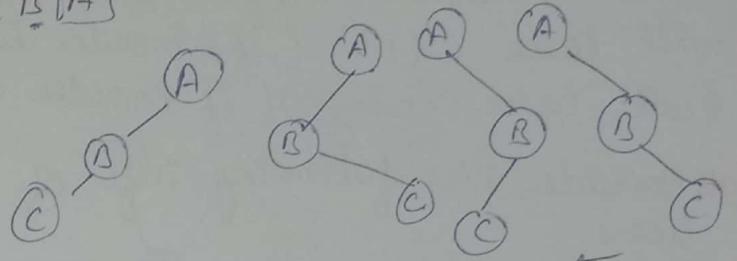


Ques Consider the following BT info.

Pre: [A] B C

post C B [A]

↳ many guess.



Note:

To construct unique binary tree.

~~Pre~~ In-order is compulsory.

Pre-in, Post-in } unique binary tree.

Pre, Post } Binary tree is possible but not unique

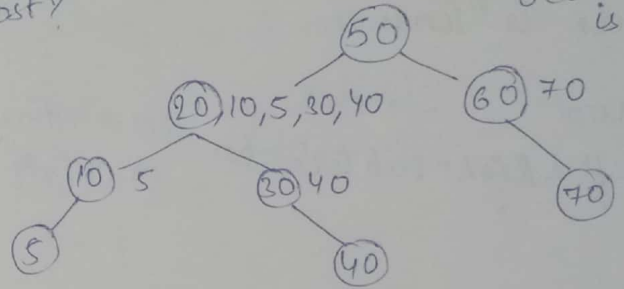
Ques Consider the following BST info.

Pre: 50 20 10 5 30 40 60 70

post?

because of BST inorder is ascending order

5 10 20 30 40 50 60 70



5, 10, 40, 30, 20, 70, 60, 50.

Note:

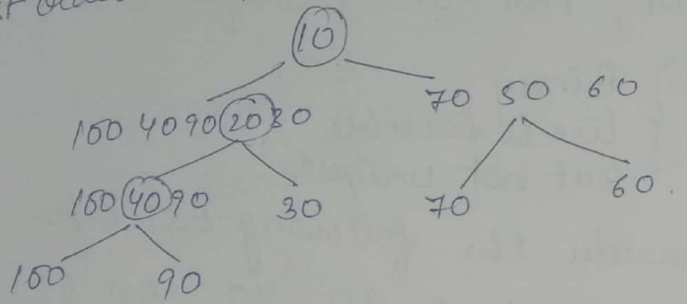
To construct binary tree for the given (preorder, Inorder) or (post-order, Inorder) will take $O(n^2)$ (if inorder is not sorted) & will take $O(n \log n)$ (if inorder is sorted)

Ques Consider the following info of min heap tree.

~~Preorder~~ – ~~10, 20, 40, 100, 90, 30, 50, 70, 60~~

Inorder – 100 40 90 20 30 10 70 50 60.

post order – 100, 90, 40, 30, 20, 70, 60, 50, 10



Note

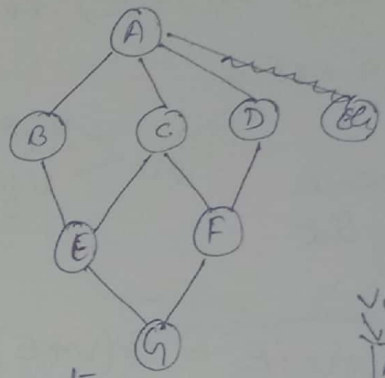
Inorder gives Left sub tree and Right sub tree if root node is known.

1) Finding root → n } $2n$ * ntimes.
2) Finding left & Right sub tree } n ⇒ $O(n^2)$.

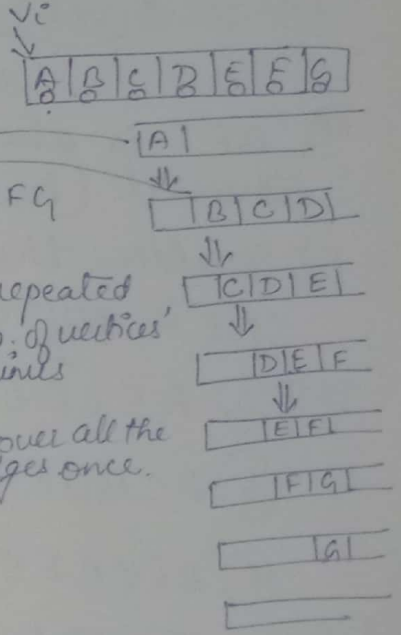
GRAPH TRAVERSAL

1) Breadth First Traversal & Depth First Traversal.

1) BFT




```
BFT(v)  → any vertex
{
  visited(v) = 1
  Add(v, Q)
  while (Q is not empty) }→ repeated 'no. of vertices' times
  {
    x = delete(Q)
    for all w adj to x. }→ cover all the edges once.
    {
      if (w is not visited)
      {
        visited(w) = 1
        add(w, Q);
      }
    }
  }
}
```



Note

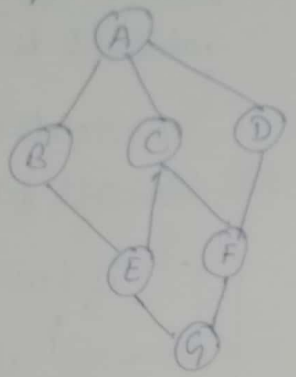
1. To implement BFT we are using 'Queue' Data structure
2. Time complexity = O(V+E) i.e. all cases [B C / W C / A C]
3. Space = i/p + extra
   - i/p: if graph is given as adjacency list
   - extra: 2V < V for array + V for queue

O(V+E)

3V+E = O(V+E)

4. BFT is also known as level-order traversal (level-by-level printing)
5. Consider the following graph

Give 4 diff" BFT's.



Sol^n (a) A B C D E F G

(b) A C B D E F G

(c) A D C B F E G

(d) A C B D F E G

(e) A C D B E F G

(f) C A E F B D G

(g) G E F B C D A

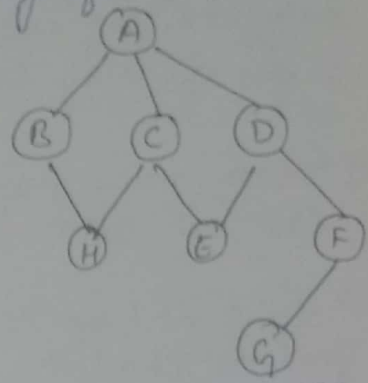
(h) E B G A F C D X

C

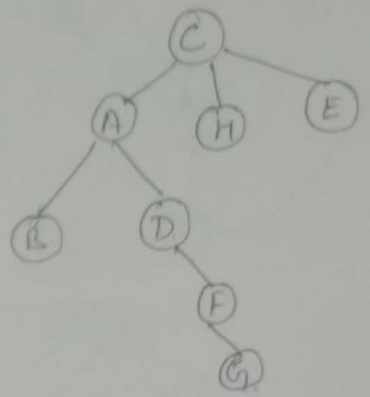
Ques Consider the following graph

Give 4 diff" BFT starting from G

1. G F D E A C B H
2. G F D A E B C H
3. G F D A E C B H
4. ~~G F D E A C H~~
5. ~~G F D A E A~~



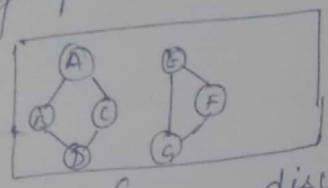
BFT (tree) is a spanning tree (covering all the vertices with (n-1) edges)



(BFT) of above graph

Applications of BFT

1. using BFT we can check a graph is connected or not
2. using BFT we can find out no. of connected components in the given graph.



G_1 disconnected

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

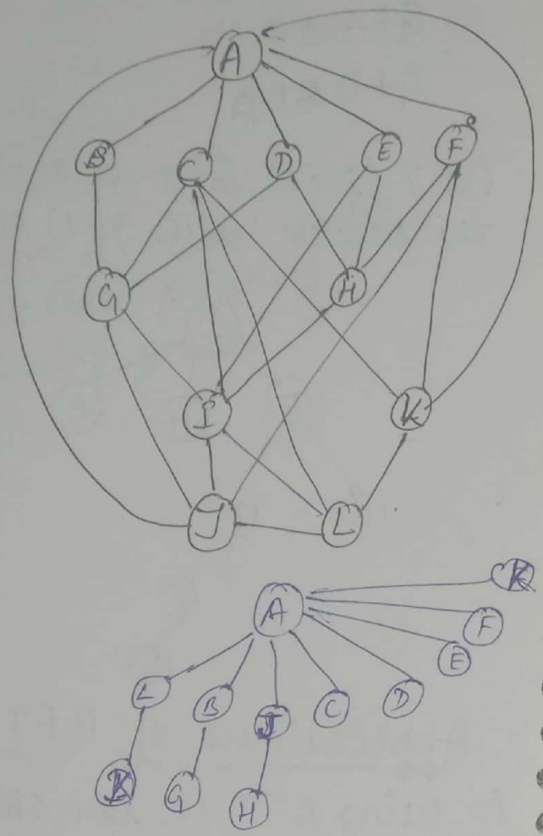
Applied BFT one time from

For the above example we applied BFT 4 times but the complexity will be (V+E) only coz only (V+E) are covered as a whole. and 1 can of away to judge the application of BFT.

3. For a given graph contain cycle or not we can use BFT. It is a P-class problem

4. Using BFT we can find out shortest path from the given source to every other vertex in the unweighted graph O(V+E)

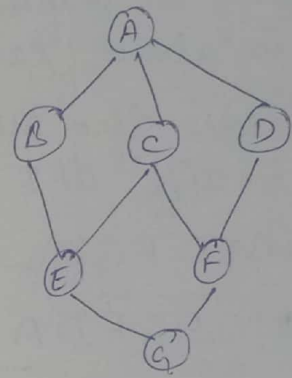
7. Using BFT we can verify a given graph is bi-partite graph or not



DEPTH FIRST TRAVERSAL

```
DFT (v)
{
1. visited (v) = 1
2. pf (v);
3. for all w adj to v
   {
     if (w is not visited)
     DFT (w);
   }
}.
```

each function call will occupy 1 stack place return of function call free the stack space



A

ø	ø	ø	0	ø	0	0
A	B	C	D	E	F	G

DFT(A)
1. ✓
2. A
3. w = B(1)
1. B

DFT(B)
1. ✓
3 w = AE
1. A ✓ E. DFT(E)

DFT(E)
1. ✓
3. w = BC G
1. B covered ✓
2. C DFT(C)

DFT(C)
1. ✓
3. w = AEF
1. A covered ✓
2. E covered ✓
3. F DFT(F)

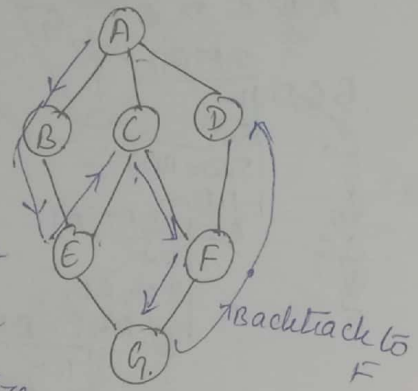
DFT(F)
1. ✓
3. w = C D G
1. C covered
2. D DFT(D)

DFT(D)
1. ✓
3. w = A, F
1. A covered
2. F covered

1% To implement DFT, we all using stack
2% For loop is covering all edge and recursion is covering all vertices so time complexity is $O(V+E)$ {all cases i.e. BC, WC, AC}
3% Space complexity = input + extra
input $(V+E)$
extra: stack size max 'V' ; Array of size V
$3V+E \equiv O(V+E)$

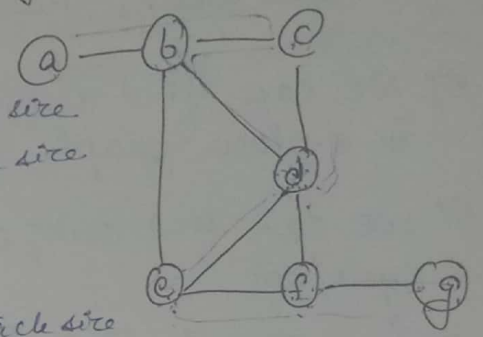
Max size of stack can be used = 'V' size
According to graph size of stack can vary.
Like is above case 'V-1' was the stack size

Ques Consider the graph.
4-diff^{n} DFT



1% A B E C F G D → 6 stack size.
2% ~~A B~~ C E G F D A B → 7 stack size
3% A D F G E C B. → 6 stack size
4% A B E G F D C → 6 stack size
backtrack
5% F C E G B A D → 6 stack size
backtrack
6% D F C A B E G

Ques Consider the following graph :-

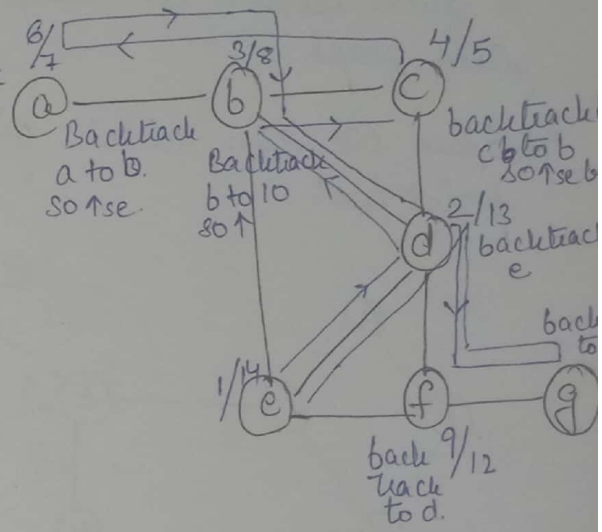


DFT (T/F)
B.T

1% e d b c a f g ✓ 4 stack size
2% g f e d b a c ✓ 6 stack size
3% b d c a f e g X
4% a b c d f g e ✓ 6 stack size
B.T.

whenever you backtrack ↑se a no.

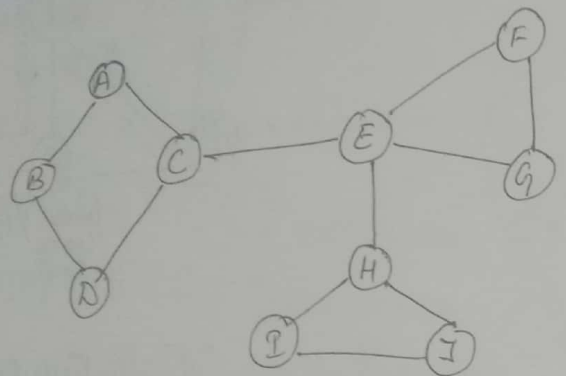
1% e d b c a f g (1/14, 10/11)



'1' starting time
14 finishing time

Application of DFT

1. we can verify given graph is connected/not
2. we can find out no. of connected components in a given graph
3. we can find out, a given graph contain a cycle/not
4. we cannot find out shortest path from a given vertex to every other vertex in a the given graph.
5. Given We can check given vertex is articulation point/not



articulation pt: C, E, H
deleting on which the graph becomes disconnected.

To check 'X' is articulation pt. or not
1. with 'X' on graph apply DFT & if all vertexs covered
2. then without 'X' apply DFT if all vertex are not covered (i.e. ~~their~~ respective values of some vertex remains zero then we came to know that that graph has becomed disconnected) and the 'X' was articulation pt.

6. we can check the given graph is strongly connected / not.

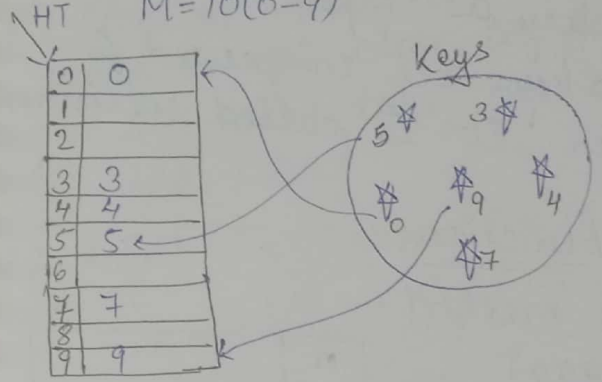
strongly connected — In Directed graph if there is path b/w every pair of vertices → strongly connected graph.

P, NP, NP

Hashing

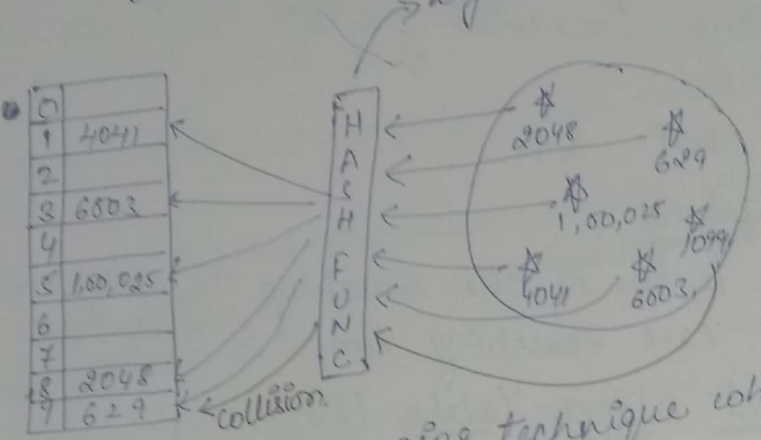
It is one of the searching technique where worst case searching time is O(1)

Direct Address Table (DAT)



(1) In direct adds table key itself is the address without any calculation
(2) worst case searching time is O(1)
(3) The drawback with the DAT is that even when though the no. of keys very less but one key may be very large (2^{1000}) then there

there is a need of Hash table approx. 2^{1000} i.e. $2^{1000}-1$. To eliminate this drawback we are going to the hash func.



Hash func is a mapping technique which takes very large key as input.

Collision – if two keys are compressed in same manner then it is called collision.

Types of Hash Functions

1. Divison modulo method –

M = 1000 (0-999)

Key = 13265487 9

$hf(k) = k \bmod m$

= 879

0	
1	
2	
⋮	
879	13265487 9
⋮	
999	

ex-2

$M = 8 (2^3)$

$2^3 \equiv 8$ → 0-7 remainder after dividing no. by 8 so to represent 0-7 → 3 bits are used to

$10101116000010101 \bmod 2^3 = 101$

ex-3

$m = 2^k$

$1150111160001010\underbrace{10010101}_{\text{suppose k bits}} \bmod 2^k = 10010101$

Note

1. Don't choose m values exactly powers of 2 because if $m = 2^k$ then hash func of keys = LSB 'k' bits always.
2. Pick 'm' value which is a prime and which is not close to powers of 2

511 → don't take coz it is close to $2^9 = 512$

1023 → don't " " " " " $2^{10} = 1024$

2. Digit Extraction Method (TRUNCATION METHOD)

M = 1000 (0-999)

K = 1 2 3 4 5 6 7 8 9
13 2 65 4 8 7 9

hf(k) = 249

0	
1	
2	
⋮	
249	13265487 9
⋮	
999	

3) MID-SQUARE METHOD

M = 1000 (0-999)

Key = 8492617

$(Key)^2 = (8492617)^2$

⇓

abcdefghijklm

↳ Take middle 3 digits

0	
1	
2	
3	
⋮	
ghi	8492617
⋮	
999	

4) Folding Method

(i) Fold ~~shifting~~ Boundary

M = 1000 (0-999)

key = 132654798

132
+ 798
930 index

0	
1	
2	
3	
⋮	
930	132654798
⋮	
999	

(ii) Fold shift Boundary

M = 1000 (0-999)

k = 132654798

132
654
+ 798
1584

158
+ 4
162

Either 158

0	
1	
2	
⋮	
162	132654798
⋮	
999	

COLLISION RESOLUTION TECHNIQUES

- chaining (outside)
- open addressing (inside)
  - linear probing
  - Quadratic probing
  - Double Hashing

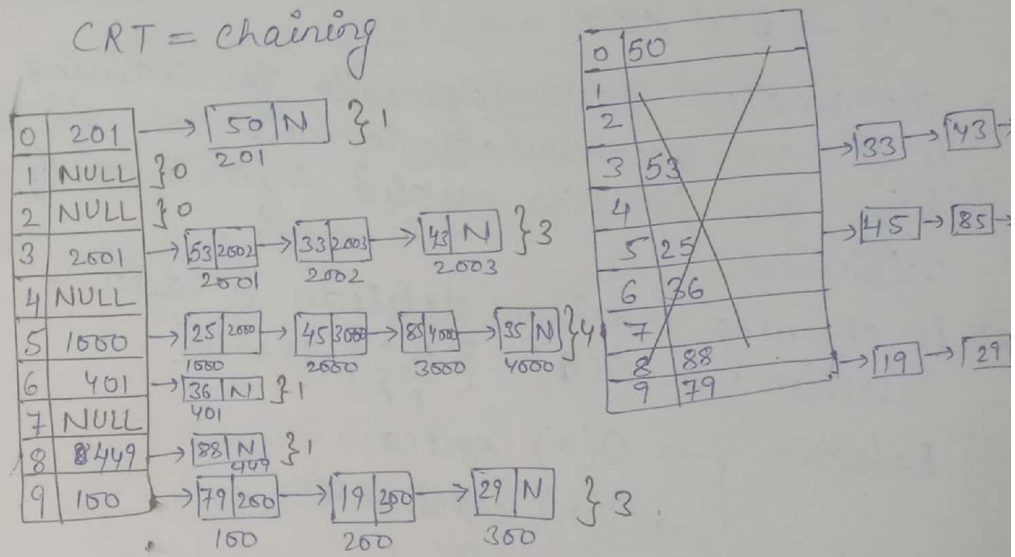
Chaining

ex: m = 10 (0-9)

hf(k) = k mod m

keys = 25, 79, 50, 45, 85, 53, 36, 19, 35, 29, 88, 33, 43

CRT = chaining



length of longest chain = 4

In the worst case, the length of longest chain of n elements = n.

Average chain length = $\frac{1+0+0+3+0+4+1+0+1+3}{10}$

$\frac{13}{10} = 1.3$.

Note

The lengths of the longest chain possible in worst case is 'n' because of this reason the worst case of searching is $O(n)$.

Drawback

The drawback with the chaining is we are wasting lot of space in the form of linked list even those place available inside the table.

$O(1) \rightarrow$ best case

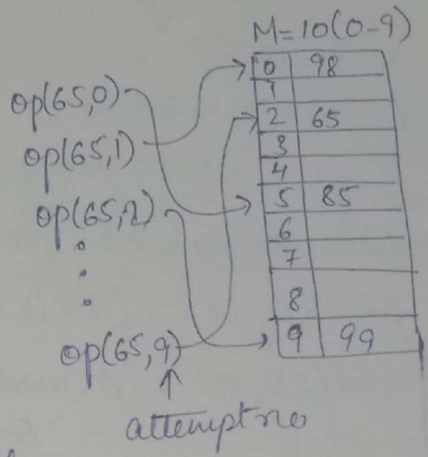
⇒ The greatest advantage with the chaining is it can accomodate infinite no. of keys. or it can resolve infinite no. of collisions.

⇒ In chaining insertion, deletion is easier.
Insertion is $O(1)$ { B C, W C, A C } all cases.

Deletion → $O(1)$ Best case
→ $O(n)$ worst case.

Open Addressing

1. If I hash to a particular slot & if it is already full then rehash again with diff^n hash func. (same as hash func^n with diff^n variable) until an empty slot is found



Continue this step max 'm' times.

2. we are not wasting space unnecessarily in the form of link-list.
3. Searching time is $O(n)$ max (worst case)
" " " $O(1)$ min (Best case)

m slots n-keys.

m-slots = n-keys.

1 slot = $\frac{n}{m}$ keys/slot

(load factor (α))

$0 \leq \alpha \leq 1$ [open addressing]

$0 \leq \alpha \leq \infty$ [chaining

Linear Probing

ex: M=10(0-9)

$hf(x) = x \bmod m$

keys = 25, 79, 43, 55, 67, 90, 143, 78, 95

Collision Resolution Technique = ~~LP~~ Linear Probing

$hf(key, i) = (hf(key) + i) \bmod m$

① $i=0$ $hf(25,0) = (25 \bmod 10 + 0) \bmod m$

$= 5+0 = 5^{th}$ ~~index~~

$= (5+0) \bmod 10$

$= 5^{th}$ index

$\forall i \in \{0, 1, 2, \ldots m-1\}$

soln LP(25,0)

(i) $(25 \bmod 10 + i) \bmod 10$

$(5+0) \bmod 10$

$= 5$

(ii) LP(79,0)

$(79 \bmod 10 + i) \bmod 10$

$(9+0) \bmod 10 = 9$

(iii) $(43 \bmod 10 + i) \bmod 10$

$(3+0) \bmod 10 = 3$

(iv) $(55 \bmod 10 + i) \bmod 10$

(a) LP(55,0) $5+0 \bmod 10 = 5^{th}$ is full

so (b) LP(55,1) $(5+1) \bmod 10$

$6 \bmod 10 = 6$

0	90
1	95
2	
3	43
4	143
5	25
6	55
7	67
8	78
9	79

(v) $(67 \bmod 10 + i) \bmod 10$

$= (7+0) \bmod 10$

$= 7$

$(143 \bmod 10 + i) \bmod 10$

$(3+0) \bmod 10$

3 is full

$3+1 \bmod 10 = 4 \bmod 10$

$= 4$

1) we will attempt (probe) one-by-one slot in the sequence.

2) Worst case searching time O(n)

best case " " O(1)

3) Deletion difficult - but we can manage with the special symbol '$'. If more '$'s then rehash the table again.

Quadratic Probing

ex:

$m = 10(0-9)$

$hf(x) = x \bmod m$

keys = 25, 79, 43, 55, 67, 90, 143, 78, 95

CRT = Quadratic Probing $c_1=1$ $c_2=1$

$QP(key, i) = (hf(key) + c_1 i + c_2 i^2) \bmod m$

$c + bx + ax^2$, $a=2$

$\forall i \in \{0, 1, 2, \ldots m-1\}$.

soln

$QP(25,0) = 5 + 1.0 + 1.0^2 = 5$

$QP(79,0) = 9 + 1.0 + 1.0^2 = 9$

$QP(43,0) = 3 + 1.0 + 1.0^2 = 3$

$QP(55,0) = 5 + 1.0 + 1.0^2 = 5$ full

$5 + 1.1 + 1.1^2 = 7$

0	90
1	95
2	
3	43
4	
5	25
6	
7	55
8	78
9	79

$QP(67,0) = 7 + 1.0 + 1.0^2 = 7$ full

$QP(67,1) = 7 + 1.1 + 1.1^2 = 9$ full

$QP(67,2) = 7 + 1.2 + 1.2^2 = 13 \bmod 10 = 3$ full

$QP(67,3) = 7 + 1.3 + 1.3^2 = 29 \bmod 10 = 9$ full

$QP(67,4) = 7 + 1.4 + 1.4^2 = 7 + 4 + 16 = 27 \bmod 10 = 7$ full

$QP(67,5) = 7 + 1.5 + 1.5^2 = 7 + 5 + 25 = 37 \bmod 10 = 7$ full

$QP(67,6) = 7 + 1.6 + 1.6^2 = 7 + 6 + 36 = 49 \bmod 10 = 9$ full

$QP(67,8) = 7 + 1.8 + 1.8^2 = 7 + 8 + 64 = 79 \bmod 10 = 9$ full

$QP(67,9) = 7 + 1.9 + 1.9^2 = 7 + 9 + 81 = 97 \bmod 10 = 7$ full

67 cannot be stored.

If we are probing in quadratic equation manner

2. worst case searching time $= O(m)$

best case " " $= O(1)$

3. Deletion is ~~deletion~~ difficult but we can manage with help of '$'. If more dollars rehash it.

Double Hashing

ex: $M = 10(0\text{-}9)$

$hf(x) = x \bmod m$

keys = 25, 79, 43, 55, 67, 90, 143, 78, 95

CRT = Double Hashing.

$$DH(key, i) = (\{hf_1(key) + i \cdot hf_2(key)\}) \bmod m$$

$\forall i \in \{0, 1, 2, \ldots m-1\}$.

$hf_2(key) = 1 + (key \bmod (m-2))$

Soln $DH(25,0) = 5 + 0 \cdot hf_2(25)$

$= 5$

$DH(79,0) = 9 + 0 \cdot hf_2(79)$

$= 9$

$DH(43,0) = 3 + 0 \cdot hf_2(43)$

$= 3.$

$DH(55,0) = h_1(55) + 0 \cdot h_2 55$

$= 5$ (full)

$DH(55,1) = 5 + 1.(55 \bmod 8 + 1)$

$= 5 + 1.(7+1) = (5+8) \bmod 10 = 3.$

$DH(55,2) = 5 + 2.8 = (5+16) \bmod 10 = 1$

$DH(67,0) = 7 + 0.hf_2(67)$

$= 7^{th}$ place

$DH(90,0) = 90 + 0.hf_2(90)$

$= 0$

0	90
1	55
2	
3	43
4	
5	25
6	
7	67
8	
9	79

$DH(143,0) = 3 + 0.hf_2 = 3$ full

$DH(143,1) = [3 + 1.(143 \bmod 8 + 1)] \bmod 10$

$= 3 + 1.1(7+1)$

$= 11 \bmod 10$

$= 1$

$DH(143,2) = [3 + 2.(143 \bmod 8 + 1)] \bmod 10$

$= 3 + 2.(8)$

$= 3 + 16 = 19 \bmod 10 = 9$

$DH(143,3) = 3 + 3.8$

$= 3 + 24 = 27 \bmod 10 = 7$

$DH(143,4) = 3 + 4.8$

$= 3 + 32 = 35 \bmod 10 = 5$

ex

$M = 10\ (0-9)$

$hf(k) = k \bmod m$.

keys = 54, 70, 93, 82, 71, 10, 20

CRT = L.P

$L.P = 54 \bmod 10 = 4$

$70 \bmod 10 = 0$

$L.P = 10 \bmod 10 = 0$

started from '0' $0+0=0$, $0+1=1$, $0+2=2$, $0+3=3$, $0+4=4$, $0+5=5$ done 6 attempts

$L.P = 20 \bmod 10 = 0$

started from '0' $0+0=0$, $0+1=1$, $0+2=2$, $0+3=3$, $0+4=4$, $0+5=5$, $0+6=6$ done 7 attempts

0	70
1	71
2	82
3	93
4	54
5	10
6	20
7	
8	
9	

Primary clustering

If two keys started from same Hash address. then those two key follow the same path unnecessarily to find an empty slot. in linear manner.

Because of this reason average searching time uses. This problem is known as. primary clustering

eg: In above example, '10' did 6 attempts and after '20' did '6 + 1 inertia' ie, 7 attempts

1. Linear probing is suffering with primary clustering

2. Average searching time of linear probing

$$= \frac{1+2+3+4+5+\ldots+m}{m} = \frac{\frac{m(m+1)}{2}}{m} = \frac{m+1}{2} = O(m)$$

Quadratic probing

ex: M = 10 (0-9)

h(k) = k mod m

keys 54, 70, 93, 82, 71, 10, 20

CRT = Quadratic probing

QP(10,0) = 0 + 1.0 + 1.0² = 0

QP(10,1) = 0 + 1.1 + 1.1² = 2

QP(10,2) = 0 + 1.2 + 1.2² = 6

0	70
1	71
2	82
3	93
4	54
5	
6	10
7	
8	
9	

QP(20,0) = 0 + 1.0 + 1.0² = 0

QP(20,1) 0 + 1.1 + 1.1² = 2

QP(20,2) 0 + 1.2 + 1.2² = 6

0 + 1.3 + 1.3² = 12 mod 10 = 2.

0 + 1.4 + 1.4² = (4 + 16) mod 10 = 20 = 0

0 + 1.5 + 1.5² = (5 + 25) mod 10 = 30 = 0

0 + 1.6 + 1.6² = (6 + 36) mod 10 = 42

⋮

⋮

Secondary clustering

1. If two keys are started from same hash address. They both follow the same path unnecessarily to find an empty slot in Quadratic manner.

Because of this address searching time is used. (less than the linear probing.

this avg searching time)

This problem is known as secondary clustering

2. Quadratic probing is suffering from secondary clustering.

3) Avg searching time = O(m)

[mathematically decreased but order of n]

Ques ex: M = 10 (0-9)

$h_2(k) = 1 + (k \bmod (m-2))$

$h_1(k) = k \bmod m$

keys = 54, 70, 93, 82, 71, 10, 20

0	70
1	71
2	82
3	93
4	54
5	20
6	10
7	
8	
9	

DH(10,0) = $h_1(10) + 0.h_2(10)$

(10,0) = 0 + 0.3 = 0

(10,1) = 0 + 1.3 = 3

(10,2) = 0 + 2.3 = 6.

DH(20,0) = $h_1(20) + 0.h_2(20)$ (20,1) = 0 + 1.5 = 5

(20,1) 10 + 0.5 = 0

$DH(30,0) = 0 + 0.7$

1. Avg case of double hashing. searching time $= O(1)$ because all the keys will follow diff^n path.

2. In double hashing 99% clustering is eliminated but 1% is still left out.

Note:

1. Using perfect hashing we will get worst case searching time $O(1)$

For every slot a new hash func^n is used. So for a table of m partitions (m+1) hash function (1 for a whole table and m for slots). So m hash tables are formed.

→ perfect Hashing.

Formulaes

① **A.P**

$\text{term}_2 = \text{term}_1 + d$ or $a, a+d, a+2d, a+3d, \ldots$

(a) $t_n = a+(n-1)d$

↓ first term ↳ common difference

(b) no of term $= \frac{l-a}{d} + 1$

$l \rightarrow$ last term

(c) Sum of first n terms

$S_n = \frac{n}{2}[2a+(n-1)d] = \frac{n}{2}[a+l]$

(d) If a, b, c are in AP

$2b = a+c$

To solve most of the problems related to A.P., the terms can be conveiently taken as.

3 terms : $(a-d), a, (a+d)$

4 terms : $(a-3d), (a-d), (a+d)(a+3d)$

5 terms : $(a-2d)(a-d), a, a+d\ (a+2d)$

(e) $T_n = S_n - S_{n-1}$

(f) If each term of an A.P is ↑sed ↓sed, ✕ or ÷ by same non-zero constant, the resulting sequence also will be in A.P

② Harmonic Progression (H.P)

(a) Non-zero nos are in H.P if $\frac{1}{a_1}, \frac{1}{a_2}, \frac{1}{a_3}, \ldots \frac{1}{a_n}$ are in A.P

↓

$a_1, a_2, a_3 \ldots a_n$

(b) if $\frac{1}{a}, \frac{1}{a+d}, \frac{1}{a+2d} \ldots$ are in H.P

n^{th} term of H.P $= \frac{1}{a+(n-1)d}$

(c) a, b, c are in H.P

b is the Harmonic mean b/w a & c

$b = \frac{2ac}{a+c}$

or

$\frac{2}{b} = \frac{1}{a} + \frac{1}{c}$

③ Geometric Prog

ratio of any term and its preceding term is constant

$a, ar, ar^2, ar^3 \ldots$

$a \rightarrow$ first term $r =$ common ratio

(a) n^{th} term $\Rightarrow t_n = ar^{n-1}$

(b) Sum of 1st n terms $= S_n$

$$S_n = \begin{cases} a(r^n-1)/r-1 & (\text{if } r>1) \\ \frac{a(1-r^n)}{1-r} & (\text{if } r<1) \end{cases}$$

(c) Sum of an infinite G.P

$S_\infty = \frac{a}{1-r}$ (if $0<r<1$)

Example: $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \infty$

(d) a, b, c are in G.P

b = Geometric mean (G.m) $= \sqrt{ac}$

Note: if a & b are of opposite sign, their G.M is not defined.

(e) or $b^2 = ac$

(f) or $\frac{a-b}{b-c} = \frac{a}{b}$

product of terms equidistant from beginning and end will be constant.

To solve G.P problems, terms can be taken as

(a) 3 terms: $\frac{a}{r}, a, ar$

(b) 5 terms $\frac{a}{r^2}, \frac{a}{r}, a, ar, ar^2$

(4) Relationship b/w A.P, G.P, & H.P.

$Gm^2 = Am \times HM$ of 2 positive no.

(i) a, b, c are in A.P if $b = \frac{a+c}{2}$

(ii) a, b, c are in H.P if $b = \frac{2ac}{a+c}$

(iii) a, b, c are in H.P if $\frac{a-b}{b-c} = \frac{a}{c}$

(iv) $A > G > H$ of 2 positive no. (distinct)

A, G, H are in G.P.

if a series is both an A.P & G.P $\Rightarrow$ all the terms of the series will be equal.

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

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

$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4} = \left[\frac{n(n+1)}{2}\right]^2$

Formulas of Log

(1) $10^3 = 1000$ $\log_{10} 10^3 = 3 \Rightarrow \log_x x^n = n$

a = positive real no. other than 1

$a^m = x$

$\log_a a^m = \log_a x$

$m = \log_a x$

(2) $\log_x x = 1$

(3) $\log_a 1 = 0$

$\log_a x^n = n \log_a x$

$$\log_a x = \frac{1}{\log_x a}$$

$$\log_a x = \frac{\log_b x}{\log_b a} = \frac{\log x}{\log a}$$

Q1

$$T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2)+n^2 & \text{if } n>0 \end{cases}$$

$$T(18) = T(8) + 10^2$$
$$= T(6) + 8^2 + 10^2$$
$$T(4) + 6^2 + 8^2 + 10^2$$
$$T(2) + 4^2 + 6^2 + 8^2 + 10^2$$
$$T(0) + 4^2 + 6^2 + 8^2 + 10^2$$
$$1 + 4^2 + 6^2 + 8^2 + 10^2$$

$$T(n) = n^2 + (n-2)^2 + (n-4)^2 + (n-6)^2 + \ldots + 1$$

~~$n^2 + n^2 + 1$~~

$$\Rightarrow 1 + 2^2 + 4^2 + 6^2 + 8^2 + \ldots (n-2)^2 + n^2$$

Q2

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

$$T(10) = T(8) + 10$$
$$= T(4) + 8 + 10$$
$$= T(2) + 4 + 8 + 10$$
$$= T(1) + 2 + 4 + 8 + 10$$

$$T(12) = T(6) + 12$$
$$= T(3) + 6 + 12$$
$$= T$$

$$T(n) = T(n/2) + n$$
$$= T(n/4) + n/2 + n$$
$$= T(n/8) + n/4 + n/2 + n$$
$$= n + n/2 + n/4 + n/8 + \ldots + 1$$

Q3 $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 8T(n/2) + n^2 & \text{if } n>1 \end{cases}$

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

$= 8\left[T(n/4) + (n/2)^2\right] + n^2$

Formulaes

1) $1 + 2 + 3 + 4 + \ldots + n = \frac{n(2n+1)}{2}$

2) $1^2 + 2^2 + 3^2 + 4^2 + \ldots + n^2 = \frac{n^3(n^3+1)(2n^3+1)}{6}$

3) $\log_a b = \frac{\log_2 b}{\log_2 a}$

4) Sum of terms in g.P $= \frac{a(1-r^n)}{1-r}$

5) $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots + \frac{1}{n} = O(\log n)$

Priority	Specialization
1	CS & Engg
2.	VLSI
3.	Integrated Elec & Circuit
4	CoA
5	OS
6	CS & A
7	Info sys
8	N/W & Mob Comm
9.	S/W Tech
10	I.T
11	Comp. Appn

www.facebook.com / made.easy12

www.madeeasy.in ⟹ friday / 2:00p.m / weekend schedule
(What's New / Notification)

ALGORITHMS → 10Ques (Min)

References
1. Introduction to algo by coreman

Syllabus
1. Analysis
2. Divide & Conquer
3. Greedy Technique
4. ~~Dooing~~ Dynamic Prog.
5. Hashing, Graph, Tree, P, np, npc & nph

Definition:
It is a combination of sequence of finite steps to solve a particular problem.

ex: To add 2 no.

```
ATN()
{
1. Take 2 no.s (a,b)
2. C = add(a,b)
3. Return (c);
}
```

Properties of Algo
1. It should terminate after finite time
   finite steps doesn't mean finite time.
2. Minimum One output.

3) It should take 0 or more (finite) i/p.
4). It should be Deterministic / Non ambiguous.
5). It is prog. lang independent.

Steps Required to construct Algorithm

1. Problems definition
2. Design Algo. → Divide & conq, Dynamic proble, Back Tracking, Greedy Tech. } choosing any one is Algo design
3. Draw flowchart.
4. Testing.
5. Coding
6). Analysis → (finite Time & Space Complexity)

Chapter - 1 (Analysis)

Time & Space complexity

Time have higher priority (less CPU time is better)

CPU time costs more than memory space.

If any problem is having more than one solution then best one will be decided by analysis based on 2 - factors.

1. Time (CPU-time)
2. Space (main-memory)

Time Complexity

Total Time of Prog (P) = Compile Time (P) + Run Time (P)

= depends on compiler → Software → Prog lang of compiler

depends on CPU → hardware → Type of h/w processor

Analysis

Apostiary Analysis
It is dependent on prog lang of compiler and type of processor.

Apriori Analysis
It is independent of prog. lang of compiler & type of processor

Apostiary Analysis	Apriori Analysis
2. Exact Answer	2. Approx. Answer.
3. Answer changing system to system (cor. of diff. config of compiler lang & CPU)	3. Same Answer everytime
4. It is relative analysis cor. it keeps on changing with environment	4. It is absolute anal.

Apriori Analysis

It is a determination of order of magnitude of a statement.

ex:

```
main()
{
    x = y + z;        => 1 (no. of times it executed)
}
```

Time complexity = $O(1)$

```
main()                          =>  1
{
    x = y + z                   =>  1
    for(i=1; i<=n; i++)         =>  n+1
    {
        x = y + z;              =>  n.
    }
}
```

$n+1 = O(n)$

③ ex:

```
main()
{                                          -> (1)
    x = y + z;
    for(i=1; i<=n; i++)
    {                                      -> n
        x = y + z;
    }
    for(i=1; i<=n; i++)                    -> if here n/2 then 1+n+n^2/2
    {                                          => O(n^2/2)
        for(j=1; j<=n; j++)                    => O(n^2)
        {
            x = y + z;                     -> n*n.
        }
    }
}
```

$1 + n + n^2 \Rightarrow O(n^2)$

(13) Finding Time complexity means the place where CPU is spending largest time (largest loop)

(14)

```
main()
{
    i = 0;                                 -> (1)
    while(i<=n)                                1 + n/2 = O(n/2)
    {                                              = O(n)
        i = i + 2;                         -> n/2
    }
}
```

(5)

```
main()
{ while(n>=1)
  {
    n=n-1;
  }
}
```

→ (n) ⇒ O(n)

↳ 10 ⇒ n/10 ⇒ O(n)

(6)

```
main()
{ while(n>=1)
  {
    n=n/2;
  }
}
```

O(log₂n)

↳ 3 ⇒ O(log₃n)

Coz of division → log. came

① n/2 ③ n/2³ ⑤ n/2⁵

② n/2² ④ n/2⁴ ⑥ n/2⁶ .... n/2^k = 1

n/2^k = 1

⇒ n = 2^k

log₂n = log₂2^k

k = log₂n

(9)

```
main()
{ i=1
  while(i<n)
  {
    i=2*i;
  }
}
```

↓

5 ⇒ O(log₅n)

(10)

```
main()
{ i=1
  while(i<n)
  {
    i=5*i;
    i=i/10;
  }
}
```

(11)

```
main()
{ i=2
  while(i<n)
  {
    i=i^2;
  }
}
```

↳ i^3

⇒ (2)³ = 2³¹

(2³)³ = 2⁹ = 2³²

(2⁹)³ = 2²⁷ = 2³³

⋮

2^3k

i=1

2*1

2²

2³

2⁴

⋮

2^k = n

log₂2^k = log₂n

k = log₂n

i = 1 × i/10 × 2 ⇒ log₂n

suppose n=1000

2 < 1000

4 < 1000

16 < 1000

256 < 1000

256² > 1000

log₂(log₂n)

$2^{3^k} = n$

$\log_2 2^{3^k} = \log_2 n$

$3^k \log_2 2 = \log_2 n$

$3^k = \log_2 n$

$k \log_3 3 = \log_3 \log_2 n$

$k = \log_3 \log_2 n$

```
main()
{  i=15
   while (i<n)
   {
      i = i^7
   }
}
```

$15^{7^k} = n$

$7^k \log_{15} 15 = \log_{15} n$

$7^k = \log_{15} n$

$\log_7 7^k = \log_7 \log_{15} n$

$k = \log_7 \log_{15} n$

$n \Rightarrow n^{1/2} \Rightarrow (n^{1/2})^{1/2} = n^{1/4}$

$(n^{1/4})^{1/2} = n^{1/8}$

```
main()
{  while (n>2)
   {
      n = n^1/2 => sqrt(n)
   }
}
```

$n \Rightarrow n^{1/2} \Rightarrow n^{1/2^2}$

$= n^{1/2^3} = n^{1/2^4} \ldots n^{1/2^k}$

$n^{1/2^k} = 2 \Rightarrow$

$1/2^k \log_2 n = \log_2 2$

$1/2^k \log_2 n = 1$

$1/2 \log_2 n = 2^k$

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

$\log_2 \log_2 n = k \log_2 2$

```
main()
{  while (n>2)
   {
      n = n^1/5
   }
}
```

$O(\log_5 \log_2 n)$

```
main()
{  i=2
   while (i<n)
   {
      i = i^5
      i = i^1/10
      i = i^4
   }
}
```

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

$2^{3^k} = n$

$\log_2 2^{3^k} = \log_2 n$

$3^k \log_2 2 = \log_2 n$

$3^k = \log_2 n$

$k \log_3 3 = \log_3 \log_2 n$

$k = \log_3 \log_2 n$

```
main()
{ i=15
  while (i<n)
  {
    i = i^7
  }
}
```

$15^{7^k} = n$

$7^k \log_{15} 15 = \log_{15} n$

$7^k = \log_{15} n$

$\log_7 7^k = \log_7 \log_{15} n$

$k = \log_7 \log_{15} n$

$n \Rightarrow n^{1/2} \Rightarrow (n^{1/2})^{1/2} = n^{1/4}$

$(n^{1/4})^{1/2} = n^{1/8}$

```
main()
{ while (n>2)
  {
    n = n^1/2 ⇒ √n
  }
}
```

$n \Rightarrow n^{1/2} \Rightarrow n^{1/2^2}$

$= n^{1/2^3} = n^{1/2^4} \ldots n^{1/2^k}$

$n^{1/2^k} = 2 \Rightarrow$

$1/2^k \log_2 n = \log_2 2$

$1/2^k \log_2 n = 1$

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

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

$\log_2 \log_2 n = k \log_2 2$

```
main()
{ while (n>2)                    O(log_5 log_2 n)
  {
    n = n^1/5
  }
}
```

```
main()
{ i=2
  while (i<n)
  {
    i = i^5
    i = i^1/10
    i = i^4
  }
}
```

$\left.\begin{array}{l} i = i^5 \\ i = i^{1/10} \\ i = i^4 \end{array}\right\} i = i^2 = \log_2 \log_2 n$

Q1 Consider the following C prog.

```
main()
{
  for(i=1; i<=n; i=2*i)
  {
    for(j=n; j>5; j=j^1/10)
    {
      a=b+c;
    }
  }
}
```

$\log_{10}\log$
$n \Rightarrow n^{1/10} \rightarrow n^{1/10^2} \rightarrow n^{1/10^3} \cdots$

inner loop
$n^{1/10^k} = 5$
$(\log_{10}\log_5 n)$
outer $\rightarrow \log_2 n$
simultaneous $\Rightarrow O(\log_2 n \times \log_{10}\log_5 n)$

Q2 Consider the following C prog.

```
main()
{
  for(i=n; i>=10; i=√i)  → O(log2 log10 n)
  {
    for(j=5; j<=n; j=7*j)  → O(log7 n)
    {
      for(k=1; k<=n; k=k+100)
      {
        a=b+c;   → n/100 → O(n)
      }
    }
  }
```

$O(n * \log_7 n * \log_2 \log_{10} n)$

Q3

```
i=5
while(i<n)
{
  i=i+10;
  i=i/10
  i=i^10;
}
```

log n →
$\log_{10} n$
$\log_{10}\log_{5} n$ → this would effect more so, final answer would be this only.

Q Consider the following C prog.

```
Main()
{
  for (i=1; i<=n^5; i= 10*i)
  {
     j=15
     while (j< n^3)
     {
        j= j^8
     }
  }
}
```

$\Rightarrow \log_{10} n^5$

$\Rightarrow 5 \log_{10} n$

$15^8 = n^3$

$(15^8)^8 = n^3$

$(15^8)^k = n^3$

$\log_n 15^{8k} = \log_n n^3$

$8k \log_n 15 = 3$

$8k \log_{15} 15 = \log_{15} n^3$

$8k = \log_{15} n^3$

$\log_8 8k = \log_8 \log_{15} n^3$

$k = \log_8 \log_{15} n^3$

$\Rightarrow k = \log_8 3 \log_{15} n$

$5 \log_{10} n \log_8 3 \log_{15} n$

$\Rightarrow O(\log_{10} n \log_8 \log_{15} n)$

(remove constants)

Q Consider the following C prog.

```
main()
{
  for(i=1; i<=n; i++)
  {
    for(j=1; j<=i; j++)
    {
      for(k=1; k<=500; k++)
      {
        a=b+c;
      }
    }
  }
```

1 1 n
2 1,2 n+n
3 1,2,3

i=n
j= 1 2 3 n
k = 500

$n \times n^n \times 500^n$

$T = 1*500 + 2*500 + 3*500 + \ldots n*500$

$500[1+2+3+\ldots+n)$

$O\left(\frac{500 \times n(n+1)}{2}\right) = O(n^2)$

Q Consider the following C prog.

```
main()
{
  for(i=1; i<=n^2; i++)
  {
    for(j=1; j<=i^2; j++)
    {
      for(k=n; k>1; k=k/10)
      {
        a=b+c
        c=d+e
      }
    }
  }
```

i j k
1 1 n/10
2 1,2,3,4 (n/10)/10

$\log_{10} n$

$\log_2 n^2 \log\log n^2 \log_{10} n$

$i=1$	$i=2$
$j=1$	$j=1,2,3,4$
$k=\log_{10} n$	$k=\log_{10} n \quad \log_{10} n \quad \log_{10} n$

$i = n^2$

$j = 1 \ldots (n)^{n^2}$

$1^2 \log_{10} n + 2^2 * \log_{10} n + 3^2 \log_{10} n + \ldots n^2 \times \log_{10} n + (n^2)^2 \log_{10} n$

$\Rightarrow \log_{10} n [1^2 + 2^2 + 3^2 + \ldots + n^2 + (n^2)^2]$

$\log_{10} n \; \frac{n^2(n^2+1)(2n^2+1)}{6}$

Q

```
main()
{
for (i=1; i<=n^2; i++)
  for (i=1; i<=n^3; i++)
    for (i=1; i<=n^4; i++)
    {
      a = b+c;
    }
}
```

$O(n^4)$

$i \quad i \quad i$

$n^4 \quad n^3$

i

$i \quad 1$

$i \quad 1 \; 2 \; 3^4$

$i \; n^4 \; n^4 \; n^4$

$\log n^4$

n^9

$n^4 + n^4 + n^4$ (5 terms) $= n \times n^4$

$n^4 * n^5 \; n^9$

$n^4 + n^4 + n^4 \ldots$

```
main()
{
  for (i=1; i<=n^2; i++)
  {
    for (i=1; i<=n^5; i++)
    {
      for (i=1; i<=n^4; i++)
      {
      }
    }
  }
}
```

infinite loop

Q consider the following C prog.

```
A(n)
{
  if (n <= 2) return;
  else
    return (A sqrt(n))
}
```

while ($n \leq 2$)
$n = \sqrt{n}$

$O(\log_2 \log_2 n)$

Asymptotic Notations

c → is a const that differs with processor speed
Same algo run on diff. system have diff constant value c

1). Big-OH
2) Omega Notation
3). Theta Notation

Let $f(n)$ & $g(n)$ be 2 +ve functions (for from 'n' onwards that gives always +ve value).

Big-OH Notation

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

iff $f(n) \leq c \cdot g(n)$, $\forall n$, $n \geq n_0$

such that there exists 2 +ve const. $c > 0$ && $n_0 \geq 0$

@ex1 $f(n) = n + 10$

$g(n) = n$

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

$n + 10 \leq C\, g(n) \Rightarrow n + 10 = c \cdot n$

where $c \geq 0$ $n \geq 10$

@ex2 $f(n) = 3n^2 + n + 10$ $g(n) := n^2$

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

$3n^2 + n + 10 \leq c \cdot n^2$

Assume (n^2)

Then cn^2 should be atleast $5n^2$

so $c \geq 5$ $n \geq 3$

$\therefore 3n^2 + n + 10 \leq 5 \cdot n^2$, $\forall n$, $n \geq 3$

$3n^2+n+10 = O(n^2)$

ex 3 $f(n) = n^2$ $g(n) = n^2+10$

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

$n^2 = c.(n^2+10)$

$\Downarrow$ $\Downarrow$

1 0

$n^2 = O(n^2+10)$, $\forall\ n,\ n \geq 0$

ex 4

$f(n) = n+10$

$g(n) = n-10$

$f(n) = O(g(n)) \Rightarrow (n+10) = c(n-10)$

$n+10 = 2(n-10)$, $n \geq$ 30

ex 5

$f(n) = n^2$ $f(n) = O\,g(n)$

$g(n) = n$ $n^2 = c.n$ where $c \geq n$

$n \geq 1$

Big-Oh is not possible because the value of C equivalent to n and n is func..

$n^2 \neq O(n)$

Omega Notation

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

iff $f(n) \geq c\,g(n)$ $\forall\ n,\ n \geq n_0$ such that there exists 2 +ve constants $c > 0$ && $n_0 \geq 0$

ex 1

$f(n) = n^2$, $g(n) = n$

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

$n^2 \geq c.n$, $c = 1$, $\forall\ n,\ n \geq 0$

ex 2

$f(n) = n^2$

$g(n) = n^2+n+10$

$n^2 \geq c(n^2+n+10)$

$c = 1/3$ $n \geq 3$

ex 3

$f(n) = n-10$

$g(n) = n+10$

$n-10 \geq c(n+10)$

$n-10 \geq \frac{1}{10}(n+10)$ $\forall\ n,\ n \geq$ 13

ex 4

$f(n) = n$ $g(n) = n^2$

$n \geq c.n^2$

$\downarrow$ $c = 1/2$

$\therefore$ $1/n \rightarrow$ function

so $n \neq \Omega\, g(n^2)$

Theta Notation

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

iff

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

$\forall n,\ n \geq n_0$

ex1 $f(n) = n+10$

$g(n) = n+20$

(a) $f(n) \leq c\, g(n) \Rightarrow n+10 \leq c(n+20)$ $\quad c=1$

(b) $n+10 \geq c(n+20)$ $\quad n \geq 0$

$\frac{1}{2}$

ex2 $f(n) = n^2$

$g(n) = n^2+n+10$

~~$f(n)$~~ $n^2 \leq c(n^2+n+10)$

$\downarrow$ $\quad n_0 = 0$

1

$n^2 \geq c_2(n^2+n+10)$

$\downarrow$ $\quad n_0 = 3$

$\frac{1}{3}$

$n^2 = \Theta(n^2+n+10)$

$\forall n,\ n_0 \geq 3$

$c_1 = 1$

$c_2 = \frac{1}{3}$

ex $f(n) = n^2$

$g(n) = n^2$

$n^2 \leq c\, g(n^2)$

$\downarrow$ $\quad n \geq 0$

1

$n^2 \geq c_2(n^2)$

$\downarrow$ $\quad n \geq 0$

1

imp **Note** (11)

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

and also

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

this implies

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

and also vice versa

ex $f(n) = n^2$ $\quad g(n) = n$

$n^2 \leq c\, n$

$\downarrow$

function $= n$ so $O(g(n))$ is not possible

so $f(n) \neq \Theta g(n)$

ex $f(n) = n$ $\quad g(n) = n^2$

* $n \leq c_1 n^2$ $\quad n = O(n^2)$

$\downarrow$

1

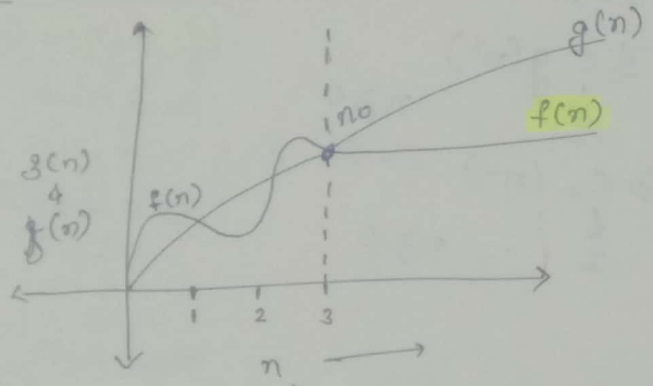
* $n \geq c_2(n^2)$

$\downarrow$

$\frac{1}{n}$ → function $\quad n \neq \Omega(n^2)$

so $n = \Theta(n^2)$ not possible

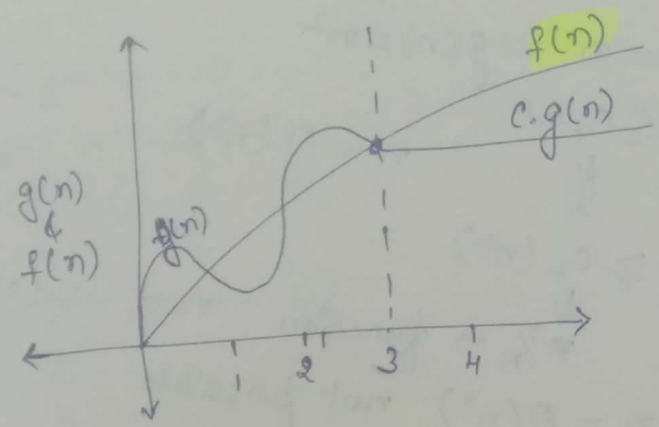
Big-OH Notation



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

$f(n) \leq c\,g(n), \quad \forall n, \; n \geq n_0$

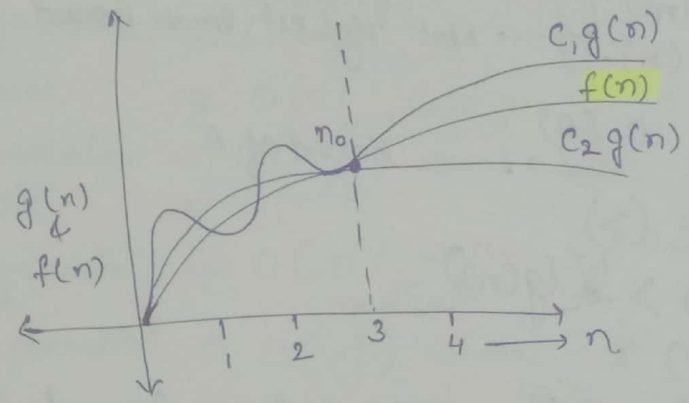
Omega



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

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

Theta Notation



Big-oh

It gives upper bounds.

$n^2 = O(n^2)$ ↳ tightest upb

In all the upper bounds ~~equo~~, the exactly equal upper bound is tightest upper bound

$n^2 = O(n^3)$, $n^2 = O(n^{10})$ → not tightest upper bound.

$A = O(B)$ maybe

Not Tight upper bound

tight upper bound.

Small-OH Notation (<)

$n^2 = o(n^2)$ ✗

$n^2 = o(n^3)$ ✓

$n^2 = o(n^{10})$ ✓

$A = o(B)$

B is upper bound of A but not tightest

Omega Notation

$n^3 = \Omega(n^3)$ → Tighest lower bound
$n^2 = \Omega(n^2)$
$= \Omega(n)$
lower bounds → Not tighest lower bound

$A = \Omega(B)$ → lower bound of A

Small Omega ($>$)

$f(n) > \omega(g(n))$

$n^3 \neq \omega(n^3)$ ✗
$= \omega(n^2)$ ✓
$= \omega(n)$ ✓
] Not tighest lower bound

Theta Notation

$n^3 = O(n^3)$ && $n^3 = \Omega(n^3) \Rightarrow n^3 = \Theta(n^3)$

$O(n^3)$: tighest upper bound
$\Omega(n^3)$: tighest lower bound
$\Theta(n^3)$: tighest upper & lower bound

* $T(A) = O(n^3)$
Algo-A worst case is n^3
* $T(A) = \Omega(n^3)$
Algo-A best case is n^3
* $T(A) = O(n^3)$ && $T(A) = \Omega(n^3)$
$\Rightarrow T(A) = \Theta(n^3)$

Algo-A best case & worst case both are same.

COMPLEXITY CLASSES

(13)

1) Constant $\Rightarrow O(1)$ (smallest time complexity)
2) Logarithmic $\Rightarrow O(\log n)$ → $\log\log n$ → $\sqrt{n}$
3) Linear $\Rightarrow O(n)$ → $n\log n$ → $n^{3/2}$
4) Quadratic $\Rightarrow O(n^2)$
5) Cubic $\Rightarrow O(n^3)$
6) Polynomial $\Rightarrow O(n^c)$ where $c > 0$ && c is a constant
7) Exponential $\Rightarrow O(c^n)$ where c is a constant && $c > 1$

Note:
$\log n < \sqrt{n}$

$\log 1000 = 10$
$\sqrt{1000} = \text{app. } 33$

$n^{3/2} = n^{1.5} = n^1 * n^{0.5} = n * \sqrt{n}$

8) $2^n < 3^n < 4^n \ldots$

$2^n \cdot 1 \;|\; 3^n$
$2^n \cdot 1 \;|\; (2 \times 1.5)^n$
$2^n \cdot 1 < 2^n \cdot (1.5)^n$ coz $1 < 1.5$

* $3^n \cdot 1 \;|\; 5^n$
$3^n \cdot 1 \;|\; \left(3 \times \frac{2.5}{3}\right)^n = 3^n \times \left(\frac{2.5}{3}\right)^n$
$1 < \frac{2.5}{3}$

9) $2^n = O(n^n) \Rightarrow 2^n = O(n^4)$ ✓

10) $n! = \Omega(2^n)$

11) $(\log n)^2 < n$

12) $(\log n)^4 < n$.

13) $(\log n)^{10000} < n$

14) $\log n$ always log will be smaller one than $\log n$.

15) $\log(\log n) < n$

16) $\log n < n$.

17) $(\log n)^2 \not< n$

18) $(\log n)^3 < n$

19) $(\log n)^{1000} < n$

20) $(\log n)^{\log n} > n$. $\log\log n \log n > \log n$

21) $2^n = \Omega 2^{n \log n}$

22) $\log_2 n > \log_{10} n$.

23) $\log_2 n = \Omega(\log_3 n)$

$\log_2 n$ | $\log_3 n$

$\Downarrow$

$\frac{\log_2 n}{\log_2 3} = \frac{1}{\log_2 3} \log_2 n = \frac{1}{1.5} \log_2 n$

$\log_2 n > \frac{1}{1.5} \log_2 n$

24. $\log_2 n = \Omega \log_3 n$

$\neq \omega(\log_3 \frac{n}{3})$

12. (i) $n = O(5n)$

$n \neq O(5n)$, $c = 1/5$

(ii) $10n = \Omega(n)$

$10n \neq \omega(n)$

$c = 10$

(iii) $n = O(n^2)$

$n = o(n^2)$

(iv) $n^2 = \Omega(n)$ ✓

$n^2 = \omega(n)$ ✓

Note

Small O satisfied then Big Scale is not necessary - that Big-Oh is to satisfied.

Sol (b) $\sqrt{\log n} = (\log n)^{1/2}$

$(\log n)^{1/2} = \Theta(\log \log n)$ true

$\frac{1}{2}\log \log n = \Theta(\log \log n)$ true

(c) $2^x \leq 2^y$

$x \leq y \Rightarrow$

$(\log n)^{1/2} \leq \Theta \log \log n$

$\frac{\log \log n}{2} \geq \log \log \log n$

So False.

(c) $2^x \leq 2^y$ true

$x < y$

(b) $\log(n+k)$

So. $(n+k)^n$

$2^{n+1} = O$

$2^n \cdot 2 = C \cdot 2^n$

true

(c) $2^{2n} = O($

2^{2^n} ↓ exponential

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

Note

* If Θ is possib

(e) $2^{2n} = O(2$

2^{n+n} ∴ 2

$2^n \cdot 2^n \neq 2$

↳

Note

* Don't apply log directly
* If you want to apply log first simplify and afterwards apply log.

Ques check the following statements are true or false

(a) $n^2 \cdot 16^{\log_4 n} = O(n^8)$

$n^x \cdot 16^{\log_4 n}$ $\quad n^{x+6}$

$16^{\log_4 n} = n^6$

$\log_4 n \log 16 \leq 6 \log n$ (true

(b) $\frac{4^n}{2^n} = O(2^n)$

$\frac{4^n}{2^n} = O(2^n)$

$2^n = O(2^n)$ true

$c = 1$

(c) $64^{\log_2 n} = O(n^5)$

$n^{\log_2 64} = n^5$

$n^6 \nleq c \cdot n^5$

↳ $c = n \Rightarrow n^6 = n^6$

false but c cannot be function so this is false

d) $\frac{1}{n} = O(1)$

true

$1/n \leq 1$

Note

1) $n^{\log_b a} = a^{\log_b n}$

2) $n^{\log_4 16} = n^2$ coz $4^2 = 16$

3) $\log_a b \Rightarrow$ if $a^x = b$ then we can write $\log_a b = x$

Anywhere log is given as power use above formulas

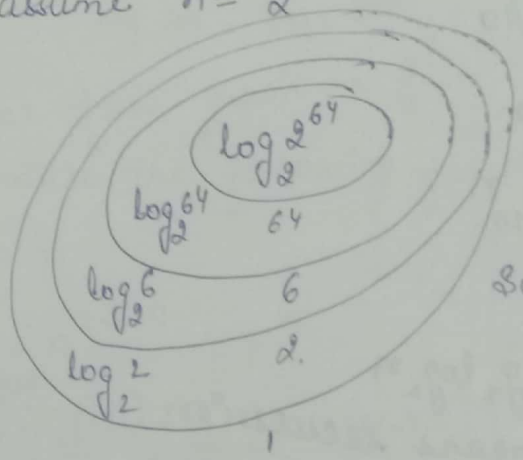
Ques consider the following 2 functions

$f(n) = \log^*(\log n)$

$g(n) = \log(\log^* n)$ - then find relation b/w $f(n)$ & $g(n)$

Soln

NOTE assume $n = 2^{64}$



To make $\log_2^* 2^{64} = 1$ we have applied log for 4 times

so $\log_2^* 2^{64} = 4$

$\log_2 2^{64} = 64 > \log_2 2^6 = 6 > \log_2^2$

Now to solve the above problem assume

$$n = \underbrace{2^{2^{2^{\cdots^{2}}}}}_{65,536}$$

$\log^* n = 65536$

$f(n) = \log^*(\log n)$

$= \log^* 65,535$

$= 65$

$g(n) = \log(\log^* n)$

$\log(65536) = 16$

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

ex 1 $f(n) = n - 10$

how many times will we sub 10 from n to make it 1

$f^*(n) = n/10$

ex 2 $f(n) = n/10$

$f^*(n) = \log_{10} n$

ex 3 $f(n) = \sqrt{n}$

$f^*(n) = \log_2 \log_2 n$

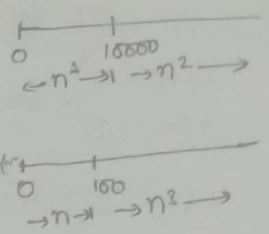
Note $f^*(n)$ here $(*)$ means recursion

Sub 6

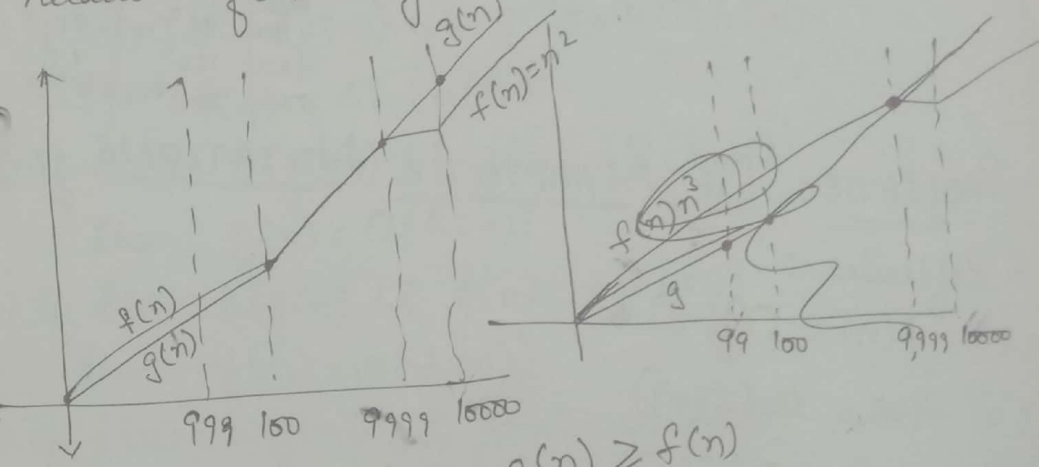
Consider the following two functions

$$f(n) = \begin{cases} n^3 & 0 \leq n < 10000 \\ n^2 & n \geq 10,000 \end{cases}$$

$$g(n) = \begin{cases} n & 0 \leq n < 100 \\ n^3 & n \geq 100 \end{cases}$$



relation $f(n)$ & $g(n)$



clear from graph → $g(n) \geq f(n)$

$n_0 > 9999$

Sub 7 consider the following functions

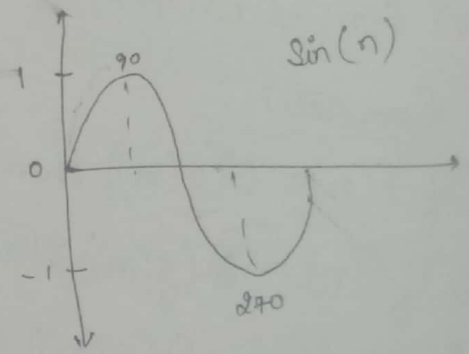
$f(n) = n$

$g(n) = n^{1+\sin n}$

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

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

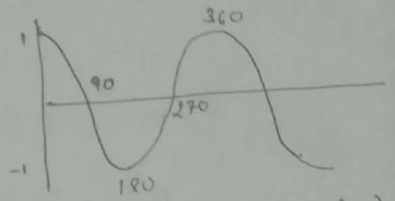
n	f(n)	g(n)
0	0	0
90	90	90^2
180	180	180
270	270	1
360	360	360



f(n) and g(n) are non comparable

Ques 8
what will be relation between the following func.

$f(n) = n^2$
$g(n) = n^{1+\cos n}$
$f(n) \geq g(n)$



n	f(n)	g(n)
0	0	$n^2 = 0$
90	90^2	90
180	180^2	↑
270	270^2	270

so $f(n) = \Omega g(n)$
$n_0 \geq 0$

PROPERTIES OF ASYMPTOTIC NOTATIONS

1. Reflexive :
(i) $f(n) = O(f(n))$
(ii) $f(n) = \Omega(f(n))$
(iii) $f(n) = \Theta(f(n))$
(iv) $f(n) \neq o(f(n))$
(v) $f(n) \neq \omega(f(n))$

2) Symmetric
(i) if $f(n) = O(g(n))$ then $g(n) \neq O(f(n))$
(ii) if $f(n) = \Omega(g(n))$ then $g(n) \neq \Omega f(n))$
(iii) if $f(n) = \Theta(g(n))$ then $g(n) = \Theta f(g(n))$

(iv) if $f(n) = o(g(n))$ then $g(n) \neq o(f(n))$
(v) if $f(n) = \omega(g(n))$ then $g(n) \neq \omega f(n))$

3). Transitive
(i) if $f(n) = O(g(n))$ & $g(n) = O(h(n))$
then $f(n) = O(h(n))$
(ii) if $f(n) = \Omega(g(n))$ && $g(n) = \Omega(h(n))$
then $f(n) = \Omega(h(n))$
(iii) if $f(n) = \Theta(g(n))$ && $g(n) = \Theta(h(n))$
then $f(n) = \Theta(h(n))$
(iv) if $f(n) = o(g(n)$ && $g(n) = o(h(n))$
then $f(n) = o(h(n))$
(v) if $f(n) = \omega(g(n))$ && $g(n) = \omega(h(n))$
then $f(n) = \omega(h(n))$

4). If $f(n) = O(g(n))$
then
$f(n) \cdot h(n) = O(g(n) \cdot h(n))$

5). If $f(n) = O(g(n))$ && $d(n) = O(e(n))$
then
(i) $f(n) + d(n) = O[\max(g(n), e(n))]$

(ii) $f(n).d(n) = O(g(n).e(n))$

Ques1. Let $f(n), g(n)$ & $h(n)$ be three positive functions which are ~~dif~~ defined as follows.

(i) $f(n) = O(g(n))$ && $g(n) \neq O(f(n))$ $\Rightarrow g(n) > f(n)$

(ii) $g(n) = O(h(n))$ && $h(n) = O(g(n))$ $g(n) = h(n)$

then check following stmts are true/false

(a) $f(n) = O(h(n))$. True

(b) $f(n) * g(n) = O(g(n) . h(n))$ True

(c) $g(n) . g(n) = O(g(n) . h(n))$ True

(d) $h(n) . f(n) = O(g(n), h(n))$ True

Ques2 Suppose $T_1(n) = O(f(n))$ && $T_2(n) = O(f(n))$

check the following stmts $T_1 \lesseqgtr_{T_2} f(n)$

(a) $T_1(n) + T_2(n) = O(f(n))$ True

(b) $T_1(n) = O(T_2(n))$ False

$T_2(n) = O(T_1(n))$ False

$T_1(n) = O(T_2(n))$ False

} can't be determined

DIVIDE & CONQUER (DAC)

1. Recursion
2. Recurrence Relation
3. Recurrence Relation Solving

RECURSION

A function is calling itself to solve a particular problem is called a recursion.

ex: $f(6) \rightarrow 6 \times f(5) \rightarrow 5 \times f(4) \rightarrow 4 \times f(3) \rightarrow 3 \times f(2) \rightarrow 2 \times f(1) \rightarrow$

Solving of Bigger Problems into smaller problems.

Note:

1. Recursion is nothing but solving big problems in terms of smaller problem.
2. To execute the recursive programs we use stack data structure.
3. Every recursive prog should have termination condition otherwise prog will go to infinite loop and finally it will produce a error message stack overflow.
4. In the recursive prog from one func called another func the value / parameter will be changed but not no. of parameter. otherwise the prog will go to infinite loop & finally will produce stack overflow error msg.

5. Comparing recursive and non recursive prog, recu... prog take more stack space because of more ... function calls.

6. For every recursion prog equivalent non recu... prog is possible with the help of loop (for, while loop)

7. Recursive prog are very easy to write comparison ... to non recursive.

From computer pt. of view non recursive prog are ... best while users " " " recursive prog are best

Q WA.P (Recursive) in recurrence relation to factorial of n.

```
fact (n)
{
  if(n==1) return (1)
  else
    return (n*fact(n-1));
}
```

Recurrence Relation

W.A.P & recurs... b/w 2 +ve no...

```
multipl...
{
  if(...
  else
  ...
}
```

```
mul...
{
  if(...
  else
  ...
}
```

T (multiply (m...

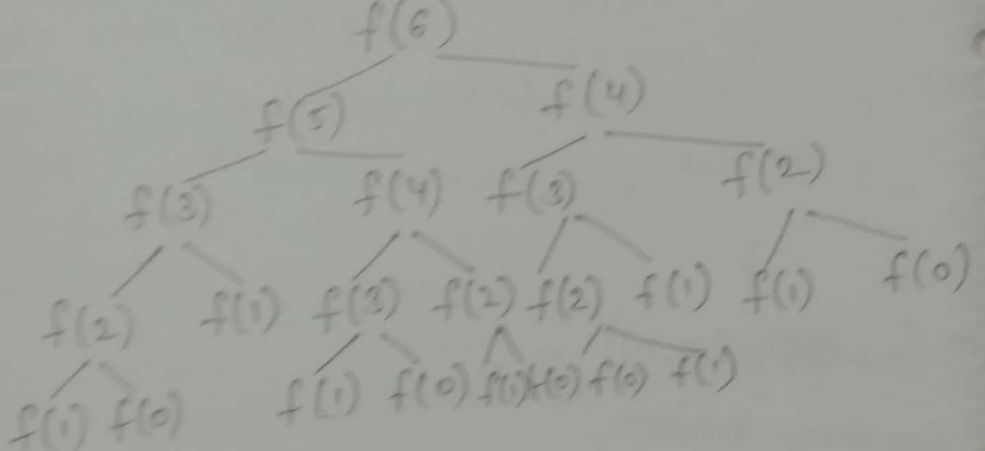
Recurrence Relat...

multiply(m,n)

WAP (recursive) to find nth fibonacci no.

```
fib(m, n, k)
{
  if(k=0)
    return(n)
  else
    return(n, m+n, k-1);
}
```

m n
0 1
1 m+n
0 1 1 2 3 5 8 13




```
fib(n)
{
  if(n==0||n==1)
    return n
  else
    return(fib(n-1)+fib(n-2))
}
```


Recurrence Relation

$$f(n) = \begin{cases} n & \text{if } n=0 \text{ or } n=1 \\ fib(n-1)+fib(n-2) & \text{otherwise} \end{cases}$$

$T(fib(n)) = O(2^n)$ — n level

↳ because binary tree is formed.

W.A recursive prog and recurrence relation to find gcd of two +ve no m & n.

```
gcd(m,n)
{
  if(m==0&&n==0)return(∞)
  if(m==0)
    return(n)
  if(n==0)
    return(m)
  return(m%n, m);
}
```

$$gcd(m,n) = \begin{cases} \infty & \text{if } m=0 \& n=0 \\ n & \text{if } m=0 \\ m & \text{if } n=0 \\ gcd(n\%m, m) & \text{otherwise} \end{cases}$$

$T(gcd(m,n)) = \log n$

GCD(23,29)

23)29(1
23
6

GCD(6,23)

6)23(3
18
5

GCD(5,6)

5)6(1
5
1

GCD(1,5)

1)5(5
5
0

gcd(0,1)
g

* if the given no.s are prime no. i.e the gcd of prime no is to be find then the gcd algo is in worst case.
* If both the no. are multiples of each other. then gcd algo would give the best case.

RECURRENCE RELATION SOLVING

1. Subsitution Method
2. Recursive Tree Method
3. Master - Theorem

SUBSTITUTION METHOD

Substituting the given function repealedly until given func. is removed.

ex 1

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

$T(50) = T(49)+50$

$T(48)+49+50$

$T(47)+48+49+50$

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

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

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

$\downarrow$ 10

$T(n-10)+(n-9)+(n-8)+\ldots+(n-1)+n$

$\downarrow n-1$

$= T(n-(n-1)+n-(n-2)+n-(n-3)+n-(n-4)+\ldots+(n-1)+n$

$T(1)+2+3+4+5+\ldots+(n-1)+n$

$1+2+3+4+\ldots+n$

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

$\log a + \log b + \log c = \log abc$

2x2

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

$T(10) = T(9)+\log 10$

$= T(9-1)+\log 9+\log 10$

$= T(7)+\log 8+\log 9+\log 10$

$\downarrow$

$T(1)+\log 2+\log 3+\ldots+\log 10$

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

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

$= T(n-3)+\log(n-2)+\log(n-1)+\log n$

$\downarrow$

$= T(n-(n-1))+\log(n-(n-2))+\log(n-(n-3))$

$+\ldots+\log(n-2)+\log(n-1)+\log n$

$= 1+\log 2+\log 3+\log 4+\ldots\log(n-1)+\log n$

$= 1+\log(2.3.4.5\ldots n)$

$1+\log(n!)$

$1 + \log n!$

$1 + \log n^n$

$1 + n \log n \Rightarrow O(n \log n)$

upper bound of $n! = n^n$

$n \times (n-1) \times (n-2) \times \ldots 1$

↓ ↓ ↓

assume 'n' at each place

so upper bound

$n \times n \times n \times \ldots$ (n times) $= n^n$

ex: $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2) + \log(n) & \text{if } n>0 \end{cases}$

$T(10) = T(8) + \log 10$

$T(6) + \log 8 + \log 10$

$T(4) + \log 6 + \log 8 + \log 10$

$T(2) + \log 4 + \log 6 + \log 8 + \log 10$

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

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

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

$T((n-2)-(n-3)) + \ldots + \log(n-6) + \log(n-4) + \log(n-2) + \log n$

$1 + \log 2 + \log 4 + \log 6 + \ldots + \log(n-2) + \log n$

$n - 2k = 0$

$k = n/2$

$1 + \log(2.4.6 \ldots n)$

$1 + \log 2^n (2.3.4.5.6 \ldots n/2)$

$T(n-2k) + \log(n-(2x-2)) + \log(n-(2x-4)) + \ldots \log(n-(2k-6)) + \ldots + \log n$

$= 1 + \log 2 + \log 4 + \log 6 + \ldots + \log n$

$1 + \log(2.4.6 \ldots n) = 1 + \log(2^n (1.2.3 \ldots n/2))$

$(n-2) - k = 1$

$n - 3 = k$

$\log a \cdot \log b = \log a + \log b$

$1 + \log(2.1) + \log(2.2) + \log(2.3) + \ldots \log(2 \times n/2)$

$1 + \log 2 + \log 1 + \log 2 + \log 2 + \log 2 + \ldots \log 2 + \log n/2$

$1 + n/2 + \log_2^2 + (\log 1 + \log 2 + \ldots \log n/2)$

$1 + n/2 + \log(1.2.3.4 \ldots n/2)$

$1 + n/2 + \log(n/2)!$

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

$= O(n \log n)$

Ques $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2) + n^2 & \text{if } n>0 \end{cases}$

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

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

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

Ques $T(n) = \begin{cases} 2 & \text{if } n=2 \\ \sqrt{n}\, T(\sqrt{n}) + n & \text{if } n>2 \end{cases}$

① $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-2)+n^2 & \text{if } (n>1) \end{cases}$

$\Rightarrow T(n) = 1 + 2^2 + 4^2 + 6^2 + 8^2 + \ldots k^2$

$= \frac{(n^2)(2n^2+1)(n^2+1)}{6}$

$\Rightarrow O(n^6)$

Q3

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

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

$= 8\left[8T(n/2^2) + (n/2)^2\right] + n^2$

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

$= 8^3 T(n/2^3) + (n/2^2)^2] + 2n^2 + n^2$

$= 8^3 T(n/2^3) + 4n^2 + 2n^2 + n^2$

$= 8^k T(n/2^k) + 2^2 2^2 n^2 + 2^1 n^2 + 2^0 n^2$

$n/2^k = 1 \Rightarrow n = 2^k \Rightarrow \log n = k \log_2 2$

$\Rightarrow 8^{\log_2 n} T(n/2^k) + n^2[2^{k-1} + 2^{k-2} + \ldots + 2^1 + 2^0]$

$\Rightarrow 8^{\log_2 n} T(1) + n^2\left[2^0 + 2^1 + \ldots + 2^{\log n}\right]$

$n^3 \times 1 + n^2\left[1\frac{(2^{\log n}-1)}{2-1}\right]$

$= n^3 + n^2(n-1) = n^3 + n^3$

$= 2n^3 = O(n^3)$

(2)

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

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

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

$$= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2} + n$$

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

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \log n = k$$

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

$$= T(1) + n\left[\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \left(\frac{1}{2}\right)^{\log n}\right]$$

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

$$= 1 + n\left[\frac{1-0}{1/2}\right] = 1 + 2n$$

$r < 1$ → decreasing

whenever decreasing G.P series is there, G.P series is 1

(4) $$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n/2) + c & \text{if } n>1 \end{cases}$$ where c is const

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

$$= T\left(\frac{n}{2^2}\right) + c + c$$

$$= T\left(\frac{n}{2^3}\right) + c + c + c.$$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \log n = k$$

$$T\left(\frac{n}{2^{\log n}}\right) + c \log n$$

$$1 + \log n\, c$$

$$O(\log n)$$

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

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

$$= 2\left[2\left[2T\left(\frac{n}{2^3}\right)\right] + \frac{n/2^2}{\log\left(\frac{n^2}{2}\right)}\right] + \frac{n/2^2}{\log n/2^2} + \frac{n}{\log n}$$

$$2^3\left(T\left(\frac{n}{2^3}\right)\right) + \frac{n/2^2}{\log(n^2/2)} + \frac{n/2^2}{\log(n/2^2} + \frac{n}{\log}$$

$\Rightarrow 2^3 T\left(n/2^3\right) + \frac{n}{\log n/2^2} + \frac{n}{\log n/2^1} + \frac{n}{\log n}$.

$\Rightarrow 2^3 T\left(n/2^0\right) + n\left[\frac{1}{\log n/2^2} + \frac{1}{\log n/2^1} + \frac{1}{\log n/2^0}\right]$

$\Rightarrow 2^k T\left(n/2^k\right) + n\left[\frac{1}{\log n/2^0} + \frac{1}{\log n/2^1} + \frac{1}{\log n/2^n} + \cdots + \frac{1}{\log n/2^{k-1}}\right]$

$\Rightarrow 2^{\log_2 n} T(1) + n\left[\frac{1}{\log n/2^0} + \frac{1}{\log n/2^1} + \frac{1}{\log n/2^n} + \cdots + \frac{1}{\log n/2^{\log n - 1}}\right]$

$\Rightarrow n + n\left[\frac{1}{\log_2 n - 0} + \frac{1}{\log_2 n - 1} + \frac{1}{\log_2 n - 2} + \cdots + \frac{1}{\log_2 n - (\log_2 n - 1)}\right]$

$n + n[$

Q5 $T(n) = \begin{cases} 2 & \text{if } n=2 \\ \sqrt{n}\, T(\sqrt{n}) + n & \text{if } n>2 \end{cases}$

$T(n) = \sqrt{n}\, T(\sqrt{n}) + n$

$= n^{1/2} T(n^{1/2}) + n$

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

$= n^{3/2^2}\left(T(n^{1/2^2}) + n + n\right)$

$= n^{3/2^2}\left[n^{1/2^3} T(n^{1/2^3}) + n^{1/2^2} + n + n\right]$

$= n^{7/2^3} T(n^{1/2^3}) + n + n + n$

$n^{\frac{1}{2^k}} = 2$

$\frac{1}{2^k} \log n = \log_2 2 = 1$

$\log n = 2^k$

$\log \log n = k \log_2 2$

$\log \log n = k$

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

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

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

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

$= \frac{n}{2} \cdot 2 + n \log \log n$

$\Rightarrow n + n \log \log n \Rightarrow O(\log \log n)$

$n^{1/2^k} = 2^{\frac{1}{2^k} \log n} = \frac{1}{2^{\log \log n}} = \frac{1}{\log n}$

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

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

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

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

$T(10) = 2T(9)+1$

$= 2(2T(8)+1)+1$

$= 2(2(2T(7)+1)+1)+1$

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

$(2(2+1)+1)$

$6+1$

$n-k=1$

$k=n-1$

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

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

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

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

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

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

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

$= 2^3T(n-3)+2^2(n-2)+2(n-1)+n$

$\Rightarrow 2^{n-1}T(1)+2^{n-2}T(2)+\ldots 2^2(n-2)+2(n-1)+n$ → AP

$\Rightarrow 2^0(n-0)+2^1(n-1)+\ldots 2^{n-2}(2)+2^{n-1}(1)$ → G.P.

To solve A.P, G.P series (i.e combination of A.P & G.P)
Convert into GP.
Multiply whole series by 2

~~$2T(n) = 2^1(n-0)$~~ (here nothing)

$2T(n) = 0+2^1(n-0)+2^2(n-1)+\ldots+2^{n-2}.3+2^{n-1}.2+2^n.1$

$T(n)-2T(n) = n-2^1.1-2^2.1-2^3.1\ldots-2^{n-2}.1-2^{n-1}.1-2^n.1$

$-T(n) = n-[2^1+2^2+2^3+\ldots+2^{n-1}+2^n]$

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

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

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

$O(2^{n+1}) \Rightarrow O(n)$

Note

* $T(n) = 2T(n/2) + n$
  ans $n \log n$
* Now $T(n) = 2T(n/2 - 10) + n$

here also the ans = $n \log n$ coz '10' won't effect more

* $T(n) = 2T(\sqrt{n} + n)$
  Neglect 'n'
  $T(n) = 2T(n/2) + 2n$
  ans $\Rightarrow 2n \log n$
  $T(n) = 2T(n/2) + n/2$
  ans $= \frac{n}{2} \log n$

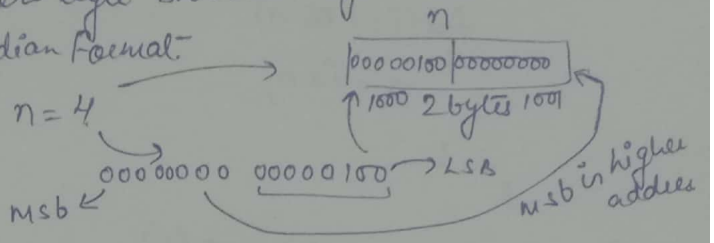
changes are in constant only

* How Recursive prog execute inside the comp.

main memory is 100 times faster than harddisk.

Slower byte stored in lower address (LB-LA)
and higher byte stored in higher address (HB-HA)

Little Endian Format



Space fact(n) = no. of function * One function space
= n * 6 Bytes
= 6n Bytes
= O(n)

Total space = 2 Bytes (in main func) + 6n Bytes (in sub fun fact)
= 6nB
= O(n)

```
main()
{
  int n;
  printf("%d", fact(n));
}
fact(int n)
{
  int a, b;
  if (n==1)
    printf(" return(1);
  else
  {
    a = fact-(n-1)
    b = a*n;
    return b;
  }
```

fact(n) Time = no. of func * Time of
= n * O(1)
= O(n)

for non-recursive factorial sub-prog

```
fact(int n)
{ int a,i;
  for(i=1 to n)
  {
    a = i*n;
  }
}
```

fact(n) space = 6B = O(1)

Total space = 2B + 6B = 8B = O(1)

Recursive prog will take more space but time we can't say (depends on logic)

DIVIDE & CONQUER (DAC)

It is used to solve big problems

1) Divide the given big problem into smaller size problems.
2) Conquer the sub-problems by calling recursively so that we will get sub problem solutions
3) Combine the sub-problem solutions to get original problem solution.

CONTROL ABSTRACTION OF DAC

```
DAC (a,P,q)
{
    if (small P,q,n)
        return (solution (a,P,q))
    else
        m = Divide (a,P,q)
        b = DAC (a,P,m)
        c = DAC (a,m+1,q)
        d = Combine (b,c)
        return (d)
}
```

How to find time Complexity of any problem if it is ~~used~~ to solved by divide and conquer.

$$T(n) = \begin{cases} O(1) & \text{if } n \text{ is small} \\ \text{if } n \text{ is big} & \\ \text{Divide} \Rightarrow f_1(n) & \\ n/2 \quad n/2 & \\ T(n/2) \quad T(n/2) & \\ \text{Combine } f_2(n) & \end{cases}$$

$$T(n) = \begin{cases} O(1) & \text{if } n \text{ is small} \\ f_1(n) + T(n/2) + T(n/2) + f_2(n) & \text{if } n \text{ is big.} \end{cases}$$

$$T(n) = 2T(n/2) + f_1(n) + f_2(n)$$

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

Ingeneral DAC Recurrence relation will appear look like as

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

$a \rightarrow$ no. of subproblems

n/b = size of subproblem

$f(n)$ = Divide and conquer func.

APPLICATIONS OF DAC

1) Find maxmin
2) Power of an element
3) Binary Search
4) Merge Sort
5) Quick Sort
6. Selection procedure
7) Strassen's Matix Multiplication.

1). Find MaxMin.

i/p: An array of n elements

o/p: Return Maximum element & & Minimum eleme

ex:

i/p: A[10, 20, 30, 11, 21, 31, 1, 2, 3]

o/p: min → 1
max → 31

without divide and conquer.

```
straightmaxmin(a, 1, n)
{
    max = min = a[1];
    for(i=2 ; i<=n; i++)
    {
        if(a[i]< min)
            min = a[i]
        else if(a[i] > max)
            max = a[i]
    } return(max, min)
}
```

Best case = n-1

worst case = n-1

Let $T(n)$ be the no. of comparisons for n-element in the above algo.

* Best case : Min. ant. of time req. to solve a problem.
No failure at all (always true)

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

(no. of comparisons)

* Worst case : max. ant of time req.

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

(no. of comparison)

Average case :

$$T(n) = \underset{\downarrow}{1}\cdot\frac{(n-1)}{2} + \underset{\downarrow}{2}\left(\frac{n-1}{2}\right) = \frac{3}{2}(n-1)$$

only first 'if' checked for half time

Both 'if' checked for half time

Best case $\leq$ Average case $\leq$ Worst case

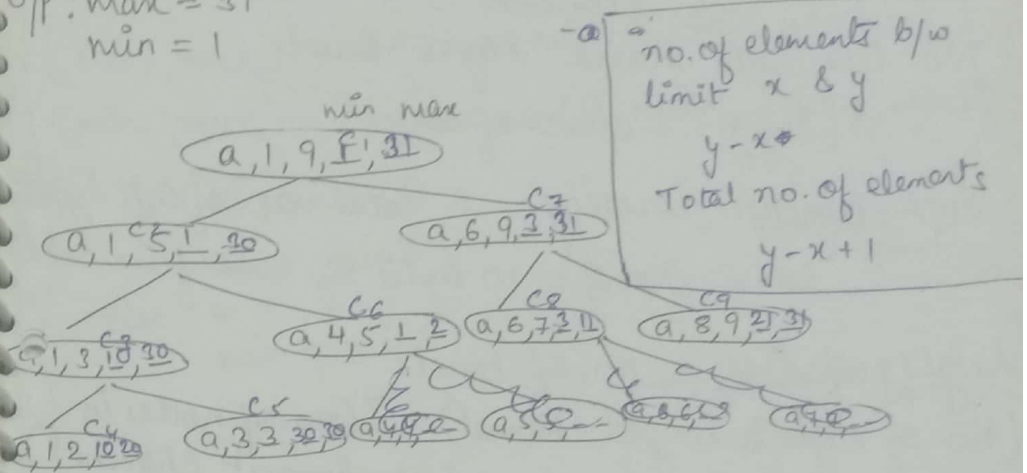
In all the cases here time complexity $= O(n)$

Space complexity = for fun + for main func
12 Bytes + 2 × n leyles
$2n + 12 = O(n)$

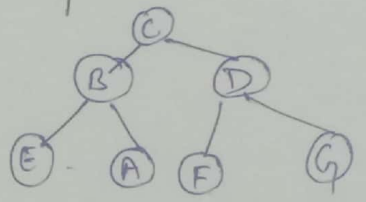
* With DAC

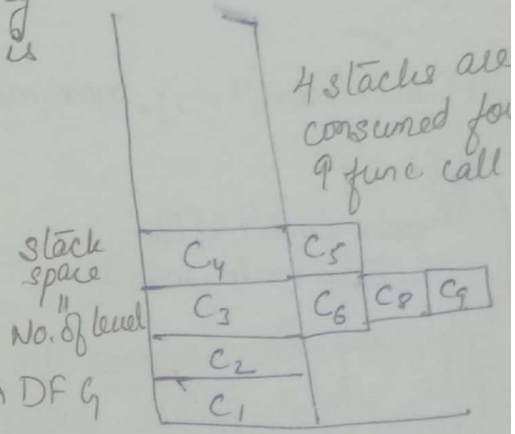
i/p : A[10, 20, 30, 1, 2, 3, 11, 31, 21]
1 2 3 4 5 6 7 8 9

o/p : max = 31
min = 1



In all programming lang func calling sequence is called pre-order.





calling : Preorder : CBEADFG

Completion : Postorder : EABFGDC

In every prog. lang. func. execution order is post order

Note :
No. of levels in function call = no. of stacks
In above ex → only 4 (i.e one stack for one level function

tree is formed by dividing no. of element by 2.
that means for 'n' element each element will take $n/2$ elements.
So. tree will have $\log n$ levels.

$O(\log_2 n)$ = space of stack

Each function is using = 5 local var. of int type
so = $2 \times 5 = 10$ bytes by each func call

* height of tree = no. of levels − 1
(no. of edges in longest channel) → (no. of nodes in longest channel)

```
DAC maxmin(a, i, j, max, min)
{
  if(i==j)
  {
    max = min = a[i]
    return (min, max);
  }
  if(i==j-1)
  {
    if(a[i] > a[j])
      max = a[i], min = a[j]
    else
      max = a[j], min = a[i]
    return (min, max);
  }
  else
  {
    mid = ⌊(i+j)/2⌋        } divide statement
    (min1, max1) = DAC maxmin(a, i, mid, max, min)
    (min2, max2) = DAC maxmin(a, mid+1, j, max, min)
    if (min1 > min2)
      min = min2
    else
      min = min1
    if (max1 > max2)
      max = max1
    else
      max = max2
    return (min, max);
  }
}
```

conquer stmts

combining stmts

Let $T(n)$ be the no. of comparisons required for the n-elements array. using above algo then, Recurrence Relation

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

for small

divide → 0 ; conquer stmts → $2T(n/2)$; combining → 2 (no. of subproblems)

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

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

$= 2\left(2\left(2T\left(\frac{n}{2^3}\right)+2\right)+2\right)+2$

$T(16) = 2T(8)+2$

$= 2(2T(4)+2)+2$

no. of level $= 2^3 T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2$

extra space $\leftarrow$ k $\rightarrow$ $= 2^k T\left(\frac{2n}{2^k}\right) + 2^k + 2^{k-1} + \ldots 2^3 + 2^2 + 2$

$= 2^{\log_2 n - 1} T\left(\frac{n}{2^{\log_2 n - 1}}\right) + 2 + 2^2 + 2^3 + \ldots 2^{\log_2 n - 1}$

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

$\frac{n}{2} \cdot 1 + \left[2\left(\frac{2^{\log_2 n - 1} - 1}{2-1}\right)\right]$

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

$\downarrow$ comparison + time also

Space consumed = no. of local variables * 2

10 * 2 = 20 Bytes

$n/2^k = 2$

$n = 2^{k+1}$

$\log n = k+1 \log 2$

$\log n + 1$

Total space $\Rightarrow$ i/p + extra stack byte

$2nB + \log nB$

$\Downarrow$

$2n \Rightarrow O(n)$

Let $T(n)$ be the comparison between the elements not between the position.

{ Best case time = Average case = Worst case time is above algo (coz in else part we cannot skip any of the stmt.)

space = n (array) + $\log n$ (stack extra space.)

$\Rightarrow O(n)$

Note:

Comparing straightmaxmin & DACmaxmin straightmaxmin is better bcoz both the times are $O(n)$ and DACmaxmin will take more space in the form of stack.

POWER OF AN ELEMENT

i/p : 2 +ve integers a & n where a>1, n>1

o/p : Return a^n

* Predefined func (or any thing) takes constant time while calculating execution time

Without DAC

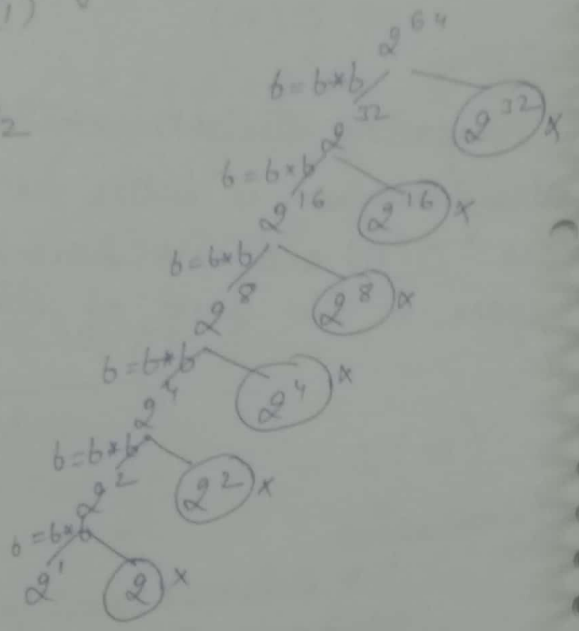
```
s=1;
for(i=1 to n)
  s=s*a;
```

Space consumed = 4 variable
= 4×2 Bytes = 8 Bytes
= O(1)

WITH DAC

$a^n = a^{n/2} \cdot a^{n/2}$

```
DACPOW(n, p)
{
  if(n==1)
    return(p)
  else
    mid = n/2;
    DACPOW(mid
```




```
Pow(a,n)
{
  if(n==1)
    return a;
  else
  {
    mid = n/2;
    pow=(a,mid);
    c=b*b
    return c;
  }
}
```

Let T(n) be the no of multiplications

Recurrence Relation :-

$$T(n) = \begin{cases} n=0 & \text{(small problems)} \\ T(n/2)+1 & \text{(big problems) if } n>1 \end{cases}$$

① $T(n) = T(n/2) + 1$

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

$= T(n/8) + 1 + 1 + 1$

To — k

$T(n/2^k) + 1 + 1 + \ldots$ k times 1

$T(n/2^{\log n}) + 1 + 1 + \ldots + 1$ k times last time

$= T(1) + \log_2 n + 1 \cdot k = 0 + \log_2 n$

$= \log_2 n = O(\log_2 n)$

$n/2^k = 1$

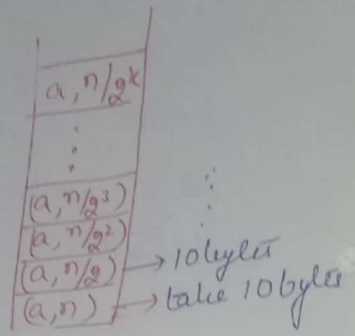
$n = 2^k$

$\log n = k \log 2$

$\log n = k$

In the else case before return statement there is no exit/break/return so we cannot skip the 3 stmts in else part. Because of this average = worst = best case are same.

Total Space = i/p + extra stack space
= 4Bytes + stack of $10\log n$
= $10 \log n$
= $O(\log n)$



If the power is not in the form of 2^x eg: 64, 128, etc we can make a sub function named "adjustment" where we can find the nearest 2^n of power of that value.

eg: 2^{100} ⇒ adjustment ; find 2^{64} $2^{64}/2^{36}$,

becoz
2^{100}
2^{50}
2^{25}
call adjustment

SEARCHING

* **Linear Search**

i/p: array of n element, element-x
o/p: position of x-element : if x is found
else
return (-1)

ex:
i/p: A[10, 20, 30, 1, 2, 3, 31, 21, 11] (1 2 3 4 5 6 7 8 9)

x = 31	x = 50
o/p: 7	-1

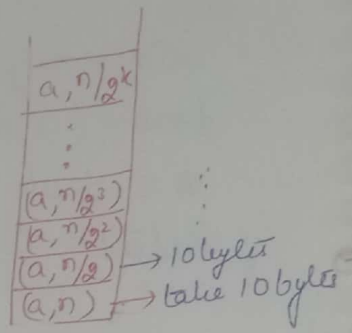
No. of Comparisons
Best case: x at first position
: O(1)
worst case: O(n)
Average case: $\frac{n(n+1)/2}{n} = \frac{(n+1)}{2}$

* **Binary Search**

	Graph	Tree
1)	Combination of (V, E)	Combination of (V, E)
2)	No root element.	Always root
3)	Connected / Disconnected	Always connected
4)	It is directed / undirected	Always directed
5)	May be cyclic / non cyclic.	It don't have cy

In the else case before return statement there is no exit/break/return so we cannot skip the 3 stmts in else part. Because of this average = worst = best case are same.

Total Space = i/p + extra stack space
= 4 Bytes + Stack of 10 log n
= 10 log n
= O(log n)



* If the power is not in the form of 2^x eg: 64, 128, etc we can make a sub function named "adjustment" where we can find the nearest 2^x of power of then value.

eg: 2^{100} ⇒ adjustment
? find 2^{64}
$2^{64}/2^{36}$

becoz

2^{100}
2^{50}
2^{25}
? call adjustment

SEARCHING

* Linear Search

i/p: array of n element, element-x
o/p: position of x-element : if x is found
else
return (-1)

ex:

i/p: A[10, 20, 30, 1, 2, 3, 31, 21, 11]
(1) (2) (3) (4) (5) (6) (7) (8) (9)

x = 31	x = 50
o/p: 7	-1

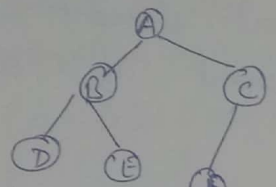
No. of Comparisons

Best case: x at first position
: O(1)
worst case: O(n)
Average case: $\frac{n(n+1)/2}{n} = \frac{(n+1)}{2}$

* Binary Search

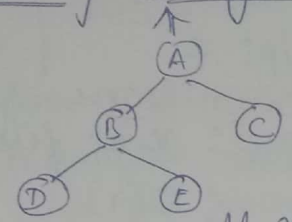
	Graph	Tree
1).	Combination of (V, E)	Combination of (V, E)
2).	No root element.	Always root
3).	Connected / Disconnected	Always connected
4).	It is directed / undirected	Always directed
5).	May be cyclic / non cyclic.	It don't have cycl

* In the given tree, everyone has no more than 2 children. So it is a binary tree.



* No node / vertices ⇒ empty binary tree.

* In a binary tree if it can have 0 or 2 or 0 child only then it is strictly binary tree.



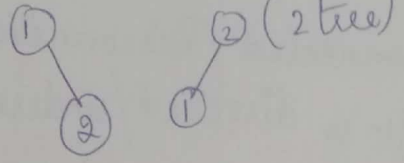
Binary Search Tree

In the given binary tree comparing root data all the element present on left hand side is smaller and all element present on right side are greater people call it BST.

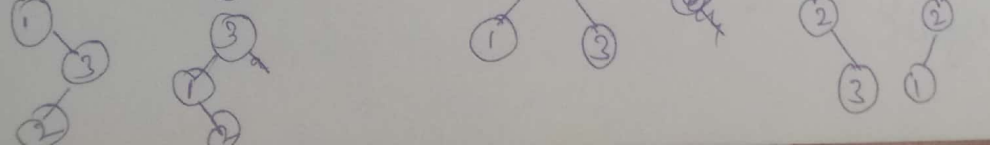
1) How many BST possible with n distinct nodes

(a) $n=1$ ⇒ 1 tree

(b) $n=2$ ⇒ (1) (2)

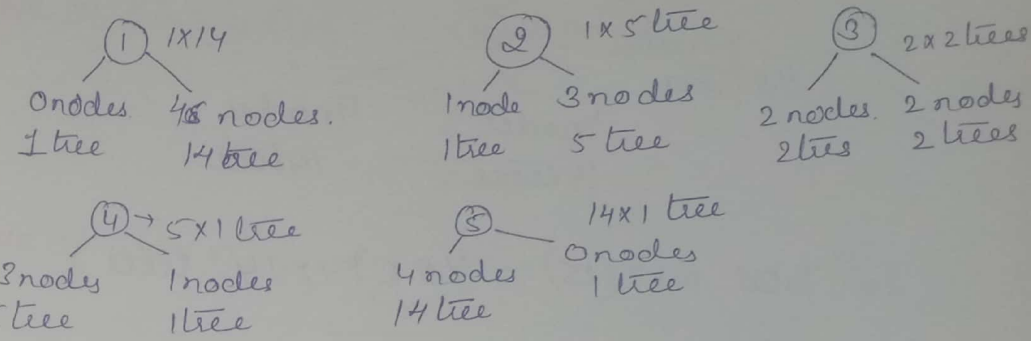


(c) $n=3$ ⇒ (1,2,3) ⇒ 5 tree



(d) $n=4$ (1,2,3,4)

(3) $n=5$ (1,2,3,4,5) BST?



$14+5+4+5+14 = 42$ BST

no. of BST with n nodes $= BST(n) = \sum_{i=1}^{n} BST(i-1).BST(n$

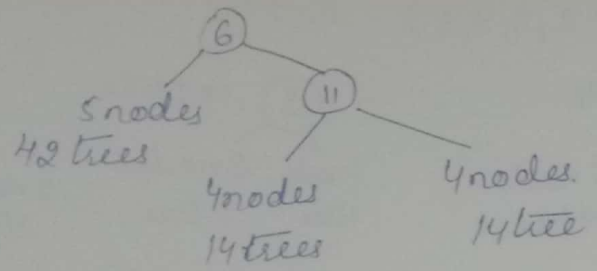
(we will assume i as root node each time we calculate eg: node 1 as root

$BST(0).BST(n-0)$

Or

$= BST(\text{left side nodes}).BST(\text{Right side nodes})$

? How many BST with nodes 15 and labels (1,2,3,4,5 ... 15) in such a way that root node is 6 and right hand side root node is 11



So Total BST(15) = 42 × 14 × 14 trees.

No. of Binary tree = No. of BST × n!

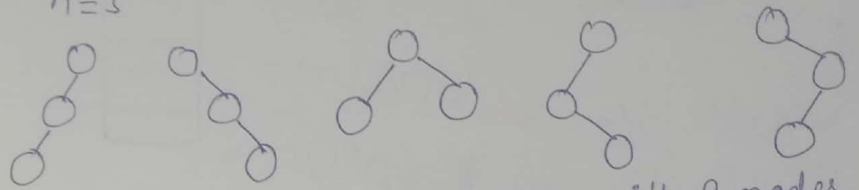
$$= \frac{{}^{2n}C_n}{n+1} \times n!$$

$$\text{No. of BST}(n) = \frac{{}^{2n}C_n}{n+1}$$

$\frac{BT}{2n}$

Q. How many unlabelled binary tree possible with n nodes?

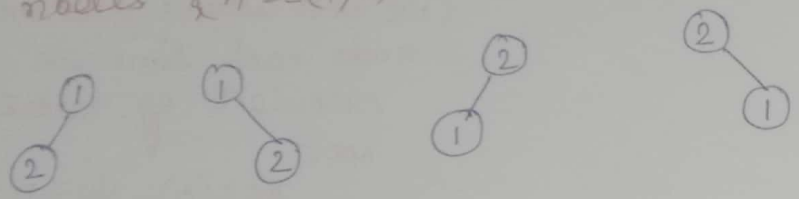
n = 3



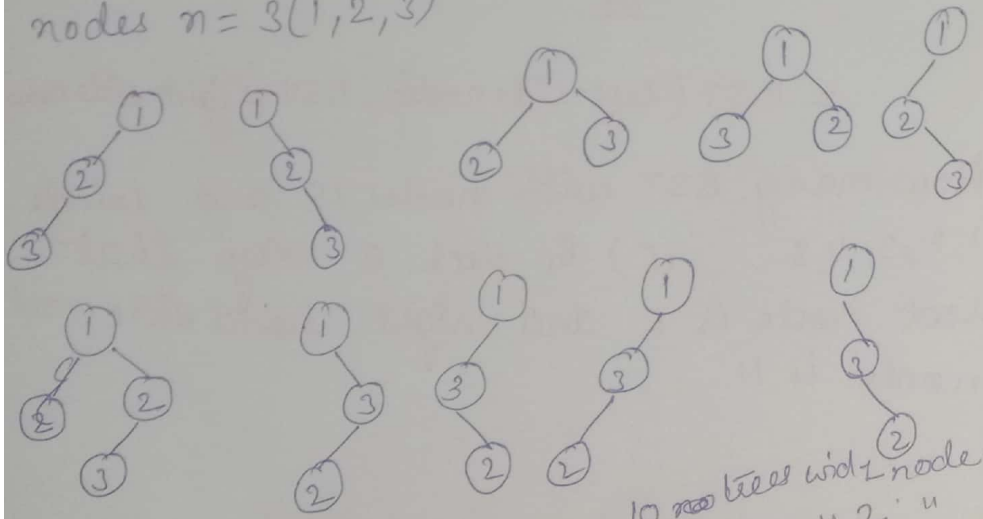
5 unlabelled BT with 3 nodes.
& shape different

Q How many Binary tree possible with n nodes {n = 2(1,2)

4 trees



nodes n = 3(1,2,3)



= 10 trees with 1 node as root
10 " " 2 " " "
10 " " " " "
30

no. of BST(n) = no. of unlabelled Binary tree(n)

Ques i/p: A set with n-element and an unlabeled BT with n node

o/p: How many ways you can insert lables into unlabeled BT so that it will become BST

(a) 0 (b) 1 (c) n! (d) $\frac{{}^{2n}C_n}{n+1}$

(if not mentioned then 'b' answer should be answer)

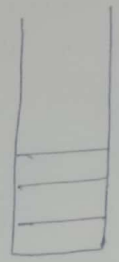
unlabelled BT & BST are same in diagram.

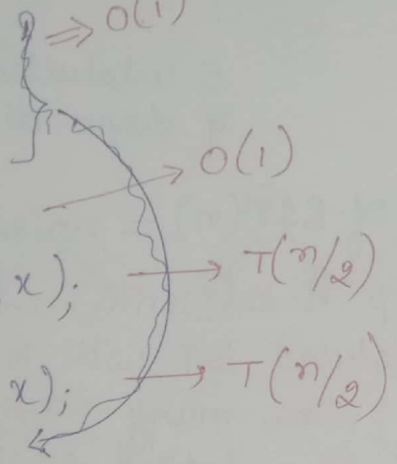
Binary Search Algo

i/p: Sorted array of n elements, element - x
o/p: Position of x-element

ex: i/p: A[10, 20, 30, 40, 50, 60, 70] x = 40

```
BS(a, i, j, x)
{
  if(i==j)                          } constant time
    if(a[i]==x)                     } O(1)
      return(i)                     }
    else                            }
      return(-1)                    }
}
else
{ mid=(i+j)/2.;          ⇒ O(1)
  if(a[mid]==x)          ⇒ O(1)
  {
    return mid;
  }
else
  if(a[mid]>x)
    BS(a, i, mid-1, x);
  else
    BS(a, mid+1, j, x);
}
```





Let T(n) be the time complexity on n element using above algo. Then

recurrence Relation, T(n)

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

In binary search there is divide, conquer but no combine that why it is also called partial divide and conquer algo.

$$T(n) = T(n/2) + O(1)$$
$$= T(n/2) + C$$
$$= T(n/2^2) + C + C$$
$$= T(n/2^3) + C + C + C.$$
$$\vdots$$
$$= T(n/2^k) + \log n\, C.$$
$$= T(n/2^{\log n}) + \log n\, C.$$
$$= T(1) + \log n\, C$$
$$O(1) + C.\log_2 n = C.\log_2 n$$
$$= O(\log n)$$

for last level

$n/2^k = 1$
$n = 2^k$
$\log n = k$

Worst case
Average case

for best case

$$T(n) = O(1)$$

Total Space = i/p + extra space

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

Sum of

Average case;

$$(1 + 2 + 3 + \ldots + \log n)$$

$$\log n \frac{(\log_n n + 1)}{2} = \frac{\log n + 1}{2}$$

$$= O(\log n)$$

In case of linear search will take more space in comparison to binary searching

If the array would not be sorted $= O(n \log n)$

Q// If: A sorted array of n-distinct element find any ele in A[i] such that a[i] = i

(Best algo, worst case time.

A[-50, -40, -30, -20, -10, -2, 0, 3, 6, 8, 11, 15, 20, 44,
0 1 2 3 4 5 6 7 8 9 10 11 12 13

55, 90, 100, 150, 250]
14 15 16 17 18

① Linear Search is $O(n)$ LS ⇒ OS

② BS is possible. P > V go right
else go left.

Ques i/p: An array of n-elements until some place all are integers and afterwords all are infinite

o/p: find position of 1st infinite. i.e 19

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
A[	50	5	26	37	-55	-42	0	76	94	16	-5	-7	20	46	29	90	100	150	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞]

mid = $\frac{33+1}{2} = \frac{34}{2} = 17$

A[17] = 100 ⇒ move right

mid = $\frac{18+33}{2} = \frac{51}{2} = 25$.

A[25] = ∞ ⇒ not 1st infinite move left → A[24] = ∞ so

mid = $\frac{18+24}{2} = \frac{42}{2} = 21$ A[20] = ∞ so

A[21] = ∞ ⇒ not 1st infinite move left

mid = $\frac{18+20}{2} = \frac{38}{2} = 19$

A[19] = ∞ and A[18] = 150 (integer)

So 19 is answer

② i/p: An array of n-ele where until some place all are integers and after words all are infinite (assume n is unknown & after our array all are ∞)

o/p: find position of 1st infinite.

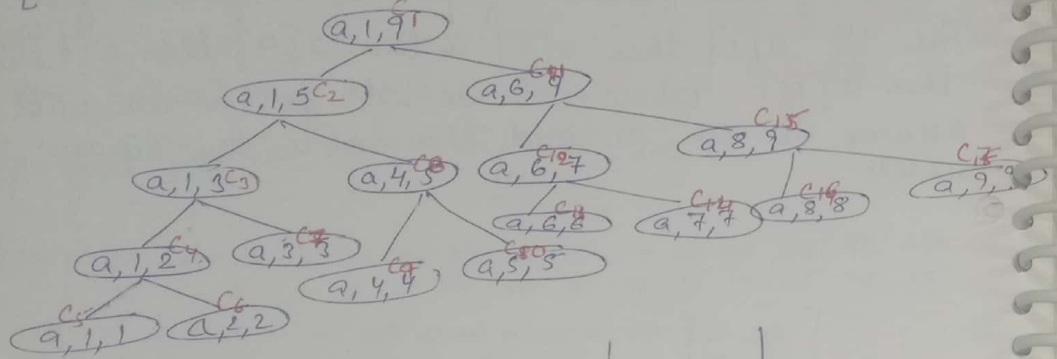
① checking the values of position in power of 2 like 1st a[1] then a[2] then a[4] then a[8] then a[16] where ever we will get ∞ we will assume it as n. and then solve by Binary search

Merge Sort

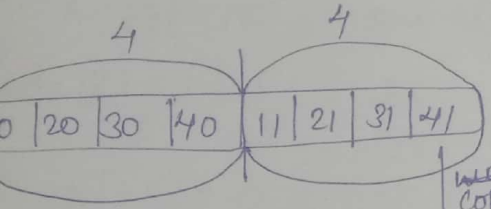
Note: Merging 2 sorted subarrays.

ex:

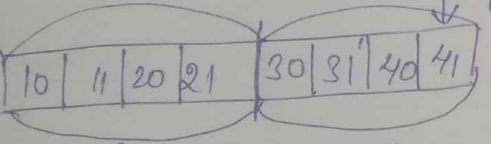
	1	2	3	4	5	6	7	8	9
A[	25	68	16	42	15	91	77	19	20]

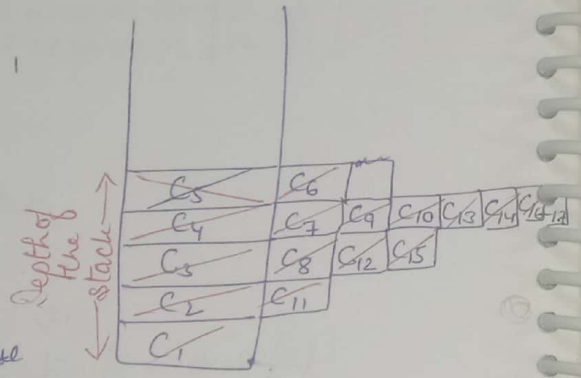


Merge Algorithm



move copy





(10, 11) = 10
(20, 11) = 11
(20, 21) = 20
(30, 21) = 21
(30, 31) = 30
(40, 31) = 31
(40, 41) = 40

comparisons.

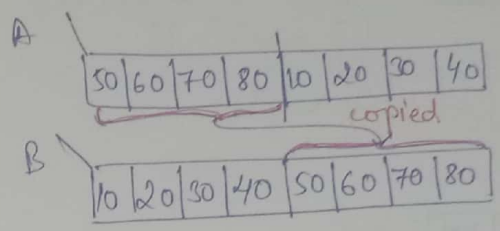
$4+4-1$
$n/2+n/2-1$
$m+n-1$

moves

$4+4$
$n/2+n/2$
$m+n$

whichever is more b/w comparisons & moves. the time complexity = $O(n)$ or $O(m+n)$

Merge Algo best case



(50, 10) = 10
(50, 20) = 20
(50, 30) = 30
(50, 40) = 40

Moves

$4+4$
$n/2+n/2$
$m+n$

comparisons.

$4, 4 \Rightarrow 4$
$n/2, n/2 \Rightarrow n/2$
$m, n \Rightarrow$ whichever is smaller $\min(m,n)$

Time complexity = moves + comparison
= but we will take whichever is larger
= in above case: moves. $(m+n)$
$O(n) = O(m+$

ex:

A (500 elements sorted) B (600 elements sorted)

worst case comparision = $m+n-1 = 500+600-1 = 1099$
best case = 500
movement in best case / worst case = $500+600 = 11$

* Merging two sorted sub array of size m & n req $O(m+n)$
* Because the merging take place in some other algo that's why it is called out-place m algo. (not-in-place)

Time complexity of out-place merge algo = $O(n)$

" " " in-place " " = $O(n^2)$

MergeAlgo [worstcase, Inplace]

A

50	60	70	80	10	20	30	40

(50,10) => 10 swap(50,10)

50	20	30	40

unsorted

so (50,20) = 20 swap

20	50	30	40

unsorted

(50,30) = 30 swap

20	30	50	40

(50,40) = 40 swap

20	30	40	50

so

$n/2$ swaping for $n/2$ times $n/2$

$n/2 \cdot n/2 = n^2/4$

$= O(n^2)$

Merge_Sort (a, i, j)

```
{
  if(i==j)
    return(a[i]);          } O(1)
  else
  {
    mid = [(i+j)/2]        } O(1)
    merge_sort(a, i, mid);   } 2T(n/2)
    merge_sort(a, mid+1, j);
    merge(a, i, mid, mid+1, j);  } O(
    return(a);
  }
}
Merge(
```

Recursion Recurrence Relation (merge_sort_out-place)

Let $T(n)$ be the time complexity of merge sort

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

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

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

$$= 2\left[2\left(2T(n/2^3) + n\right) + n\right] + n$$

$$= 2^3 T(n/2^3) + 3n$$

$$\downarrow k$$

$$2^k T(n/2^k) + k.n$$

$$2^{\log n} T(n/2^{\log n}) + n.\log n$$

$$2^{\log n} T(1) + n.\log n$$

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

$n/2^k = 1$

$n = 2^k$

$\log n = k$

If a sorting algo takes more than $\log n$ extra space then it is out-place algo

vice versa

max of $\log n$ space then in-place

imp Ques

Total space = i/p + extra

n (array) + n (merge) + $\log n$ (stack)

$= O(n)$

Mergesort_In-place

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

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

decreasing g.p series summation = 1

$$= O(n^2)$$

($O(n\log n)$ = Best case = Worst case = Average case because in 'else' part of merge-sort algo there are no r/n, exit / break (we cannot minimize any stmt there whichever sequence of i/p came))

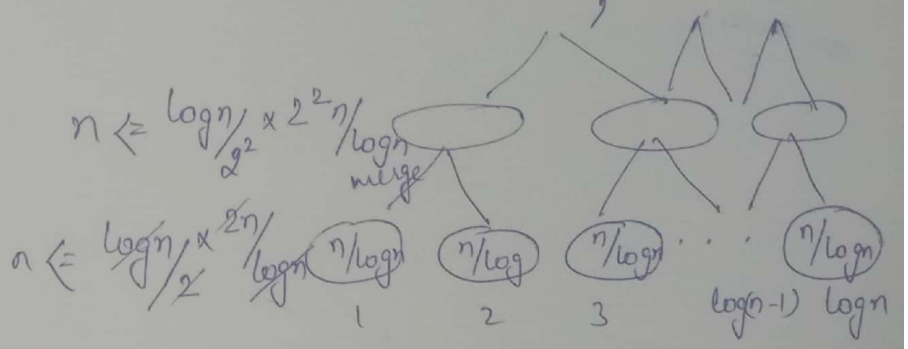
i/p : $\log n$ sorted subarrays each of size $n/\log n$

o/p : find one sorted subarray of size n

$\downarrow\downarrow$

Best algo worst case

$n = \log n/2^k * 2^k n/\log n$



$n \leq \log n/2^2 \times 2^2 n/\log n$ merge

$n \leq \log n/2 \times 2n/\log n$

$\frac{\log n}{2^k} = 1$

$\log n = 2^k$

$\log\log n = k$

$n * k \Rightarrow n \times \log\log n$

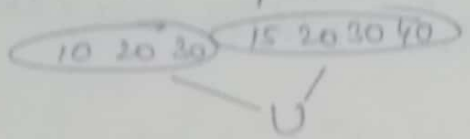
because grouping each time of base of log would be 2

if we group 3 at a time base of log would be 3.

we assume
Ques k subarrays | each subarray (size of subarray) n^2/k

Ans $n^2 . \log k$ (discuss how?)

Ques i/p: 2 sorted subarray A contain m elements and B contain n elements (apply merge algo)
o/p: Find A ∪ B & A ∩ B.
In union no repeatation



10, 15, 20, 30, 40

* Tips
* Any problem 2 sorted / 5 sorted / logn sorted subarray
any no. of sorted subarray ↓ more than 1 use merge sort

* "One sorted array" → B

```
swap(a[i],a[j])
}
}
swap(a[p],a[i])
return(i);
}
```

i is going to execute (n-1) times both in (best & worst case)

29 42 10 19

29 42 65 15 32 91 10 75 19 32
1 2 3 4 5 6 7 8 9 10

26 | 15 10 19 | 29 | 91 65 75 42 32 |
smaller greater

(26 15 10 19) 29 (91 65 75 42 32)
1 2 3 4 5 6 7 8 9 10
i

eg → (2)

55 26 99 42 23 79 56 21 99
1 2 3 4 5 6 7 8 9
i j i j

21 26
21 23 99 26 55 79 56 42 99
i

21 23 29 26 55 79 56 42 99

21 20 42 23 55 79 56 99 99

(n-1) times (both in best and worst case)

So time complexity of partition = $o(n)$

total no of swaps = max n (worst case)
min swap = 1 (best case)

Ex (3)

60 21 35 15 42 10 20 90 70 80

i j j j j j i i i

i i i i i j

60

20 60 21 35 15 42 10 20 90 70 80

(20 21 35 15 42 10) 60 (90 70 80)

smaller greater

Quicksort Algorithm

```
Quicksort(a, p, q)
{
  if(p==q)
    return(a[p]);
  else
  {
    m = partition(a, p, q)
    quicksort(a, p, m-1)
    quicksort(a, m+1, q)
    return a);
  }
}
```

P.R (let T(n) be the time complexity of quick sort algo of n element)

$$T(n) = \begin{cases} & \\ & \end{cases}$$

then R.Recurrence Relation

$$T(n) = \begin{cases} o(1) & \text{if } (p=q) \\ o(n) + T(n/2) + T(n/2) + 0 & \text{else} \end{cases}$$

↳ (when elements are dividing into 2 equal part)

In case of balanced partition

$$T(n) = 2T(n/2) + n$$ (only one case)

$$2\left[2T\left(n/2^2\right)\right] + n/2 + n$$

$$2^2 T\left(n/2^2\right) + \frac{2n}{2} + n$$

$$2^2\left[2T\left(n/2^3\right)\right] + n/2^2 + \frac{n}{2} + n$$

$$2^3 T\left(n/2^3\right) + n/2^2 + n/2 + n$$

⋮ k times

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

$n = 2^k$

$k = \log_2 n$

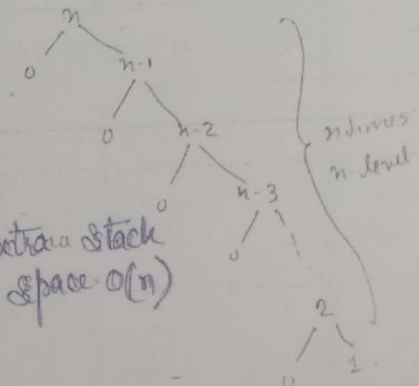
$= \log n\, T(1) + n\left[\frac{1}{2^0} + \frac{1}{2^1} + \cdots + \frac{1}{2^{k-1}} + \frac{1}{2^k}\right]$

$= \log_2(n) + n \log n$

generalised case

$T(n) \Rightarrow O(n) + T(m-p) + T(q-p) + 0$ if $(n>1)$

best case (balanced partition)	worst case (unbalanced partition)
$T(n) = n + 2T(n/2)$	$T(n) = n + T(0) + T(n-1)$
$= O(n \log_2 n)$	$n + T(n-1)$
no. of levels $= \log_2 n$	$= T(n-1) + n$
each costs $= O(n)$	$= O(n^2)$
extra space $(\log n)$	



if pivot element position keeps on changing then it is called randomized quicksort

Randomized Quicksort

quicksort → best → $O(n \log n)$
worst → $O(n^2)$

extra space can be reduced to from n to $\log n$ by writing better algo i.e. $O(\log n)$

Average case

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

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

$2T(n/2) + 2n \Rightarrow 2n \log n$

$= O(n \log n)$ {which is nothing but best case only}

[Avg case and best case are same]

in case unlucky case first then

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

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

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

$= 2n \log n$

$O(n \log n)$

Avg case of quicksort is same as that of best case of quicksort.

quicksort is applied on following 2 i/p

increasing order

i) 1,2,3,4,5 ..... n $O(n^2)$ swaps.

decreasing order

ii) n, n-1, n-2, n-3, ... 2,1 $O(n^2)$ swaps

Let C_1, C_2 be the two comparison made for the i/p 1 & 2 respectively then what will be the relation between C_1 & C_2

i) $C_1 < C_2$.

ii) $C_1 > C_2$.

iii) $C_1 = C_2$

iv) non-comparable.

(ii) if array is in increasing order then that is the worst case of quicksort

let say.

10, 20, 30, 40, 50, 60, 70

(1) 10 (20, 30, 40, 50, 60, 70) → 6-1

20 (30, 40, 50, 60, 70) → 5-1

→ 4-1

$T(7) = 6 + T(6)$

$6 + 5 + T(5)$

$4 + T(4)$

$3 + T(3)$

$2 + T(2)$

$1 + (T(1)$

$n + T(0)$

$= 1 + 2 + 3 + \dots + n$

$\frac{n(n+1)}{2}$

$O(n^2)$

Swaps $= n-1$,

(ii) for decreasing series

10 60 50 40 30 20 70

70 60 50 40 30 20 10

$\Rightarrow n$ swaps.

P(7) ⇒ 7 =

(10, 20, 30, 40, 50, 60) 70 (0)

6-1 ⟶ n swaps

(10) (20, 30, 40, 50, 60)

P

5-1 ⟶ 1 swaps

(20, 30, 40, 50) (60) →

4-1 ⟶ n swaps

(1) 20 (30, 40, 50)

3-1 = ⟶ 1 swaps

(30, 40) (50) (1)

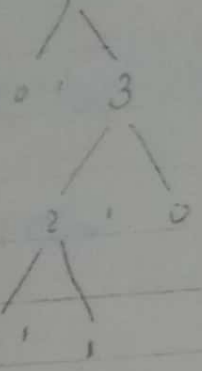
1

1 swaps

total no of swaps

$= n + 1 + n + 1 \dots + 1$

$= \frac{n}{2} * n \rightarrow O(n^2)$



So for $c_1 = n$
$c_2 = n^2$

So $c_2 > c_1$

If all elements are equal

~~10_f~~ 10_b 10_c 10_d 10_e ~~10_f~~ 10_a
~~10_a~~, ~~10_b~~, ~~10_c~~, ~~10_d~~, 10_e, ~~10_f~~
1 2 3 4 5 6

10_f 10_b 10_c 10_d 10_e 10_a. ( )

⇓

always right hand side is gap.

comparison = n^2
swaps = n^2

8- In case of sorted

if the array is

case

4- quick sort is bet

not

In the

quick sort

sort is

Recursive tree

$T(n) = n + T($

$\log_2 n$

$n/2$

$n/2^2$ $n/2^2$

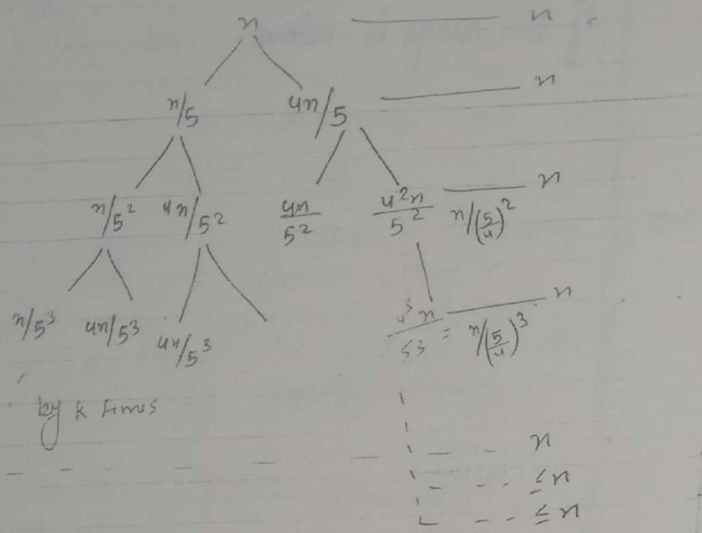
k time

II $T(n) = n + T(n/5) + T(4n/5)$

$n + n/5 + T(n/5^2) + \frac{4n}{5} + T(\ldots)$

$n + \frac{n}{5} + \frac{n}{5^2} + T\left(\frac{n}{5^3}\right) + \left(\frac{4}{5}\right)^2 n + T\left(\left(\frac{4}{5}\right)^3 n\right)$

$n + n + T\left(\frac{n}{5^3}\right) + T$



$n/5^k$

no. of levels $= \log_5 n$

$n/(5/4)^k$ ⇒ this will have more no. of levels

no. of levels $= \log_{5/4} n$

Bcz $\frac{n}{(5/4)^k} > \frac{n}{5^k}$

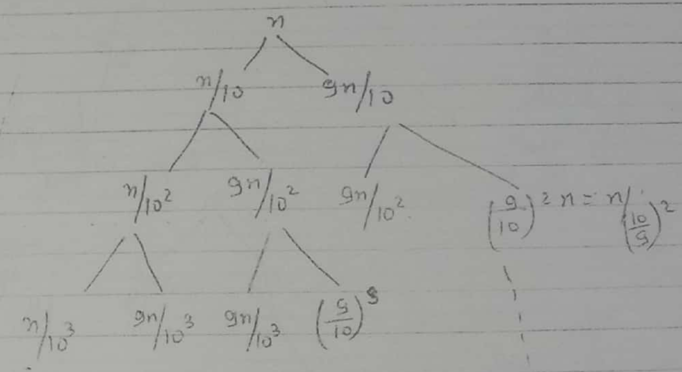
$n\log_5 n \le T(n) \le n\log_{5/4} n$

$T(n) = O(n\log_{5/4} n)$

$T(n) = \Omega(n\log_5 n)$

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

3⇒ $T(n) = n + T(n/10) + T(9n/10)$



$n/10^k$

no. of levels $= \log_{10} n$

$n/(10/9)^k$

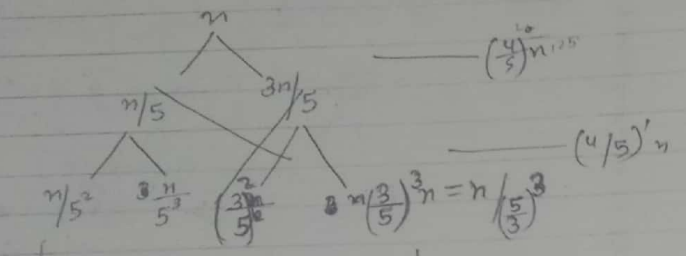
no. of levels $= \frac{n}{(10/9)^k} = 1$

$n = \left(\frac{10}{9}\right)^k$

$k = \log_{10/9} n$

IV

$T(n) = n + T(n/5) + T(3n/5) \Rightarrow O(n)$



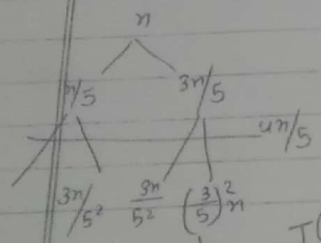
$(4/5)^0 n$

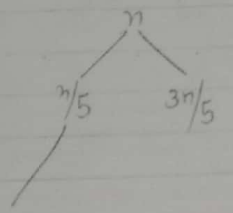
$(4/5)^1 n$

$(4/5)^{\log_5 n}$

$n/5^k$

no of levels
$k = \log_5 n$





$(4/5)^{\log_{5/3} n}$

$T(n) = n \left[(4/5)^0 + (4/5)^1 + \cdots (4/5)^{\log_{5/3} n} \right]$

$= n\, O(1) = O(n)$

$\frac{n}{(5/3)^k}$

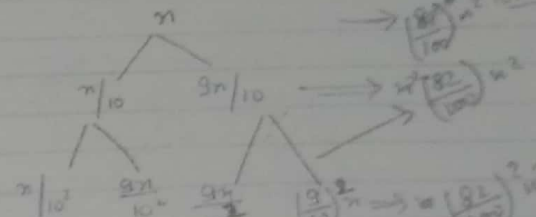
here $k = \log_{5/3} n$

??

V

$T(n) = n^2 + T(n/10) + T(9n/10)$

Q2.



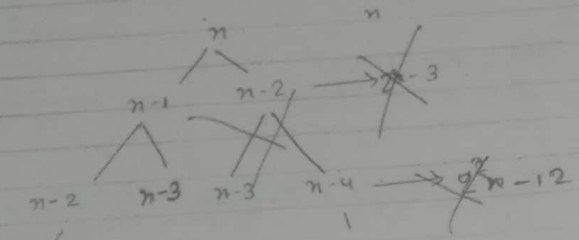
$n/10^k$

no of levels
$k = \log_{10} n$

$n/(10/9)^k$

no of levels
$k = \log_{10/9} n$

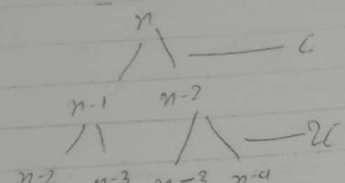
VI) $T(n) = C + T(n-1) + T(n-2)$



no. of levels $K = n$

$2^K n$ - (same constant)

no. of levels $K = \log n$



no. of levels $K = n$

no. of levels =

$T(n) = 2^0 C + 2^1 C + \cdots + 2^n \cdot C$

$C[2^0 + 2^1 + \cdots + 2^n)$

$= 2^0 \left\{ \frac{2^n - 1}{2-1} \right\} = O(2^n)$

$T(n) = T(n/2) + T(n/3) + T(n/5) + C \Rightarrow O(3^n)$

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

$T(n) = n + T(n/5) + T\left(\frac{4n}{5}\right)$

$T(n) = n + T(n/100) + T\left(\frac{99n}{100}\right)$

$T(n) = n + T(\alpha \cdot n) + T((1-\alpha) \cdot n)$] generalized form

$0 < \alpha < 1$ (for best & average case)

then stack on no. of levels

$= \log_{\frac{1}{\alpha}} n$

$= \log_2 n$

stack space

$\log_{(\frac{1}{1-\alpha})} n$

let suppose $\alpha = \frac{1}{5}$

then $\frac{1}{\alpha} = 5$

so $\log_5 n$

So stack size $= \max\left(\log_{\frac{1}{\alpha}} n, \log_{\frac{1}{1-\alpha}} n\right)$

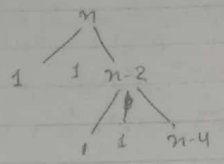
Time : $O(n \log n)$

base doesn't matter.

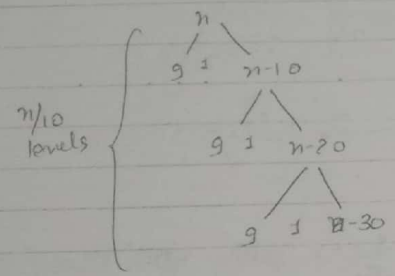
Worst_case

$T(n) = n + T(0) + T(n-1) \Rightarrow n * n \rightarrow O(n^2)$

$T(n) = n + T(1) + T(n-2) \Rightarrow \frac{n}{2}(\text{levels}) * n = O(n^2)$



$T(n) = n + T(9) + T(n-10) \Rightarrow \frac{n}{10} * n \Rightarrow O(n^2)$



$T(n) = n + T(c-1) + T(n-c) = \frac{n}{c} * n = O(n^2)$

constant stack space

n/c levels

n/c

O(n) — extra space

using better algo it can be reduced to $\log_2 n$

where c is a constant & & $c \geq 1$

Prob In quick sort the sorting of n numbers, the $n/5^{th}$ smallest element is selected as pivot using $O(n)$ time complexity Algorithm; then what will be the best case time complexity of quicksort

i) $O(n^2)$
ii) $O(n \log n)$
iii) $O(n^3)$
iv) $O(n)$

$T(n) = O(n)$ (for pivot) $+ O(n)$ (partition) $+ T\left(\frac{n}{5} - 1\right) + T\left(n - \frac{n}{5}\right)$

$= 2n + T(n/5) + T\left(\frac{4n}{5}\right)$

$= O(n \log n)$

Prob In quick sort the sorting of n numbers the $n/5^{th}$ smallest element is selected as pivot using $O(n^2)$ time complexity algo, then what will be the worst case time complexity

$\Rightarrow O(n^2)$

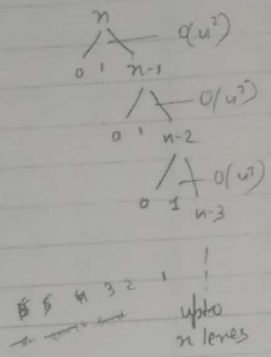
In quick sort the sorting of n numbers, the $n/10^{th}$ element is selected as pivot using $O(n^2)$ time complexity algo, then what will be the worst case time complexity of algo

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

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

$= \frac{n(n+1)(2n+1)}{6}$

$= O(n^3)$



In quick-sort the sorting of n no., the $n/5^{th}$ largest element is selected as pivot using $O(\log n)$ time complexity algo, Then what will be the the best case time complexity

$T(n) = O(\log n) + O(n) + T(n - n/5) + T(n/5)$

$\Rightarrow O(n \log n)$

Median = $\frac{n}{2}$ smallest element in sorted array

Randomized Quick-sort

ex 10 40 20 30 10 50 60 70

$i = RG_1(1,7)$

swap(1,4)

(20 30 10) 40 (70 60 50)

$i = RG_1(1,3)$ $\quad$ $i = RG_1(5,7)$

swap(1,1) $\quad$ swap(5,7)

10 (20) 30 $\quad$ (60,50) 70 ( )

(nothing but best, avg case)

Randomized Quick-sort

choosing pivot-element randomly is called randomized Quick-sort (in case of sorted array)

```
RQS (a, P, q)
{
  if (p == q)
    return (a[p]);
  else
  {
    r = RG1(P, q)
    swap(a[P], a[r])
  }
  m = Partition(a, P, q);
  RQS (a, P, m-1);
  RQS (a, m+1, q);
}
  return(a);
```

in randomized Quick-sort all three complexities are same i.e. $O(n \log n)$

note | for some cases randomized quick-sort will give worst-case performance of $O(n^2)$ these are

i) All elements are same
ii) if array is not sorted then also worst case performance will come.

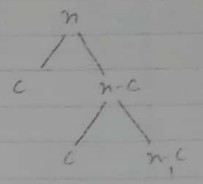
i) 7 ⇒ 1st smallest element
ii) 4 ⇒ 2nd smallest no.
iii) 1 ⇒ 7th.

60	20	50	30	10	90	80
1	2	3	4	5	6	7

suppose random no. generated = 5

at 5th partition no. = 10
so again left hand side no element & right hand side n-1
so this is also worst case.

```
QS(a,P,q)
{
  while(P<q)
  {
    m = partition(a,P,q)
    if(m-P > q-m)
    {
      QS(a,m+1,q);
      q=m-1
    }
    else
    {
      QS = (a,P,m-1)
      P=m+1
    }
  }
}
```



stack
extra space = max min
$O(\log n)$ $O(1)$

after partition call recursion on smaller size side & on larger side use while loop. while loop, do not take stack space, it will take only constant time

In array random access is possible

Selection procedure

i/p an array of n-elements and integer K
o/p return K^{th} smallest element

Algo(1)
if array is sorted and sorting ordering — 1
return(a[K]) $\Longrightarrow O(1)$

Algo(2)
i) if array not sorted
apply K passes selection sort $\Rightarrow O(Kn)$ } $O(Kn)$
ii) return (a[K]) $\Rightarrow O(1)$

diff b/w selection & bubble sort
⇓ ⇓
for min element for max element (at the end of the array)

after 1st pass of bubble sort, it gives at 1st largest element at the end of array while for min element apply selection procedure.

With divide & conquer

	1	2	3	4	5	6	7	8	9	10
A[	25	92	15	10	65	88	21	19	14	40]

~~K=6~~ if K=3.
apply partition algo

m = (14 15 10 21 19) 25 (92 65 88 40)
1 2 3 4 5 6 7 8 9 10

⇓
(2) m = (10) 14 (15, 21, 19)
⇓
(10) 15 (21,19)
3 4 5
return [a[K]]

```
{
  if (p==q)
    return
  else
  {
    m = partition (a, p, q)
```

→ to find kth smallest element.

```
Selection (a, p, q, k)
{
  if (p == q)
    return a[p]
  else
  {
    m = partition(a, p, q)        ⟶ O(n)
    if (m == k)
      return a[k]                 ⟶ O(1)
    else if (k < m)               ⟶ O(1)
      to selection(a, p, m-1, k)  ⟶ O(m-p)
    else
      selection(a, m+1, q, k)     ⟶ O(q-m)
  }
}
```

Let $T(n)$ be the amount of time required for the selection algorithm on n elements. Then recurrance relation.

$$T(n) = \begin{cases} O(1) & \text{if } (p == q) \\ O(n) & \text{if } (m == k) \\ O(n) + T(n/2) & \text{else} \end{cases}$$

$$T(n) = \begin{cases} \Theta(n) & \text{if } (n=1) \\ O(n) + O(1) + O(1) + \Theta(m-p) & \text{if } n > 1. \end{cases}$$

$O(n)$ or $O(q-m)$

best case

$\Theta(n) + T(n/2)$

$\Rightarrow O(n)$

worst case

$\Rightarrow O(n) + T(n-1)$

$\Rightarrow O(n^2)$

3 ways to get kth smallest element:

⇒ partition
⇒ selection procedure ← (best soln)
⇒ first sort then return kth smallest

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

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

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

$T\left(\frac{n}{2^k}\right) + n\left[\frac{1}{2^0} + \frac{1}{2^1} + \cdots + \frac{1}{2^{\log_2 n}}\right]$

$\Rightarrow O(n)$

This is also avg case

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

$= n + n-1 + n-2 + \cdots + 1$

$= \Theta(n^2)$

Stack space = $O(n)$ ⇓ n-levels.

Stack space = $O(\log n)$ ⇓ $\log n$ extra space

Stack space = $O(n)$

Strassen's matrix multiplication

without DAC

matrix-addition ⇓

$A_{m \times n}$, B_{mn}

$C_{m \times n} = A + B$

C contain mn elements ⇓ ⇓

$mn * 1$ ⟶ addition ⇓

$mn * O(1)$ ⇓

$O(mn) \Rightarrow O(n^2) \Rightarrow$ if $(m=n)$

- Space = O(mn)
Extra space = O(mn) (for C matrix)

matrix multiplication (without divide & conquer)

Condition

$A_{m \times n}$, $B_{n \times p}$

$C_{m \times p} = A * B$

C contains mp elements

So no. of multiplications = $mp * n$

$= mnp * O(1)$

$= mnp = O(n^3)$ if $(m = n = p)$

Algo for multiplication

```
for (i=1 to n)
  for (j=1 to n)
    for (k=1 to n)
{
  c[i][j] = c[i][j] + a[i][j] * b[k][j];
}
```

using DAC

matrix multiplication

With divide & conquer, matrix multiplication

1- Matrices sizes are $\leq 2 * 2$
Small problem

2- Matrices sizes should be of power of 2

3- Matrices should be square matrices

ex

$$A = \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right] \quad B = \left[\begin{array}{cc|cc} a & b & c & d \\ e & f & g & h \\ \hline i & j & k & l \\ m & n & o & p \end{array}\right]$$

A_{11} A_{12} A_{21} A_{22} B_{11} B_{12} B_{21} B_{22}

$\frac{4 \times 4}{2} = 2 \times 2$ $\frac{4 \times 4}{2} = 2 \times 2$

$$C = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]$$

$C_{11} = A_{11} * B_{11} + A_{12} B_{21}$

$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$

$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$

$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$

1 → matrix 4×4

⇓

8 → matrix 2×2

+

4 → additions 2×2

1 → MM → 64×64

⇓

8 → MM → 32×32

⇓

4 → MM → 16×16

⇒ 1 → MM → $n \times n$

⇓

8 → MM → $\frac{n}{2} \times \frac{n}{2}$

+

4 → additions → $\frac{n}{2} \times \frac{n}{2}$

Let $T(n)$ be the amount of time required to multiply 2 matrices of size $n \times n$

Then recurrence relation

$$T(4) = 8T(4/2) + 4 * (4/2)^2$$
$$T(64) = 8(64/2) + 4 * \left(\frac{64}{2}\right)^2$$

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

$$T(n) = \begin{cases} O(1) & \text{if } n \le 2 \times 2 \\ 8T(n/2) + 4 * (n/2)^2 & \text{if } n > 2 \times 2 \end{cases}$$

⇓

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

$$= 8\left[8T(n/2^2) + \left(\frac{n}{2^2}\right)^2\right] + n^2$$

$$= 8^2 T(n/2^2) + \frac{n^2}{2} + n^2$$

$$= 8^2\left[8T(n/2^3) + \left(\frac{n}{2^2}\right)^2\right] + \frac{2n^2}{2} + n^2 \Rightarrow 8^3 T(n/2^3) + n^2 + 2n^2 + \dots$$

$$= 8^3\left[T(n/2^3)\right] + n^2 + \frac{n^2}{2} + n^2 - \dots$$

k times

$$8^k T(n/2^k) + 2^2 n^2 + 2n^2 + 2^0 n^2$$

$$\frac{n}{2^k} = 2$$

$$n = 2^k \cdot n = 2^{k+1}$$

$$8^{\log_2 k} \quad \log_2 n = k+1$$

$$k = \log_2 n - 1$$

$$\frac{n^3}{8} + n^2\left[1 + \frac{2^{\log_2 n}}{2-1}\right]$$

$$= \frac{n^3}{8} + n^2 \log n$$

$$= O(n^3)$$

google → DAC problem

⇒ with & without DAC, matrix multiplication is order of $O(n^3)$, will takes same time of $O(n^3)$, so with DAC is ~~best~~ worst (or) need extra space), no need of extra stack space is req.

According to strassen's

$$T(n) = \begin{cases} O(1) & \text{if } n \text{ is } 2 \text{ or } 2 \\ 7T(n/2) + 18(n/2)^2 & \text{if } n > 2 \end{cases}$$

$$= 7T(n/2) + 4.5n^2$$

$$= 7T(n/2) + n^2$$

$$= O(\log^{\frac{7}{2}})$$

extra space is $O(\log n)$ $= O(n^{2.8})$

space $= O(\log^{32.29})$

Problem I

i/p An array of n-elements in which until some place-n elements are in increasing order and afterwards decreasing order.

o/p :- find n.

LS
BS

10 20 30 40 50 (60) 55 45 35 25

$O(\log n)$

② i/p An array of n-elements
o/p find no. of inversions.
without - DAC — n^2
with DAC — $n \log n$.

20	10	
1	2	

Position $1 < 2$.
but value $a[1] > a[2]$.
So this is inversions

$\log n$ → generally means Binary Search

③ i/p an array of n-elements in (x,y) plane

o/p :→ find closest plane ~~without~~ DAC

without DAC ⇒ $O(n^2)$

with DAC ⇒ $O(n \log n)$

④ i/p : 2 sorted arrays of size each of n.

o/p : find n^{th} smallest element in union array.

1) with merge algorithm ⇒ $O(2n) \Rightarrow O(n)$

without merge ⇒ $O(\log n)$ [binary search]

Masters theorem

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

$a \geq 1$, $b > 1$, $f(n)$ is a positive function.

① if $f(n) = O(n^{\log_b a - \epsilon})$ where $\epsilon > 0$ and ϵ is constant

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

if $f(n) = \Omega(n^{\log_b a + \epsilon})$ where $\epsilon > 0$ and ϵ is constant

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

③ if $f(n) = \Theta(n^{\log_b a})$

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

~~co~~ add $\log n$ coz of recursive problem.

1st problem

$$T(n) = 8\,T(n/2) + n^2$$

$a = 8$

$b = 2$

$n^{\log_b a} = n^3$ | $f(n) = n^2$

case (i)

$O(n^3)$

$f(n) = n^{\log_b a - \epsilon} = \frac{n^3}{n}$

$f(n) = n^{3-1}$

where $\epsilon = 1$ & constant

so ~~left~~ right hand side is greater by i.e. polynomial times greater

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

$a = 2 \quad b = 2$

$n^{\log_b a} = n^1 \mid f(n) = n^2$

case II

$\Theta(n^2)$

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

where $f(n) = n^2$

$n^{\log_b a} = n$

so ~~f(n) is~~ $f(n)$ is greater than $T(n)$ by polynomial n

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

$a = 2$

$b = 2$

$n^{\log_b a} = n \mid f(n) = n$

case (3)

$\Theta(n \log n)$

ex IV $T(n) = 64T(n/8) + n^3$

$a = 64 \quad b = 8$

$n^{\log_8 64} = n^2 \mid f(n) = n^3$

case (III)

$T(n) = \Theta(n^3)$

ex 3 $T(n) = 2T(n/2) + n \log n$

$a = 2 \quad b = 2$

$n^{\log_n a} = n \mid f(n) = n \log n$

Note master thm dont work bcoz $n^{\log_b a}$ is logarithmic times smaller than $f(n)$ but not polynomical times.

note 1 ⇒

1⇒ in case (I) $n^{\log_b a}$ is polynomical times greater than $f(n)$ (n^ϵ where $\epsilon > 1$

2⇒ in case (2) $n^{\log_b a}$ is polynomical times smaller than $f(n)$

3⇒ in case (3) $n^{\log_b a}$ and $f(n)$ are asymptotically equal.

special method

of the recurrence relation contain root operation

1) a) $T(n) = T(\sqrt{n}) + c$

⇓

i) assume $n = 2^k$

$T(2^k) = T(2^{k/2}) + c$

ii) Assume $T(2^k) = S(k)$

$T(2^k) = S(k)$

$S(k) = S(k/2) + c$

now it is in the form of master theorem

here $a = 1$ $b = 2$ $f(k) = c$

$k^{\log_b a} = k^0 = 1$ | $f(k) = c = 1$

$S(k) = \theta(\log k)$

iii) $T(2^k) = \theta(\log k)$

(iv) $T(2^{\log_2 n}) = \theta(\log \log n)$

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

2) $T(n) = 2T(\sqrt{n}) + \log n$

$n = 2^k$

$T(2^k) = 2T(2^{k/2}) + \log(2^k)$

assume $2^k = S(k)$

$T(2^k) = S(k)$

$S(k) = 2S(k/2) + \log(2^k)$ in k

$a = 2$ $b = 2$ $f(k) = \log 2^k$

$f(k) = k$

$k^{\log_2 2} = k$ | $f(k) = \log_2 2^k = k$

$S(k) = \theta(k \log k)$

log n is not polynomial while n log n is.

(iii) $T(2^k) = \theta(k \log k)$

iv) $T(2^{\log_2 n}) = \theta(\log n \log \log n)$

Q2. Consider the following C programme

```
A(n)
{
  if (n ≤ 1)
  return 1
  else
  return ( A(n/2) + A(n/2) + n)
}
```

↳ Here it is neat algorithm but a big problem.

Let T(n) = time complexity, then

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

$= 2\left[2T(n/2^2) + n/2\right] + n$

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

$\vdots$ K times

$2^K T(n/2^K) + Kn$

$= n = 2^K$

$K = \log n$

$= nT(1) + n\log n$

$= O(n \log n)$

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

$2T(n/2) + C$

i) $O(n \log n)$ ✓
ii) $O(n)$
iii) $= O(n^2)$
iv) $O(\log n)$

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

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

$= O(n)$

b = A(n/2) ⟹ T(n/2)

c = A(n/2) ⟹ T(n/2)

d = b + c ⟹ O(1)

e = d + n ⟹ O(1)

return(e) ⟹ O(1)

}

return T(n/2) + T(n/2) + n.

Consider the following

```
A(n)
{
  if ( n ≤ 1) return(n)
  else
  return ( 5 A(n/2) + 3 A(n/2) + n)
}
```

$T(n) = 5T(n/2) + 3T(n/2) + n$

$8T(n/2) + n \quad 2T(n/2) + n$

$= n^3 + n$

$= n^{\log_2 8}$ by master theorem

$n^{\log_2 2} = n$

$= O(n)$

b = A(n/2) → T(n/2)

c = 5 * A → O(1)
d = A(n/2) → T(n/2)
e = 3 * d → O(1)
f = c+e → O(1)
g = f+n → O(1)

⇒ 2T(n/2) + C

$$T(n) = \begin{cases} O(1) & \text{if } n \le 1 \end{cases}$$

Consider

n . n . n - - - n = $O(n^n)$ → value

s = 1
for (i=1 to n) ⇒ O(n) → time complexity
s = s * n

A(n)
{
if (n ≤ 1)
return (LS(n)) → it will not give O(1) but O(n) coz of function calling.

⇒ n+n+n+n+n+n = O(1) → time complexity.
O(n) → value.

⇒ 1+2+3+ - - - - - +n = $\frac{n(n+1)}{2}$
= $O(n^2)$ → value.

{ s=0
for (i=1 to n) = O(n) → time complexity.
s = s+i
}

Consider the following C-program.

A(n)
{
if(n ≤ 1) return 1
else
return (n * A(n-1)).
}

T(n) = A(n-1) time complexity

n(n-1)

$T(n)$ = for time complexity

$T(n) = O(n)$

$T(n)$ for no. of multiplication.

$T(n) = O(n)$

~~no of v~~ $T(n)$ = value

$T(n) = n^2$

$T(n)$ for time complexity

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

$T(n-1) + C$

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

$T(n-3)+C+C+C$

{ } k time

$T(n) + kC$

$= O(n)$.

$T(n)$ = no. of multiplication

$$T(n) = \begin{cases} 0 & \text{if } n \le 1 \\ T(n-1)+1 & \text{else} \end{cases}$$

$O(n)$

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

```
else
{
b = A(n-1) => T(n-1)
c = n * b => 1
return (c)
}
```

as it is an array, then it is value.

$T(n-3) + 1 + 1 + 1$

upto $n-1$ times

so no. of multiplication = $n-1$.

$T(n)$ = value

$$T(n) \begin{cases} 1 & \text{if } n = 1 \\ n * T(n-1) & \text{else } (n > 1) \end{cases}$$

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

$T(n-3) * n-2 * n-1 * n$

$n-1$ times

$1 \cdot 2 \cdot 3 \cdots (n-2)(n-1) \cdot n$

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

Bubble Sort

s/o	A[	1	2	3	4	5	6	7	8]
		85	15	26	75	19	40	20	99

Pass ① 15 26 75 19 85 40 26 85) 99 → 7

largest element at the last

Pass ② 15 26 19 75 40 26 75 75) 85 99 → 6 - c

second largest

second last position

2 - quest from DAC
2 - quest from greedy
total = 4

Pass ③ 15 19 26 36 40 75 85 99
15 26 19 40 36
5 - comparisons
5 swaps / 0 - best

④ 15 19 26 36 40 75 85 99
4 - comparisons
4 - swaps / 0 - best

15 19 20 36 40 75 85 99
3 comparisons

and so on

final bubble sort
15 19 26 36 40 75 85 99

15 19
15 19

total = (n-1) passes

ok
① Bubble sort requires n-1 passes

total comparisons = ~~1+2+3+...(n-1)~~ + n-2+ n-3 + ... +2+1
= ~~n(n+1)~~ $\frac{n(n-1)}{2} = O(n^2)$ [BC, WC, AC]

total swaps
BC → 0
WC → 1+2+3+.... +n-1 = $\frac{n(n-1)}{2} = O(n^2)$

Avg case
$= \frac{n}{2} * \left(\frac{n}{2} - 1\right) = O\left(\frac{n^2}{4}\right) = O(n^2)$

④ total time complexity = total comparisons + no. of swaps
$= O(n^2)$ $n^2 + n^2$ [BC, WC, AC]

⑤ total comparisons will be greater or equal to no. of swaps

⑥ in place, stable

Note ||

if in two passes of which are consecutive of bubble sort, that means n array elements are sorted, no need to go for more passes.

In bubble sort at any place 2 consecutives passes outputs are same then stop the bubble sort bcoz array is already sorted

so

BC	WC, AV	for bubble sort
⇓	⇓	
O(n)	O(n²)	

I/P 80 25 45 66 77 80
Pass ① 25 45 66 77 80
Pass ② " " " " "

Selection Sort (will give min element after each pass)

i/p A[	85 (15)	45	26	70	19	50	80	15 (89)]
	1	2	3	4	5	6	7	8

Pass ①

$i = 1$

min = ~~1~~ ~~2~~ ~~3~~ ~~5~~ 8

So minimum is at 8th position

so swap(a[i], a[min])

15	45 (19)	26	70	19 (45)	50	80	89
	4	5	6	7	8		

here comparisons = n-1

swap = 1 (in every case)

in selection, each pass contain only one swap at most one swap

Pass ② $i = 2$

min = ~~2~~ ~~3~~ 5 (n-2) comparisons

15	19	26	~~45~~ 70	45 (70)	50	80	89
1	2	3	4	5	6	7	8

Pass ③ $i = 3$ (n-3) comparisons

min = 3

Pass ④ $i = 4$ (n-4) comparisons

min = 5

15	19	26	45	~~70~~ 50	70	80	89

Pass ⑤ 15 19 26 45 50 70 80 89

note (1) it requires (n-1) passes and 1 swaps at each passes at most one

(2) total comparisons = $1+2+3+\cdots+n-1$

$= \frac{n(n-1)}{2} = O(n^2)$ [NC, BC, WC]

total no. of swaps = $n-1 = O(n)$

Selection sort is better in terms of swaps operation on the worst case than that of bubble sort

~~Time Complexity~~

Total comparisons = n^2

total swaps = $n-1$

then time complexity = $n^2 + n - 1$

$= O(n^2)$

inserting sort

i/p : A[	10	20	30	25	40	50	15	70	5]
	1	2	3	4	5	6	7	8	9

Pass ① [10] → 0 comparisons

Pass ② [10 | 20] → 1 comp

Pass ① [50 | 40 | 30]

Pass ② [40 | 50 | 30] 1-c

[40 | 50 | 25 | 30]

Pass ③ [40 | 50 | 25 | 30 | 40]

Pass ④ [40 | 50 | 25 | 30 | 40 | 60]

[40 | 50 | 25 | 30 | 40 | 60 | 15]

Pass ⑤ [10 | 15 | 20 | 30 | 40 | 50]

Pass ⑥ 10, 15, 20, 25, 30, 40, 50, 15

[10 | 15 | 15 | 20 | 30 | 40]

Pass ⑦ 10, 15, 20, 25, 30, 40, 50, 70

Note:
1) It is inplace, stable

best

Best Case

			c	s
o/p: 10, 20, 30, 40, 50				
Pass ①	[10]	0 comb	0	
Pass ②	[10 | 20]	one comb	1	0
Pass ③	[10 | 20 | 30]	1-cmb	1	0
Pass ④	[10 | 20 | 30 | 40]	1 cmb	1	0
Pass ⑤	[10 | 20 | 30 | 40 | 50]	1-cmb		

So total comparisons = n-1
Swap = 0(for) 0
Time Complexity = d·c + f·s
= O(n)

1) of all the sorting algo, insertion sort has the best case i.e. O(n) (when array is already sorted)

when array is sorted quick sort gives worst case.

1) Insertion sort algo, best case will take n-1 comparisons and 0 swaps, so total time complexity is O(n)

2) In the given array most of the elements are sorted, then insertion sort will give best case, but quick-sort will give worst case

Worst Case - (when array is sorted in reverse order)

50 40 30 20 10

Pass ①	[50]		
Pass ②	[50 | 40]	1-cmp	1 swap
Pass ③	[40 | 50 | 30]	2-cmp	2 swap
	[30 | 40 | 50]		

Pass ③ | 10 | 20 | 30 | — 1-c

Pass ④ | 10 | 20 | 30 | 25 — 1-c

| 10 | 20 | 25 | 30 |

Pass ⑤ | 10 | 20 | 25 | 30 | 40 |

Pass ⑥ | 10 | 20 | 25 | 30 | 40 | 50 |

Pass ⑦ | 10 | 20 | 25 | 30 | 40 | 50 | 15 |

| 10 | 15 | 20 | 30 | 40 | 50 |

Pass ⑧ 10, 15, 20, 30, 40, 50, 15

| 10 | 15 | 15 | 20 | 30 | 40 | 50 |

Pass ⑨ 10, 15, 10, 25, 30, 40, 50, 70, 5

note:-
⇒ It is inplace, stable

~~best~~

Best Case

		c	s
o/p: 10, 20, 30, 40, 50		0	
Pass ①	| 10 | — 0 comb	1	0
Pass ②	| 10 | 20 | — one comb	1	
Pass ③	| 10 | 20 | 30 | — 1-cmb	1	0
Pass ④	| 10 | 20 | 30 | 40 — 1 cmb	1	0

Pass ⑤ | 10 | 20 | 30 | 40 | 50 — 1-cmb

So total comparison = n-1
swap = ~~O(n)~~ 0
time complexity = c-c + f-s
= O(n)

of all the sorting algo, insertion sort has the best case i.e. O(n) (when array is already sorted)

when array is sorted quick-sort gives worst case.

1) Insertion sort algo, best case will take n-1 comparisons and 0 swaps, so total time complexity is O(n)

2) In the given array most of the elements are sorted, then insertion sort will give best case, but quick-sort will give worst case.

Worst Case - (when array is sorted in reverse order)

50 40 30 20 10

Pass ① | 50 |

Pass ② | 50 | 40 — 1-cmb | 1 swap

Pass ③ | 40 | 50 | 30 — 2-cmb | 2 swap

| 30 | 40 | 50 |

Pass ④ | 30 | 40 | 50 | 20 | 3-C | 3 swaps

| 20 | 30 | 40 | 50 |

Pass ⑤ | 20 | 30 | 40 | 50 | 10 | 4-C | 4 swaps

| 10 | 20 | 30 | 40 | 50 |

total comparison = 1+2+---+n-1
= n(n-1) = O(n²)

total swaps = 1+2+---+n-1
= n(n-1) = O(n²)

total time complexity = O(n²)

Question: while applying insertion sort, to find a correct place of a particular element we apply binary search, what will be the time complexity of insertion sort algo, if we replace linear search by binary search for n elements

worst case

| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 25 |

total inversion = 5

30, 25
40, 25
50, 25
60, 25
70, 25

in B.S. no. of comp decreases, but no. of swaps increases.

B.S.
for 1 element = log n — comp
n — swaps

n elements = n(n+log n)
= n² + n log n
= O(n²)

Note:
The no. of inversions in the given array is equal to the no of right shifts in insertion sort

2- if the no inversions are less than n, then the array is almost sorted

in the given array, if at all max n inversions are there, in the insertion sort, then the insertion sort will give best case algo

eg
10 20 30 40 50 60 70 5

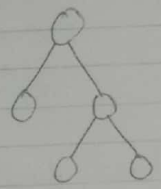
Average Case:
half best + half worst

(0 1 1 1 1 1) + (n n n n n)
n/2 * 1 n/2 * n

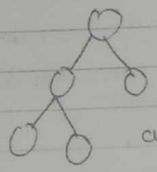
= n/2 + n²/2 = O(n²)

Heap-Sort

Almost complete binary Tree

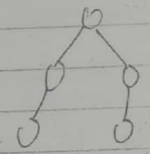


BT ✓
Almost BT ✓



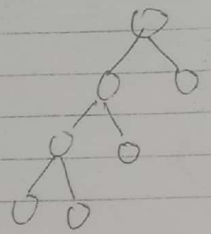
almost complete binary tree

BT ✓
Almost Complete BT ✓



BT ✓
ACBT ✗

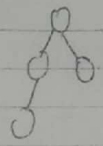
first complete left side



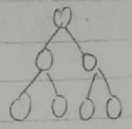
BT ✓
ACBT ✗

previous level gaps are not allowed,

greedy techniques completely uses heaps.



BT ✓
ACBT ✓

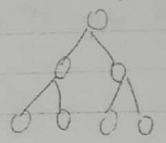


BT ✓
ACBT ✓

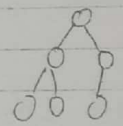
A Binary Tree is said to be a almost complete binary tree if and only if

i) without going left don't go to right (in case of every node)

ii) at every node, without completing the current level, don't go to the next level.



BT ✓
ACBT ✓
CBT ✓

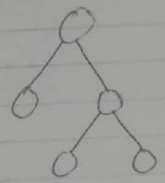


BT ✓
ACBT ✓
CBT ✗

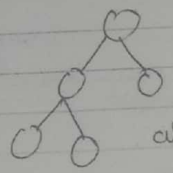
A maximal almost Complete BT is called Complete BT.

Heap-Sort

Almost complete binary Tree

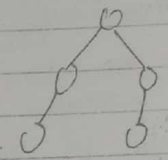


BT ✓
Almost BT ✓



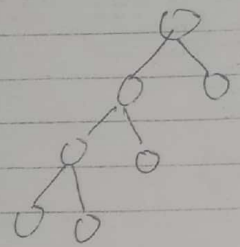
almost complete binary tree

BT ✓
Almost Complete BT ✓



BT ✓
ACBT ✗

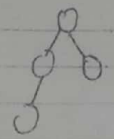
first complete left side



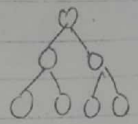
BT ✓
ACBT ✗

previous level gaps are not allowed,

greedy techniques completely uses heap.



BT ✓
ACBT ✓

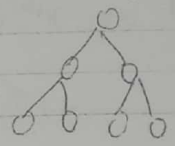


BT ✓
ACBT ✓

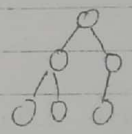
A Binary Tree is said to be almost complete binary tree if and only if

i) without going left don't go to right (in case every node)

ii) at every node, without completing the current level, don't go to the next level.

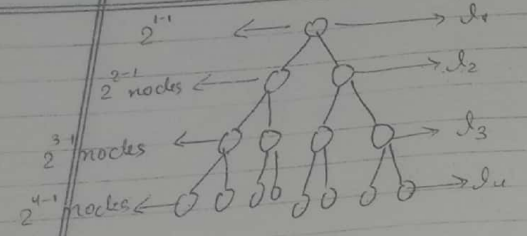


BT ✓
ACBT ✓
CBT ✓



BT ✓
ACBT ✓
CBT ✗

A maximal almost complete BT is called complete BT.



① in ACBT.

$15 = \lceil \frac{15}{2} \rceil = 8$ external nodes

$15 \Rightarrow \lfloor \frac{15}{2} \rfloor = 7$ internal nodes

① If the ACBT contain n nodes

no. of leaf node $= \lceil \frac{n}{2} \rceil$

no. of internal node $= \lfloor \frac{n}{2} \rfloor$

② The maximum no. of nodes present at K^{th} level of almost complete binary tree $= 2^{K-1}$

③ maximum nodes present in K^{th} level of ACBT $= n$.

$n = 2^K - 1$.

$2^K = n+1$

$K = \log_2(n+1)$

maximum no. of levels in BT with 15 nodes

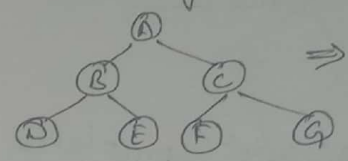
$= 15$

Balanced trees have less no. of levels than unbalanced B.T so it balanced trees will same time and will have less space required. i.e. in log.

Heap Sort

Q How binary tree will store in comp?

ex



⇒

1	2	3	4	5	6	7	
A	B	C	D	E	F	G	

N L R

If a node is stored in i^{th} place of the array then -

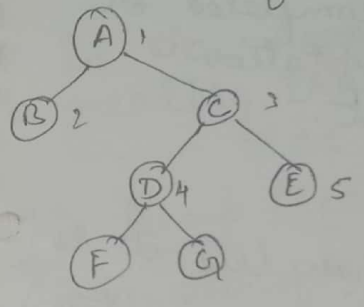
(i) parent of $i = \lfloor i/2 \rfloor$

(ii) left child of $i = i*2$

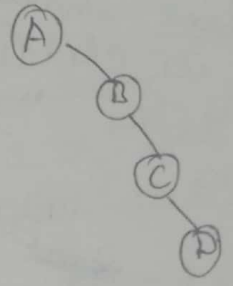
(iii) Right child of $i = (2i)+1$

$= (\text{left child of } i) + 1$

Q Store the following B.T in the form of array



1	2	3	4	5	6	7	8	9
A	B	C			D	E	F	G

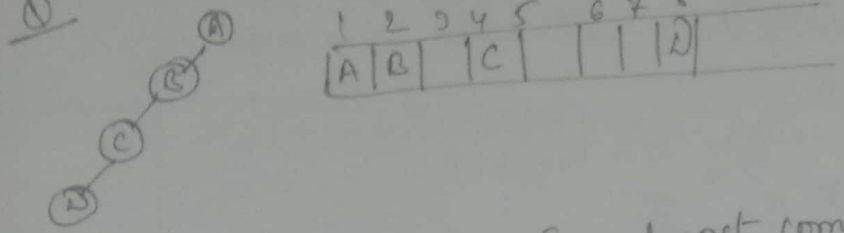


1	2	3	4	5	6	7	8	9	10	11	12
A	-	B	-	-	-	C	-	-	-	-	-

no. of nodes $= n$

no. of places $= 2^n - 1$

①



Ques Note:

n (nodes)

- Min size array → n-length array. ∴
- max. size array required to store n-nodes tree → $2^n - 1$ length array.

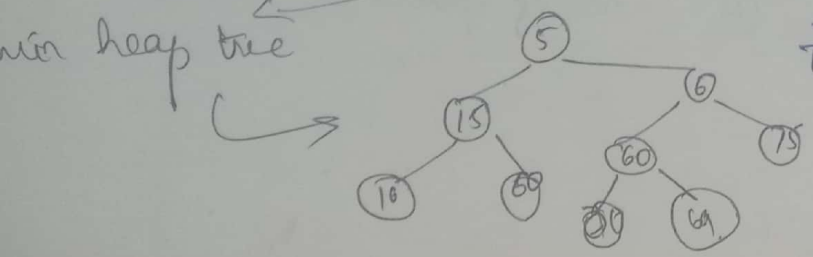
For almost completed binary tree → binary tree contains n-length array.

Note

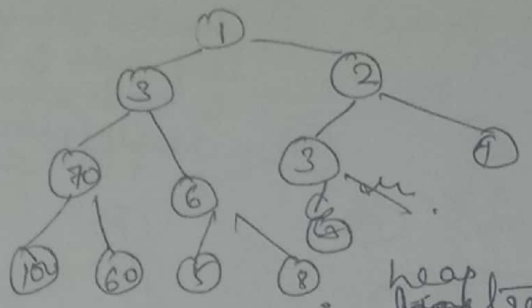
If the given binary tree is almost completed or completed then array representation is advisable otherwise link-list.

Heap Tree

In almost completed binary tree it is a min heap if root node is minimum comparing its children.



Time to find max = O(1)
find min = O(n)



In min heap tree → min heap tree root is small of array.

In almost completed binary tree in which at every tree root is min among most other children. or almost equal.

* In Max heap tree the uppermost root node is greater than all the other. or equal / not equal.

Min heap tree.

(min element can be found = O(1)
max " " " " = O(n)

Purpose of Data structure create and destroy nodes according to the require mem

Time to find min element if:
- sorted in ascending = O(1)
- sorted in descending = O(n)

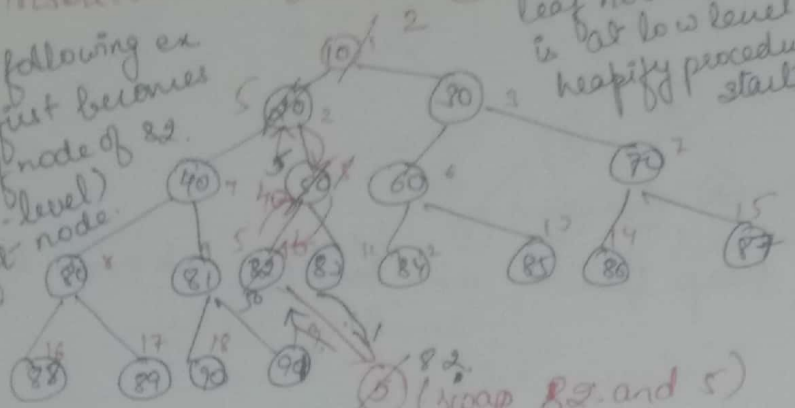
Time to find max element is O(n/2) i.e. O(n) min heap (n/2) becoz last level element would

Min-Heap

Insertion → Insert (5)

Inserted element becomes leaf node to the node which is at low level and then heapify procedure starts

In following ex 5 first becomes leaf node of 82. (4-level) left node.



(swap 82 and 5)

$\log_2 n + 1$

$\log_2 2^4$

Best Case → O(1)

worst case → log(n)

Average Case = $1 + 2 + 3 + \dots + \log n$

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

$= O(\log n)$

Note: Inserting 'n' element into min heap which already contain 'n' elements.

1) O(1) = best case
2) O(log n) = worst and average case

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
10	20	30	40	50	60	70	80	81	82	83	84	85	86	87	88	89

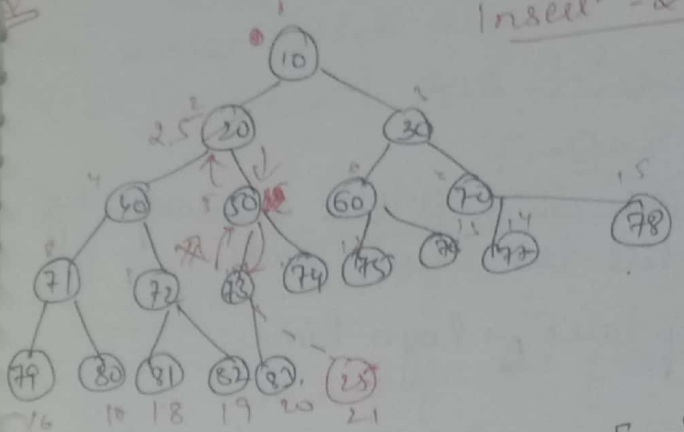
Parent of 5 = 20/2 = 10 (swap)

" " 10 = 5 = 5/2 = 2 (swap)

18	19	20
90	91	5

Parent of 2 = 1 = 5

Insert -25

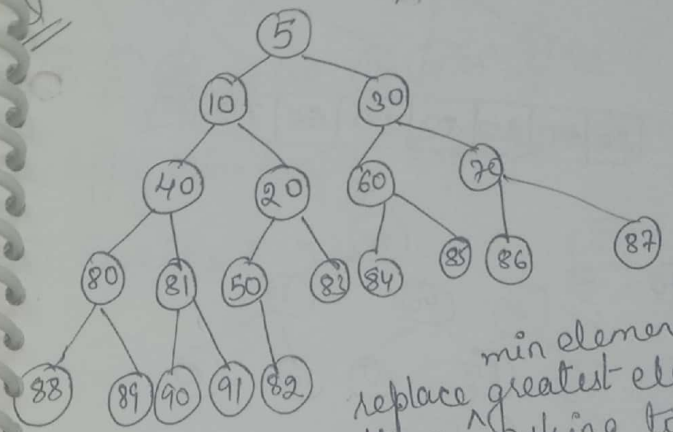


Heapify Top in case of insert

$21/2 = 10.5 = 10$ swap a[21] & a[10]

$10/2 = 5 \Rightarrow$ swap(10, 5)

$5/2 = 2.5 \Rightarrow$ swap(5, 2)



log n levels

Delete any one element from min-heap means delete min element from heap

replace min element by greatest element then min heapify bottom ↑ asking to lower levels.

5	10	30	40	20	60	70	80	81	50	83	84	85	86	87	88	89	90	91

$x = a[i]$ Total elements = n

$a[i] = a[n]$

now total elements = $n-1$

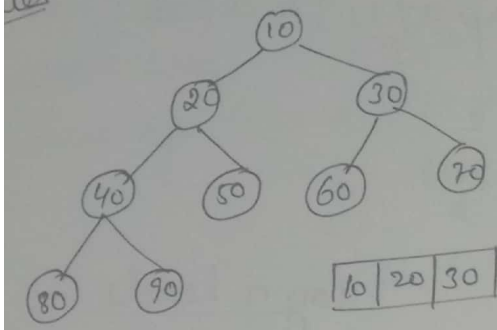
check if $a[i] > a[2*i+1]$

check from childs which is less and swap by less elements till the level ends.

min-heapify top take' → $\log n$ time

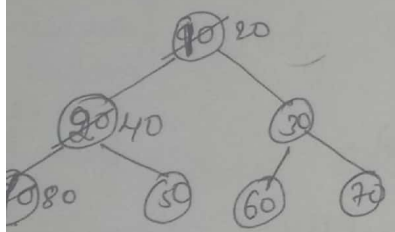
ues

Delete one element

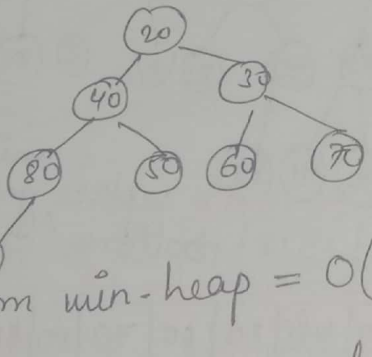


10	20	30	40	50	60	70	80	90

ultant tree



20	40	30	80	50	60	70	90



find smallest element from min-heap = $O(1)$

delete " " " " " = $\log n$.

To Note:

* To delete a element from min-heap or max heap which already contains elements $O(1)$ [best case] and $O(\log n)$ (worst case & average case)

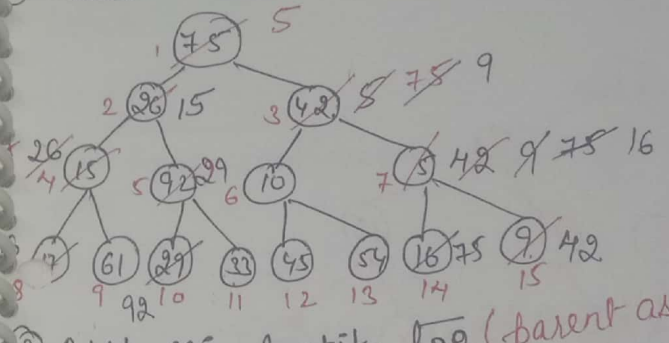
* In min-heap finding max element = $O(n)$ (in min heap max element is found in last level)

* BUILD-HEAP [creating a min heap tree ~~using~~ with $O(n)$ time]

ex: create min-heap for the following n-element array of

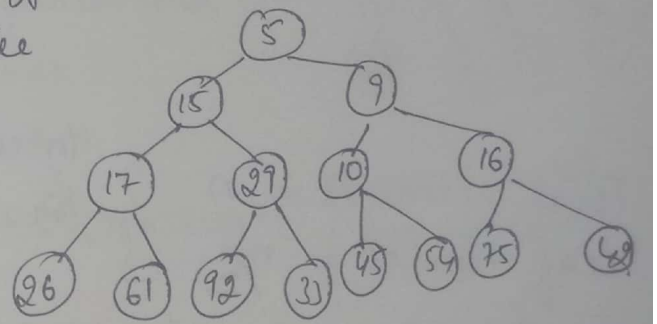
A [75, 26, 42, 15, 92, 10, 5, 17, 61, 29, 33, 45, 54, 60, 16, 9]

1) create B. Tree



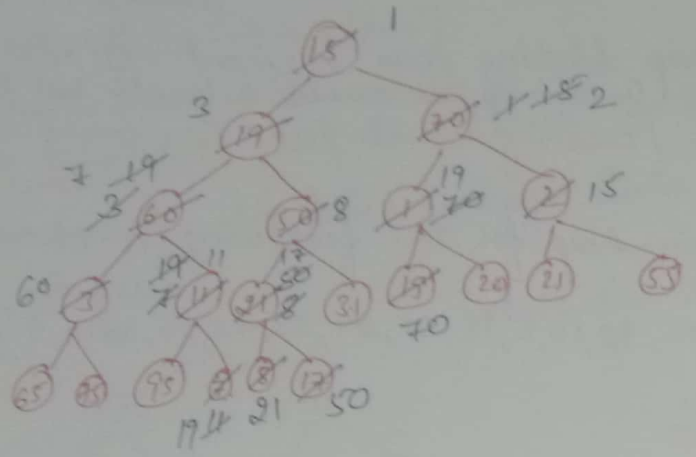
2) Apply Min-heapify top (parent ask child.) from bot

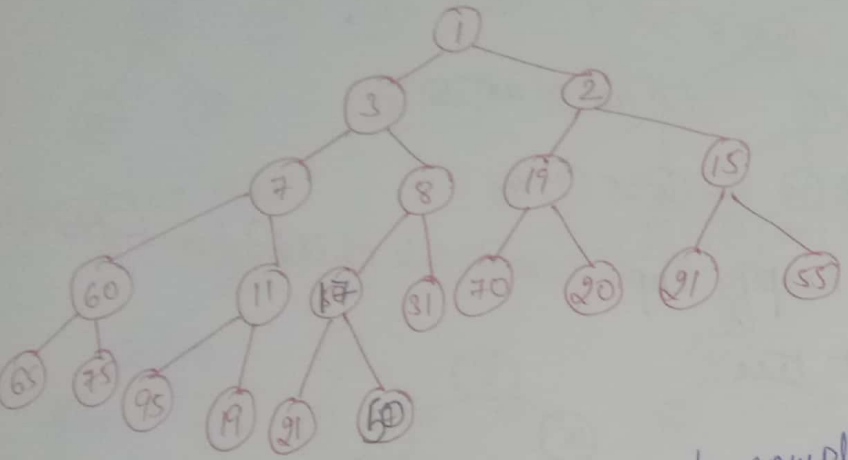
3) Resultant tree



ex2

15, 19, 70 60 50 1 2 3 11 21 31 19 20 21
55 65 75 95 7 8 17 O(n) time





if Total elements = n } In complete binary tree
then leaf nodes = n/2

Total swaps from bottom each level = $\frac{n}{2^0}\times 0 + \frac{n}{2^1}\times 1 + \frac{n}{2^2}\times 2 \cdots \frac{n}{2^{\log n}} \times \log n$.

$= n\left[\frac{0}{2^0} + \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots \frac{\log n}{n^{\log n}}\right]$

$= n\left[\left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \cdots + \left(\frac{1}{2}\right)^{\log n}\right]$

$= n * O(1)$

$= n \rightarrow$ swaps.

$= n \rightarrow$ swaps $+ 2n \rightarrow$ comparisons → each node is compared with left & right child so $2 * n$

$\Rightarrow 3(n)$

$\Rightarrow O(n)$

* create a min-heap using brute force = O(n
* create " " " " build heap = O(n)
* HEAP-SORT
min heap continuous deletion gives one by one element as descending order
max " " " " " descending order

Algo heap sort

* create min heap using Build-heap ⇒ O(n)
* Delete one by one and store from Right Hand
continue n times → n

O(log n) → O(n log n) [worst case / Average case]

if best case → O(n * 1) ⇒ O(n)

It is in-place

* It is not stable.

* To find 1st min. ⇒ 1 fighting — 0 comparison
  " " 2nd min. ⇒ 2 " — 1 —
  " " 3rd " ⇒ 3 fighting — 2 —
  To find nth min ⇒ n fighting — (n-1) comparison.
  50th min-finding ⇒ 50th fighting — 49 comparison
  To find 50th min we have to find 1st min, 2nd min, 3rd min, 4th min. ... upto 49th min. ⇒ $49 \times \frac{50}{2}$

$\Rightarrow n \times \frac{(n-1)}{2} \Rightarrow O(n^2)$

$O(n^2)$

① Selection sort n pass.
② Bubble sort 1 pass

Min-Heap Tree

1. Creating min-heap ⇒ O(n) Build heap
2). Insertion ⇒ O(1) [Best case)
   O(logn) [worst / Average case)
3) Deletion ⇒ O(1) [B.C)
   O(logn) (WC, A C]
4) @ 10th (10 ways) min. finding ⇒ $\frac{9 \times 10}{2} = 45 = O(1)$
⑤ → Delete 9 min ⇒ 9 logn.
⑥ 10 pass of selection sort = O(n)

3) max-element finding ⇒ $\frac{(n-1)n}{2} = O(n^2)$
* Bubble sort 1st pass ⇒ O(n)

GREEDY TECHNIQUE

Sorting techniques

Comparison based Sorting techniques

	B.C	W.C (upper bounds)	A.C
* Merge Sort	nlogn	nlogn	nlogn.
* Quick Sort	nlogn	n^2	nlogn
* Bubble Sort	n^2	n^2	n^2
* Selection sort	n^2	n^2	n^2
* Insertion Sort	n	n^2	n^2
* Heap Sort	n	nlogn	nlogn

Non-comparison based S.T (Assumptions are required) — Linear time complexity of O(n). Read from coleman

- Radix sort
- Counting Sort
- Bucket Sort

} O(n)

In all comparison based S. Tech. upper bounds what will be lower to bound
nlogn

In all comparison based Sorting Tech. Lower bounds, lower bound is (n) → ans. (Best case)

(4 marks)

GREEDY TECHNIQUES

Note: In greedy Technique most of the problems contain n-i/p and our objective is finding a subset which will satisfy our conditions and optimize our goal.

1. Solution Space:
Set of all possible solutions over the given n- no. of i/p's is called solution space.

2. Feasible Solution:
Set of all possible solutions which will satisfy our conditions.

3. Optimal Solution.
Those feasible solutions which will optimize our goal is called optimal solution. Need not be unique.

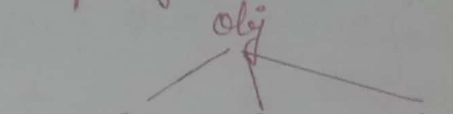
Applications of Greedy
1. Job sequencing with deadlines
2. Knapsack problem.
3. Huffman coding
4. Optimal Merge pattern
5. Minimum Cost spanning tree
   (i) Kruskal
   (ii) Prims

6. Single Source shortest path
   (i) Dijkstra's Algo.
   (ii) Bellman-ford algo
   (iii) Breadth first traversal

KNAPSACK PROBLEM (Real/fractional)

i/p : n - objects — weight, Profit

knapsack capacity = M



$0 \le x_i \le 1$

1) Problem: $\sum_{i=1}^{n} w_i > M$

2) Feasible: $\sum_{i=1}^{n} w_i x_i \le M$

3) Optimal $\sum_{i=1}^{n} x_i * P_i$ max

Ex-1

$n = 3$ $\quad m = 20$

Object	ob_1	ob_2	ob_3
Profit	25	24	15
weight	18	15	10

w_2 P_i

$15 * \frac{24}{15} + 5 * \frac{15}{10} \Rightarrow 24 + 7.5 = 31.5$ Profit max

$\sum_{i=1}^{n} w_i x_i = 0 \times 18 + 1 \times 15 + \frac{5}{10} \times 15 = 15 + 7.5 = 22.5$

(x_1, x_2, x_3)

$\frac{25}{18}$ 1.3 $\quad \frac{24}{15}$ 1.6 $\quad \frac{15}{10}$ 1.5

Note:
In greedy knapsack we will always get optimal ~~knapsac~~ solution by giving priority to both profit and weight.

Q $n=7$ $M=25$

Objects	ob_1	ob_2	ob_3	ob_4	ob_5	ob_6	ob_7
Profits	10	7	15	22	16	50	40
Weights	2	1	4	6	3	8	6
P_i/w_i	5	7	3.75	3.63	5.33	6.25	6.66

$\sum_{i=1}^{n} w_i \times p_i = 2\times\frac{10}{2} + 1\times\frac{7}{1} + 4\times\frac{15}{4} + 1\times\frac{22}{6} + 3\times\frac{16}{3} + 8\times\frac{50}{8} + 6\times\frac{40}{6}$

$\sum_{i=1}^{n} w_i = 2 + 1 + 4 + 1 + 3 + 8 + 6$

$\sum_{i=1}^{n} w_i p_i = 10 + 7 + 15 + \frac{22}{6} + 16 + 50 + 40$

$= 141.66$ (Max. Profit)

KNAPSACK ALGO (Fractional)

1). for ($i=1$ to n) $A[i] = P_i/w_i$ } $\Rightarrow O(n)$

2). Sort array A in decreasing order } $\Rightarrow O(n \log n)$ By merge sort/ heap sort.

3). Take one by one object until ~~k~~ capacity of knapsack becomes zero } $O(n)$

Final $\Rightarrow n + n\log n + n \Rightarrow O(n \log n)$

Q what is the time complexity of knapsack problem if objects are already arranged in P_i/w_i decreasing order / increasing order (sorted)

$O(n)$

JOB SEQUENCING WITH DEAD LINES

1). Single CPU available → fixed fraction of time alloted to every job
2). No - interleaving. (Round-Robin fail)
3). Arrival time of every job is same (FCFS fail)
4). Running time of every job is 1-unit (SJF fail)

ex1 $n = 4$

jobs:	j_1	j_2	j_3	j_4
Profit:	200	150	300	250
Deadline:	2	1	2	1

optimal soln := (J_4, J_3)

(j_4, j_3) 250 300 = 550 | (j_4, j_1) | (j_2)
(j_1, j_3) | (j_3)
(j_2, j_1) | (j_3, j_1) | (j_4)
(j_2, j_3) | (j_1) | $( )$

} 11 feasible solutions.

ex2 $n = 7$

jobs	j_1	j_2	j_3	j_4	j_5	j_6	j_7
Profit	(25)	(75)	15	(89)	(91)	(60)	(55)
Deadline	5	3	4	2	6	5	3

1	2	3	4	5	6
55	89	75	25	60	91
j_7	j_4	j_2	j_1	j_6	j_5

max profit = 395 penalty ⇒ j_3 (15)

1) Find the max. deadline
2) Take an array of size max. deadline
3) Fill it from right side considering their deadlines & profits

} Apply this procedure in exams

ex3 $n = 9$

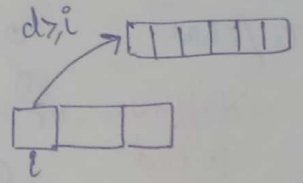
jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
profit	(25)	(15)	35	10	(45)	(55)	5	(16)	(92)
deadline	7	5	2	5	3	2	1	4	3

1	2	3	4	5	6	7
J_5	J_6	J_9	J_8	J_2		J_1
45	55	92	16	15		25

= 248

1) Sort in decreasing order of profit = $O(n \log n)$ merge sort
Job left = J_3, J_4, J_7
Penalty 35 + 10 + 5 = 50
2) Find max deadline
3) Take array of size of max deadline and start } O(
4) from R.H.S
4). For every slot i find a job with a deadline $d \geq i$ using linear search → n

} O(



Doubt:
Is it necessary to sort according to profit? If yes then how it is helpful in further process

HUFFMAN CODING

1) Data Encoding technique
2) Data Compression technique

M = 100

a = 45, b = 15
c = 2, d = 30
e = 5, f = 3

Sender → Receiver

compress.

6 char used
So we can represent 6 char with the help of 3 bits coz $2^3 = 8$ diff. seq.

a = 000
b = 001
c = 010
d = 011
e = 100
f = 101

Uniform coding ⇒

a = 45 * 3
b = 15 * 3
c = 2 * 3
d = 30 * 3
e = 5 * 8
f = 3 * 8

100 char = 300 bits
3 bits / char.

⇒ Can you compress further.

ASCII uniform coding

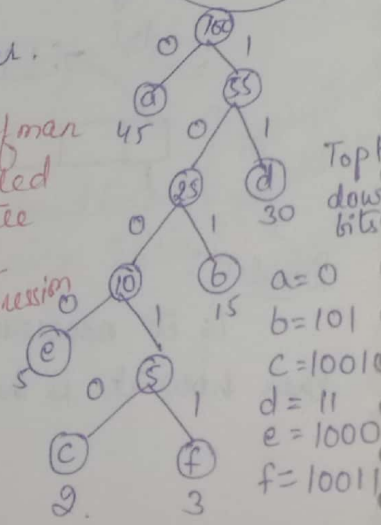
freq * bits
a = 45 * 8
b = 15 * 8
c = 2 * 8
d = 30 * 8
e = 5 * 8
f = 3 * 8

100 char = 800 bits
Avg 1 char = 8 bits

Huffmann coding
↓
Non uniform coding
↓
Ter Freq = ↓ bits
Ver Freq = ↑ bits

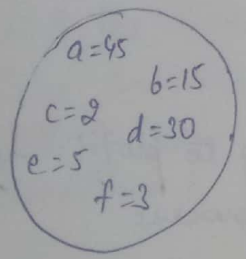
assign ⇒
left 0
right 1

Huffman coded tree for compression



Top to down bits

a = 0
b = 101
c = 1001
d = 11
e = 1000
f = 1001



left less freq
right more freq
in tree

Take 2 least frequency & add & return result

Total Bits = $a \atop 1*45 + b \atop 3*15 + c \atop 5*2 + d \atop 2*30 + e \atop 4*5 + f \atop 5*3$ (10)

= 45 + 45 + 10 + 60 + 20 + 15

100 char = 195 bits
1 char = ' '

merge sort Heap " O(n log n)

Job Scheduling with deadlines

1. Sort decreasing order of profit (n log n)
2. max deadline
3. Take array & start from R.H.S
4. for i ⇒ n find job with deadline d ≥ i using linear search (n × n) n^2

M
M_2
M_3
M_4
M_5
M_6 =

Total :

75 char
1 char

encoded =

60	
100	
101	
10	
11	
1	

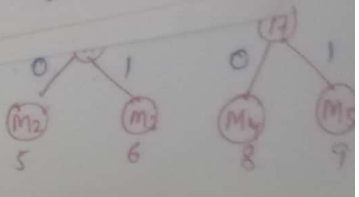
Huffman Coding

1. Data encoding
2. " compression

create min heap. n log n
Delete one min. log n
Delete one more min. log n
Add both min log n
Peek in heap (log n)
& create Huffman coding tree

M_6 ⇒ 3 log n * 2

TATA



1). Create Min-heap O(n)
2) Delete one Min from min heap we will get one min. O(logn)
3). Delete one more min. we will get another min O(logn)
4) Add both min.
5) insert in heap. O(logn)
O(3logn)
6). Follow the procedure to create huffman coding tree. Add min value on left side and higher value on right.
7). After formation of tree assign 0 to left hand and 1 to right hand (Top to bottom)

creation
Time = n + (n-1)3logn O(n²) using array
= n + 3nlogn
= O(nlogn)

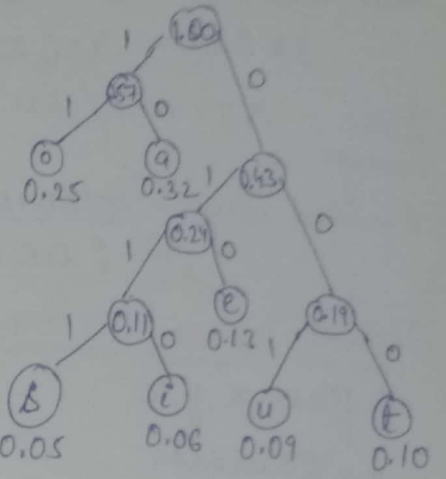
Min heap contains $n.2^n$ elements then one insertion will take ? time.
ie know in n elements min heap one insertion take logn time
so here. ⇒ $\log(n.2^n)$
$= \log n + \log 2^n$
$= \log n + n \log 2$
$= \log n + n$
$= O(n)$

Q M = (a, e, i, o, u, s, t)
0.32 0.13 0.25 0.05 0.10 0.06 0.09

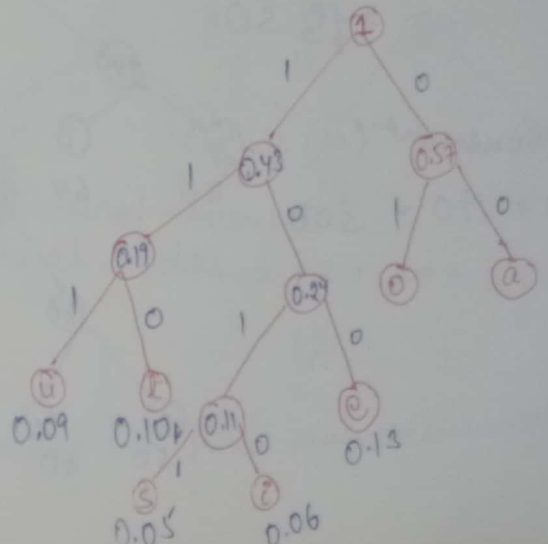
Assume left = 1 Right = 0

a = 10
e = 010
i = 0110
o = 11
u = 001
s = 0111
t = 000

Total bits = 2×0.32 + 3×0.13 + 4×0.06 + 2×0.25 + 3×0.09 + 4×0.05 + 3×0.10
= 0.64 + 0.39 + 0.24 + 0.50 + 0.27 + 0.20 + 0.30
=



a = 00
e = 100
i = 1010
o = 01
u = 111
s = 1011
t = 110



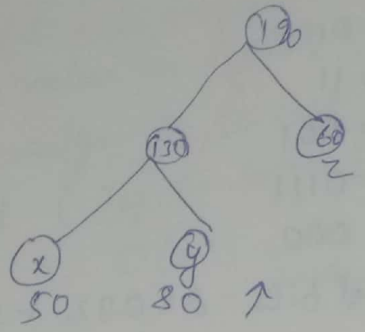
Total $= 2\times0.32 + 3\times0.13 + 4\times0.06 + 2\times0.25 + 3\times0.09 + 4\times0.05 + 3\times0.10$

1 char = 2.54 bits/char
Average

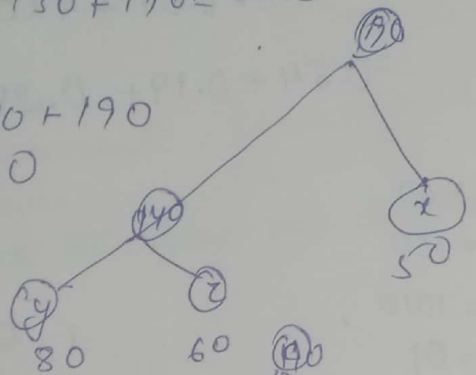
③ Encoded msgs = 10100110000111101111 10
i o e a u & u t

Optimal Merge Pattern

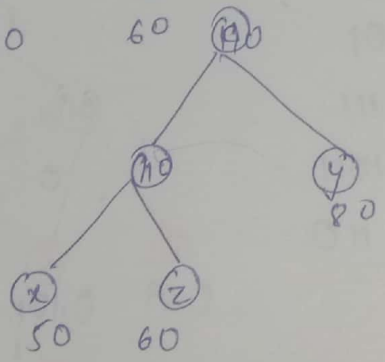
3 files x, y, z
x = 50 records
y = 80 records.
z = 60 records.



records movements = 130 + 190 = 320



records movements = 140 + 190
= 330

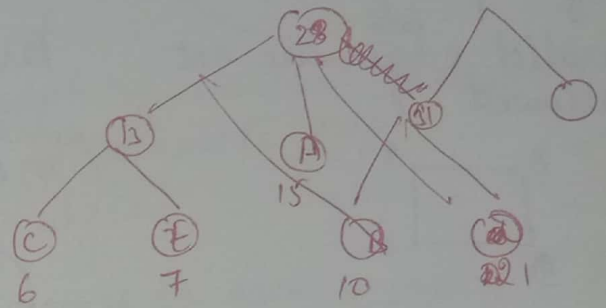


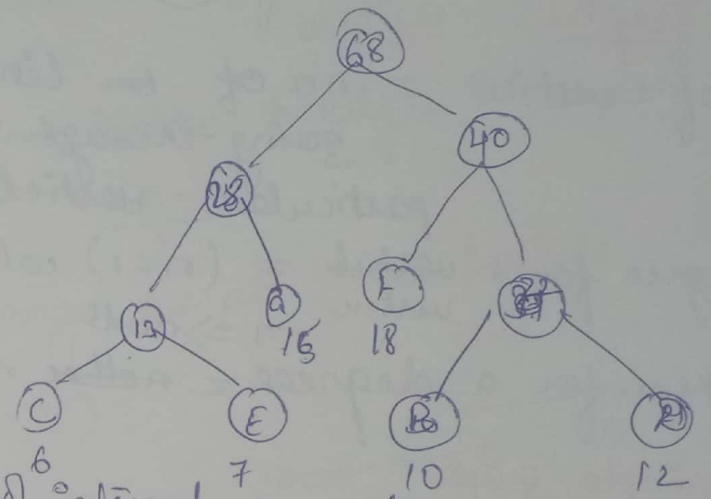
Records movement:
= 110 + 190 = 300

To merge three files. ∴ 3 merge patterns those are m_1, m_2, m_3,

Ques 6 files (2-way merge tree)

A → 15
B → 10
C — 6
D — 12
E — 7
F — 18





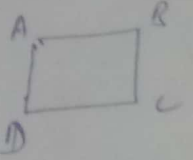
min no. of internal movements
add internal nodes. = 13 + 28 + 22 + 40 + 68
= 171

Note:
start making tree from same value nodes.

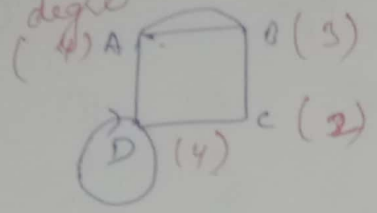
7 COST
MINIMUM SPANNING TREE (MST)

Types of Graph

Simple graph
Self loop is not allowed
&&
Parallel edges are not allowed



Self loop are allowed
OR
Parallel edges are allowed



Simple graphs

degree of vertices = no. of lines (edges) going through a particular vertex

max. degree for a vertex = (n-1) where n ⇒ node

min degree for a degree = nothing (0)

If the simple graph

max. degree of max degree = infinite (undefined)

min degree of max = 0

TYPES OF SIMPLE GRAPHS

In any simple graph any vertex deg contain no. edges = NULL graph

null graph because no edges

In any simple graph if vertex doesn't have maximum vertices → complete graph

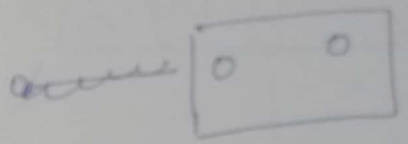
edges

no. of vertices × degree of each = faces
vertices

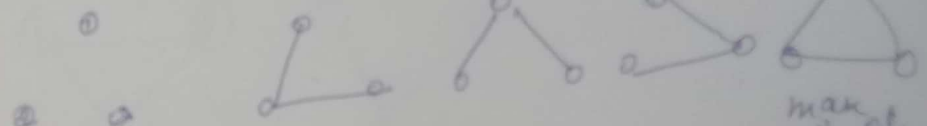
Total No. of edges in complete graph = $\frac{n(n-1)}{2}$.

How many simple graph possible with n vertices

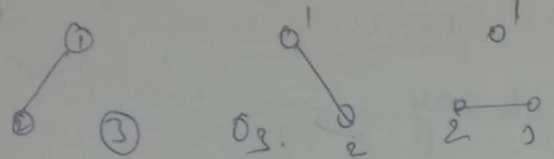
* n=2 (1,2)



* n=3 (1,2,3)



max simple graph



8 - graphs (2^n graphs)

~~For each edges~~ ~~have~~ ~~mode~~

max edges for n vertices = $\frac{n(n-1)}{2}$

Total no. of simple graphs with n vertices = $2^{n(n-1)/2}$

* Note

A simple Graph $G(V, E)$ (max. edges) $2^{\text{max edges}}$

$|E| \leq \frac{v(v-1)}{2}$

$|E| \leq c.v^2$

$|E| = O(v^2)$

$\boxed{\log E = O(\log V)}$

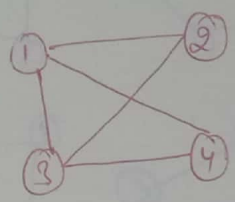
SPANNING TREE

A subgraph 'S' is said to be spanning tree after given graph 'G' iff —

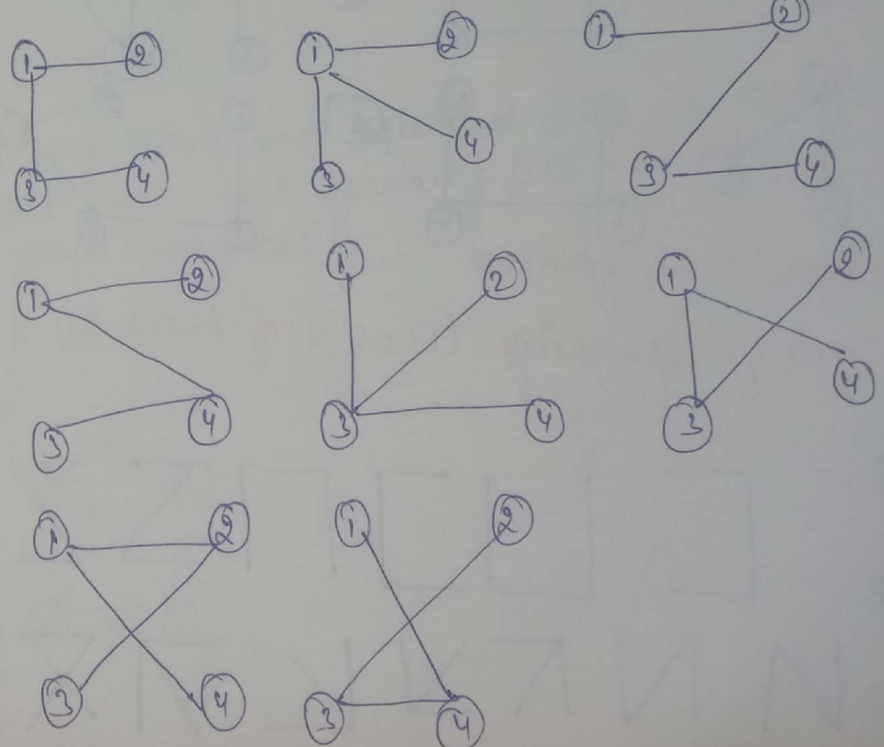
1) It should contain all vertices of G.
2) It should contain (n-1) edges. if G contains n vertices.
3) S cannot contain cycle. no cycle

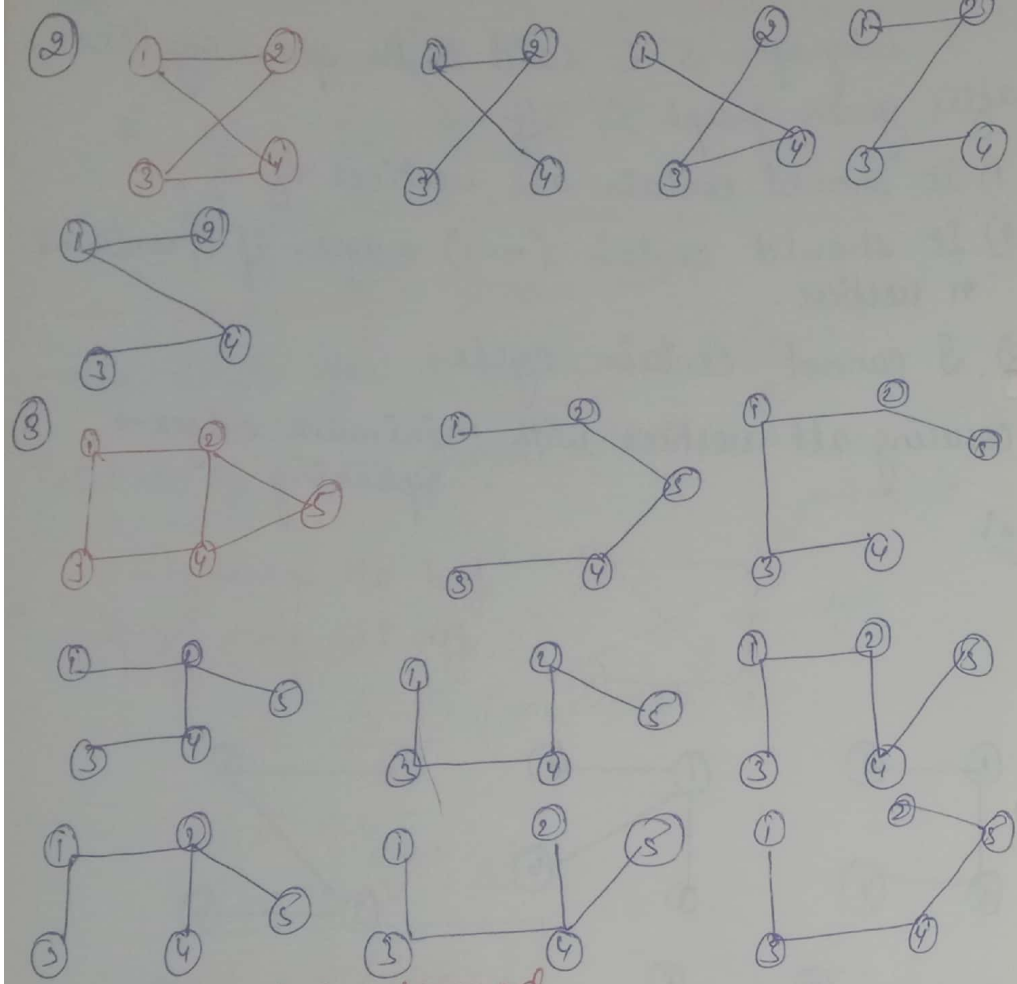
covering all vertices with minimum edges → Spanning.

ex1

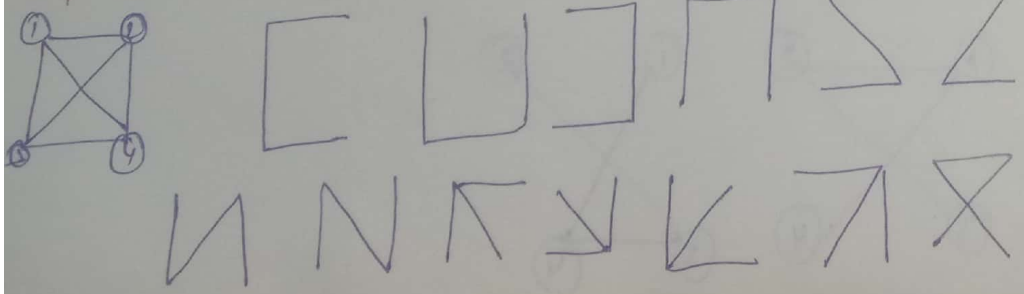


find all spanning tree for the given graph





How many spanning trees are there in K4



Spaning tree $(K_3) = 3$

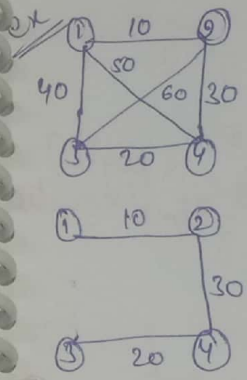
" " $(K_4) = 16$

" " $(K_5) = 125$

" " $K_n = n^{n-2}$

MINIMUM COST SPANNING TREE

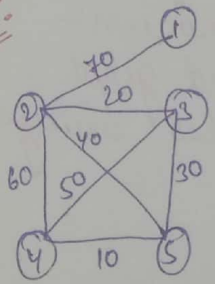
(min. edges and edges weights should also be minimum)



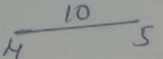
Using Prims and Krushkal algo we can find out min. cost spanning tree for the given graph very easily.

Kruskal's algo

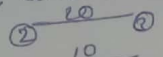
Ex:

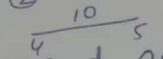


Step 1
* create min heap tree of edge weights } → O(E)
* Find min-cost-edge and add to MST } O(log(E))

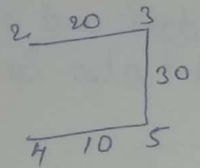


* step 2
Find next min-edge and add to MST } log(E)





* step 3.
Find next min-edge and add to MST
if no cycle



* step 4
Repeat step 3 — 3 log(E)

$40_{2,5}$ } forming cycle
$50_{4,3}$ } so rejected time wasted
$60_{2,3}$ }

$70_{1,2}$ } taken

In case of prims algo we will always get connected graph but in case of kruskal algo we may get disconnected graph in middle of algo but at last we will get connected graph

Note
In kruskal algo while generating minimum-spanning tree it can give disconnected graph in b/w while Prim's in between also always give connected graph.

assume $E \geq V-1$

Time Complexity

Best case
$E + (V-1)\log E$
$E + V\log E$
$E + V\log V$ (coz $\log E = O(\log V)$)

complete graph
$E > V\log V$
coz $E = V^2$

In null graph
$E < V\log V$

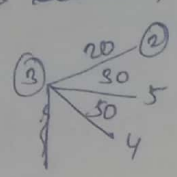
$O(E + V\log V)$

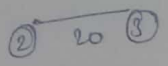
Worst case & AC
$E + E\log E$
$E < E\log E$
$E\log E$
$O(E\log V)$

In starting if we sort the array of edges cost then we will get → $E\log E$
so it would cost more.

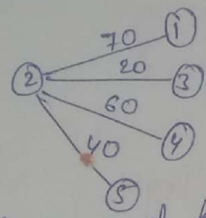
PRIMS ALGO

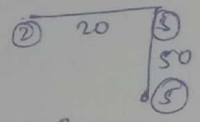
step 1
Choose any vertex and find adjacent of that vertex (how: chosen vertex will ask to all other vertexes by linear search) and ~~min~~ find min.





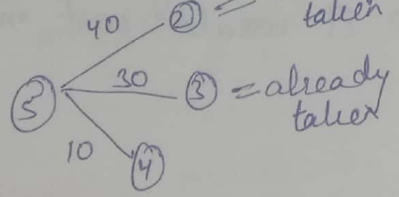
step 2
Find the adjacent of new vertex

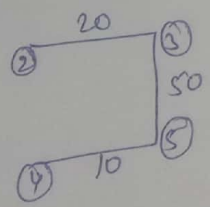




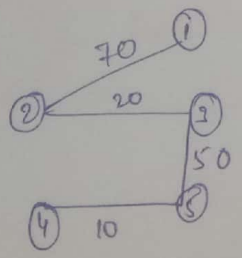
and take minimum of these and previous.
vertices

step 3
Find the adjacents of new vertex and take minimum of these and previous && add if no-cycle.

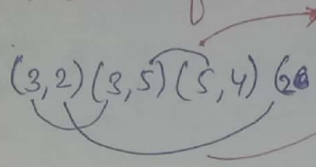




step 4
Repeat 3rd step

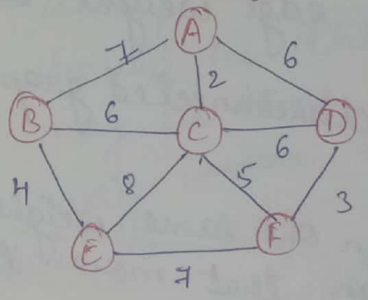


Note
The spanning tree found by kruskal and prims. may or may not be same (if 2 or more edges have same weight) but at last the sum of cost would be same for both the cases.

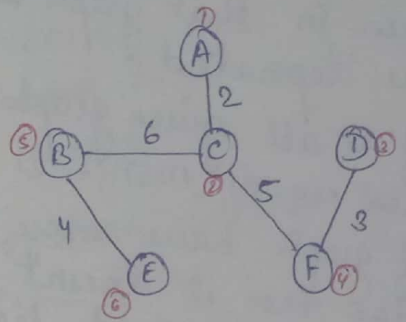


exam
① Always check first adjacency then min cost

Ques Consider the following graph.



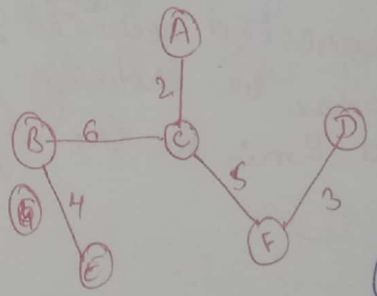
Kruskals



(A,C) (D,F) (B,E) (CF) (B,C

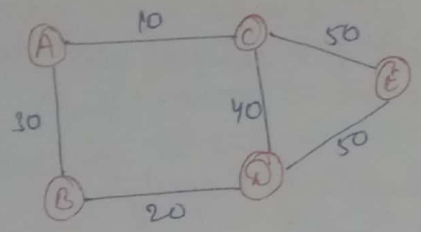
no connectivity property only cycle property

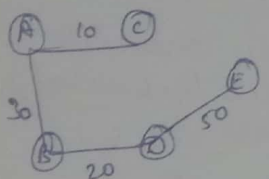
Prims

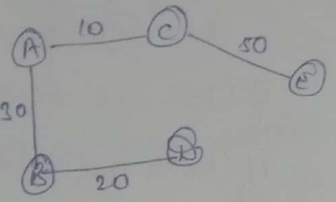


(A,C) (C,F) (F,D) (C,B) (B,E)

Ques Consider the following graph How many min cost spanning tree possible







Note:

* For the given graph more than one MST may be possible but the cost will be same
* If all e more than one MST is possible then in that graph some edge weight are repeated.
* If at all given graph is disconnected graph then no. of MST = 0
* If graph have more than one same weighted edge then it doesn't mean that more than one MST would be formed.

Ques Let G be a undirected connected weighted graph with n vertices and e_{max} be the edge with max edge weight and e_{min} be the edge with min edge weight.

True/False

(i) e_{min} should be there in MST of G. True
(ii) e_{max} should be there in MST of G False (coz of should be)
(c) e_{max} may be there in MST of G. True.
(d) G contain unique MST. True
(e) e. If e_{max} is in MST then its removal from G must disconnect G. True

Ques Let G be a undirected connected weighted graph with n vertices & w be the min. edge weight among all edge weights and e be a specific edge with weight w.

Soln True/False

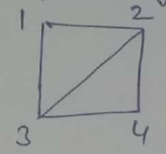
(i) e should be there in every MST of G. Fa
(ii) e may be there in some MST of G. Tru
(iii) Every ~~should~~ ~~be~~ ~~good~~ MST must contain an edge with weight w. True.
(iv) If MST not contain e then in that cycle all edge weight are w. True

Ques Consider the following graph

GRAPH REPRESENTATION

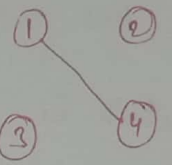
i) Adjacency Matrix
ii) Adjacency List

i) Adjacency Matrix



Best if
{ Dense graph
or
more edges

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



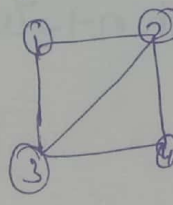
Sparse

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

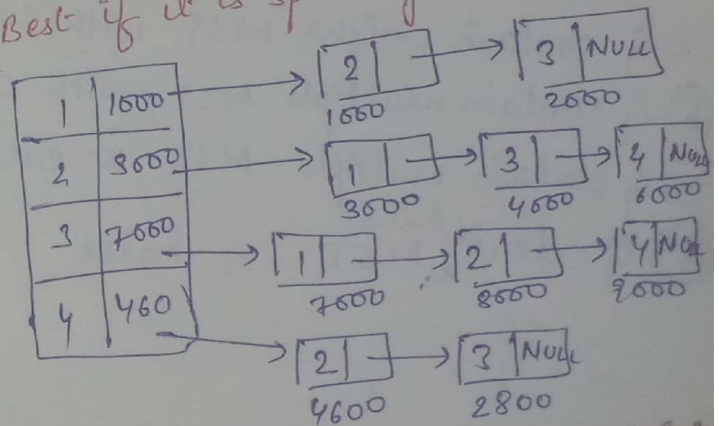
No best case in worst case to find degree of vertex = O(V)

If G(V, E) then to make matrix we will take $O(V^2)$ space

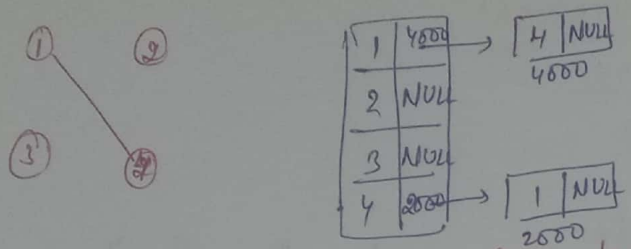
ii) Adjacency List (Best if it is sparse graph / less edges)



⇒



Best case to find the degree of any vertex is O(1)



Space taken by adjacency list for $G(V, E)$

$V + 2E$

↓ ↳ for linked list.

for array.

for complete graph $E > V$

for null graph $E < V$

Let G be a graph with n-vertices. whose adjacency matrix is given by $n \times n$ sq. matrices, in which

(i) All diagonal elements are 0's.

(ii) All non-diagonal elements are 1's.

Then check the following statements are true/false.

G contain unique MST with cost $n-1$ → False

G contain multiple MST with different cost → False

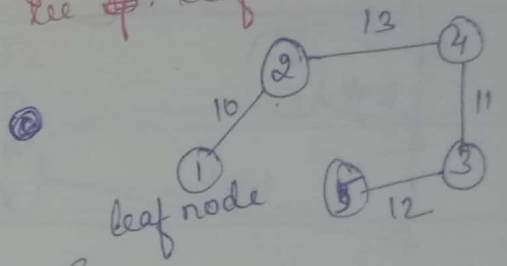
G contain multiple MST of each of cost $n-1$ → True

G - dont have MST → False

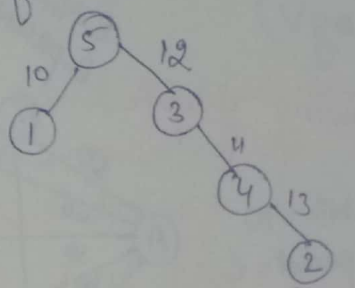
Ques Consider the following graph.

	1	2	3	4	5
1	0	10	10	10	10
2	10	0	15	13	17
3	10	15	0	4	12
4	10	13	11	0	14
5	10	17	12	14	0

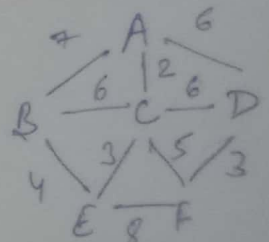
(i) What will be the cost of Min-Cost Spanning tree for the above graph where in that min-cost spanning tree will be $1-V$ leaf node.



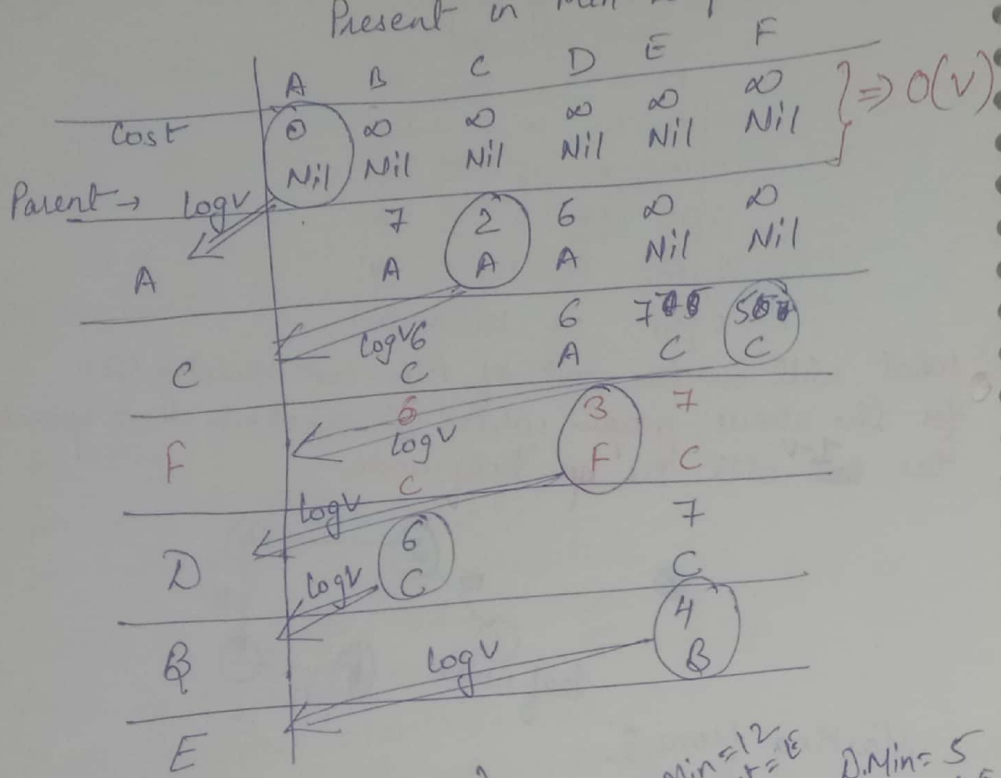
Starting from 5

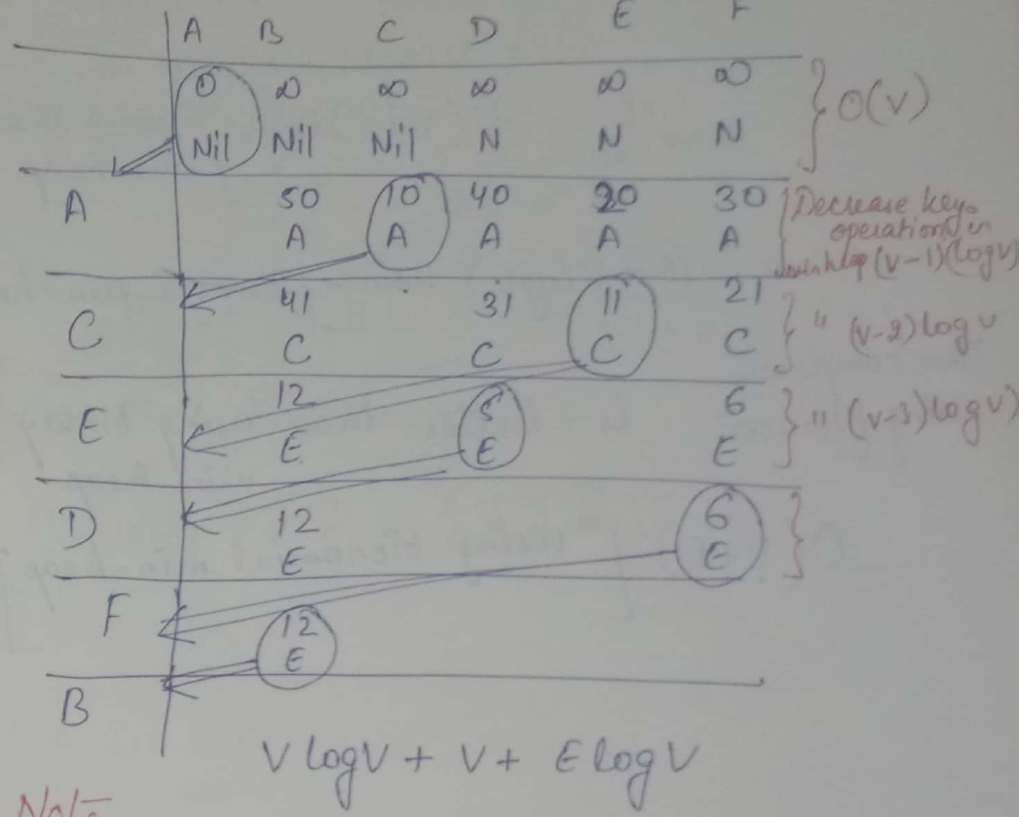


TIME COMPLEXITY OF PRIM'S ALGO [using min heap

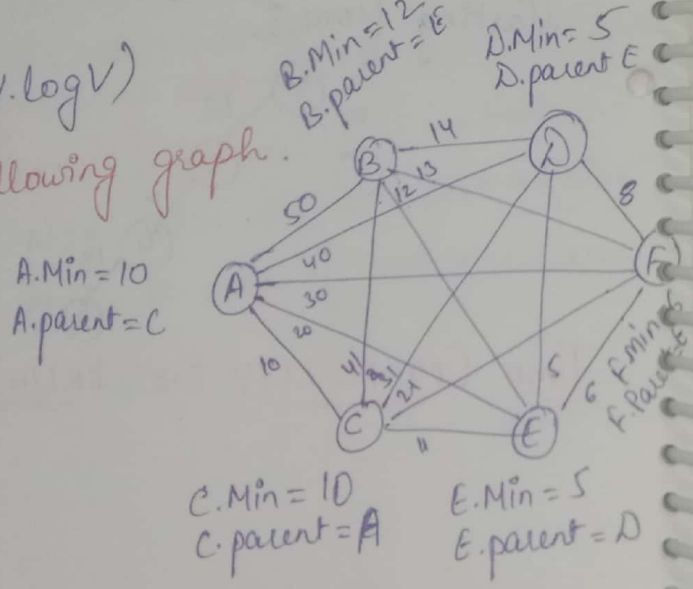


Present in Min-heap





Ques Consider the following graph.



Note

Increase key and decrease key operation in min-heap and max-heap will take O(logn) (worst case and Average case)

O(n) in best case

Also Priority Queue is also called Min-heap.

$(V-1)+(V-2)+(V-3)\ldots\ldots = E$ (Edges)

imp Time complexity (Prims) = $V\log V + V + E\log V$

$= V\log V + E\log V$

imp $= O(V+E)\log V$ [using binary min-heap]

$= O(E + V\log V)$ using fibnocci min-heap

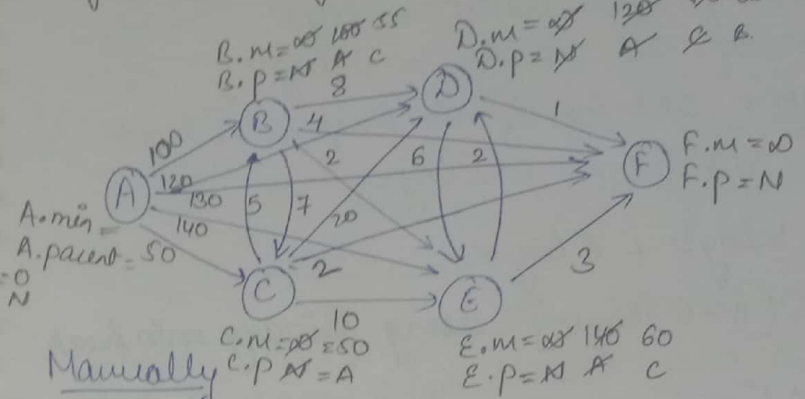
Time complexity using fibonacci is better than using binary min-heap.

$O(V+E)$ [using binomial min-heap]

ADA

Single Source Shortest Path

↳ Dijkstra's Algo



Manually

(a) A - A = 0

(b) A - B = 55

(c) A - C = 50

(d) A - D = ~~62~~ 59

(e) A - E = ~~60~~ 57

(f) A - F = 52

	A	B	C	D	E	F	
	0	∞	∞	∞	∞	∞	⇒ O(V)
logV	N	N	N	N	N	N	
		100	50	120	140	130	⇒ 5 logV
A		A	A	A	A	A	
logV		55		70	60	52	⇒ 4 logV
C		C		C	C	C	
logV		55		70	60		⇒ 0 logV
F		C		C	C		
logV				63	57		⇒ 2 logV
B				B	B		
logV					59		⇒ logV
E					E		
D							= E logV
V logV							

Time complexity of Dijkstra's algo.

* Initially to create min heap = $O(V)$
- Deletion of each vertex = $\log V$
- Deletion of all vertex = $V \log V$

After deletion ordering of min heap = $E \log V$

$T(Dij) = V + V\log V + E\log V$

$= V\log V + E\log V$

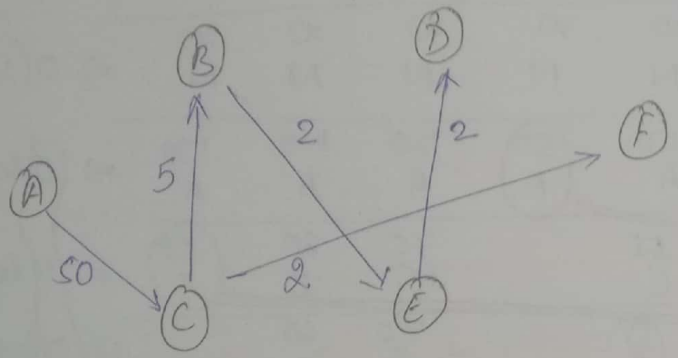
① $= O[(V+E)\log V]$ using Binary min heap.

② (not in syllabus) $= O[E + V\log V]$ using Fibonacci min heap

③ (not in syllabus) $= O(V+E)$ using binomial min heap.

④ By array.

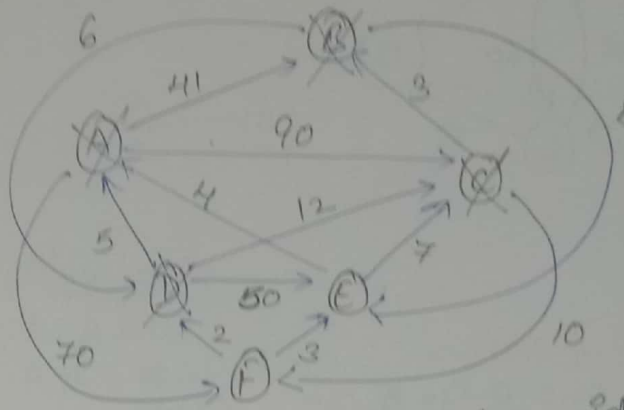
$O(V^2)$.



1) A-D (59)

2) A→C→B→E→D

Q Consider the following graph



A. min = ∞, 0 (1) P = N (0)

B. min = ∞, 90, 41 (2) P = N, A (A)

C. min = ∞, 90, 97, 47 (4) P = N, A (B)

D. min = ∞, 59 (3) P = N (A)

E. min = ∞, 121, 80, 77 (5) P = N (D)

F. min = ∞, 70, 69 (6) P = N, A (C)

(i) print the seq. of vertices identified by Dij's algo if when algo started from 'A'

(ii) A, B, D, C, F, E

(iii) what will be the cost of shortest path from A to E.

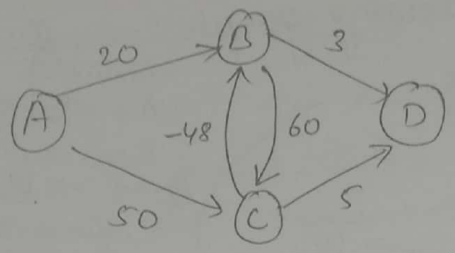
A - E (72)

(iv) what will be the shortest path from A - E

A → B → D → C → F → E

	A	B	C	D	E	F
	0 N	∞ N	∞ N	∞ N	∞ N	∞ N
A		41 A	90 A	∞ N	∞ N	70 A
B			90 A	47 B	121 B	70 A
D			59 D		97 D	70 A
C					97 D	69 C
F					72 F	

Ques



Manually

A - A = 0
A - B = 2
A - C = 50
A - D = 5 — A-C-B-D = 5
A-C-D = 55.

Dijkstra's

	A	B	C	D
	0 N	∞ N	∞ N	∞ N
A		20 A	50 A	∞ N
B			50 A	23 B
D				50 A
C				

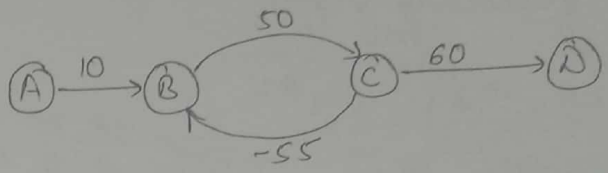
A-B-D-C
20+3+

A - B = 20
A - C = 50
A - D =

Note:

(a) If the graph contain negative edge wts Dijkstra's algo may give wrong ans.
(b) If the graph contains all positive edge wts then Dijkstra's algo always give correct ans.

Ques.



Manually

A - A = 0
A - B = 5, 0, -5, -10 ... $-\infty$ coz of negative wt edge.
A - C = 60 ... $-\infty$ (")
A - D = 120 ... $-\infty$ (")

By Dijkstra's

	A	B	C	D
	0 N	∞ N	∞ N	∞ N.
A		10 A	∞ N	∞ N
B			60 B	∞ N
C				120 C
D				

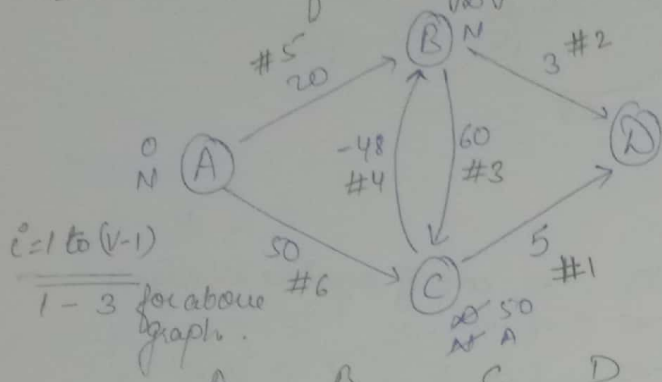
A - A = 0
A - B = 10
A - C = 60
A - D = 120

Note:

If the graph contains negative edge wt. cycle then Dijkstra algo always will fai

BELLMAN FORD ALGO

Consider the following graph



using array. Bellman ford is solved.

	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	20	50	∞
	N	A	A	N
i=2	0	2	50	23
	N	C	A	13.
i=3	0	2	50	5
	N	C	A	B

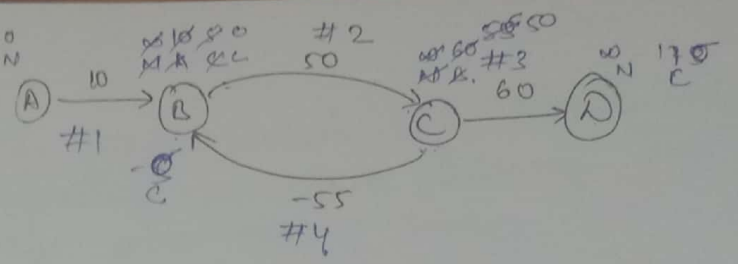
E, E, E → O(VE)

For all the graphs → O(VE)

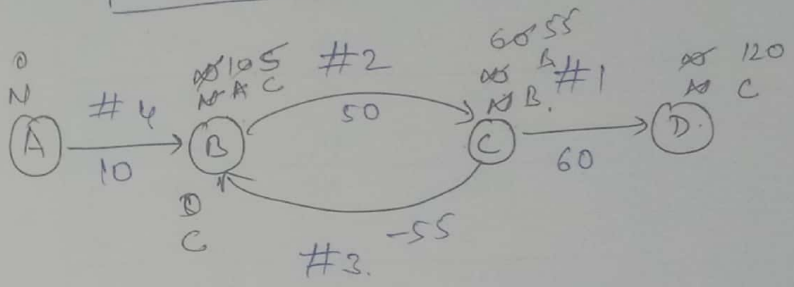
If graph is complete then the complexity $= O(V^3)$

Max complexity of Dijkstra algo $= VE \log V$
$= V^2 \log V$

While in case of Bellman ford algo the Max complexity $= O(V^3)$



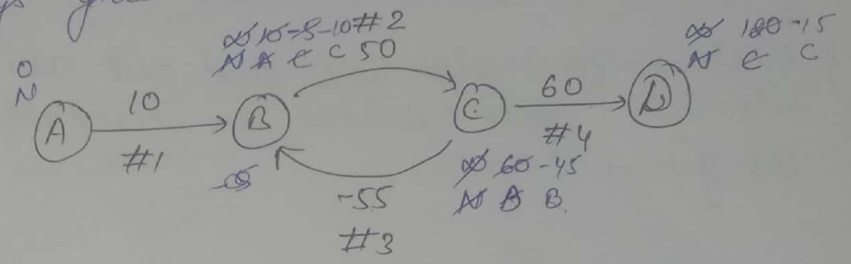
	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	5	60	120
	N	C	B	C
i=2	0	0	55	115
	N	C	B	C
i=3	0	-5	50	110
	N	C	B	C



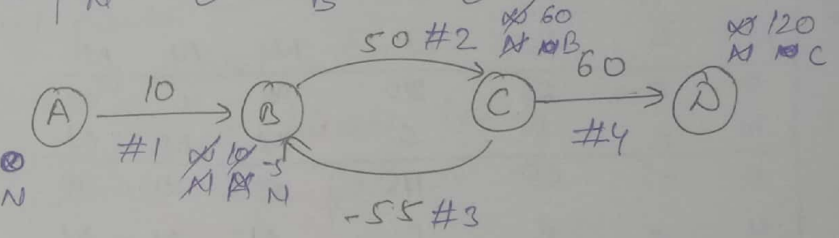
	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	10	∞	∞
	N	A	N	N
i=2	0	5	60	∞
	N	C	B	N
i=3	0	0	55	120
	N	C	B	C

Note:
If the graph contain all positive edge wt or negative wt. then Bellman ford will always give correct answer.

Ques



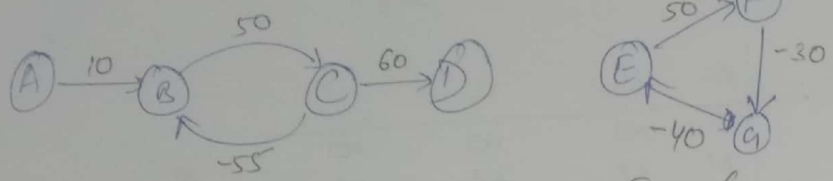
	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	10	∞	∞
	N	A	N	N
i=2	0	-5	60	120
	N	C	B	C
i=3	0	-10	-45	-15
	N	C	B	C



	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
i=1	0	-5	60	120
	N			N
i=2				
i=3				

Note — Bellman ford → Dynamic prog. (more time) (always give correct ans)

* If the graph contain negative edge wt. cycle then also bellman ford algo will give correct ans.
* Bellman ford algo will find out all negative edge wt. cycle which are reachable from the given cycle.



	A	B	C	D	E	F	G
	0	∞	∞	∞	∞	∞	∞
	N	N	N	N	N	N	N
i=1	0	10	∞	∞	∞	∞	∞
	N	A	N	N	N	N	N
i=2	0	5	60	∞	∞	∞	∞
	N	C	B	N	N	N	N
i=3	0	0	55	120	∞	∞	∞
	N	C	B	C	N	N	N
i=4	0	-5	50	115	∞	∞	∞
	N	C	B	C	N	N	N

decreasing with same diff. i.e. 5 means part of same cycle

decreasing with same diff. i.e. ∞ means part of same cycle

Dijkstra algo → Greedy algo (less time)
(some time may give wrong ans)

Dynamic Programming

Greedy Technique → Sometimes wrong answer.

Dynamic Prog → Always correct answer

G.T – few possibilities will be covered
D.P → All possibilities " " "

G.T – less time
D.P – More time

Applications of Dynamic Programming

1. Fibonacci series
2. longest common subsequence
3. Matrix Multi chain multiplication
4. Travelling salesman problem
5. 0/1 knapsack
6. All pairs shortest path
7. Sum of subset
8. Optimal cost binary search tree.
9. Optimal merge pattern

Fibonacci Series

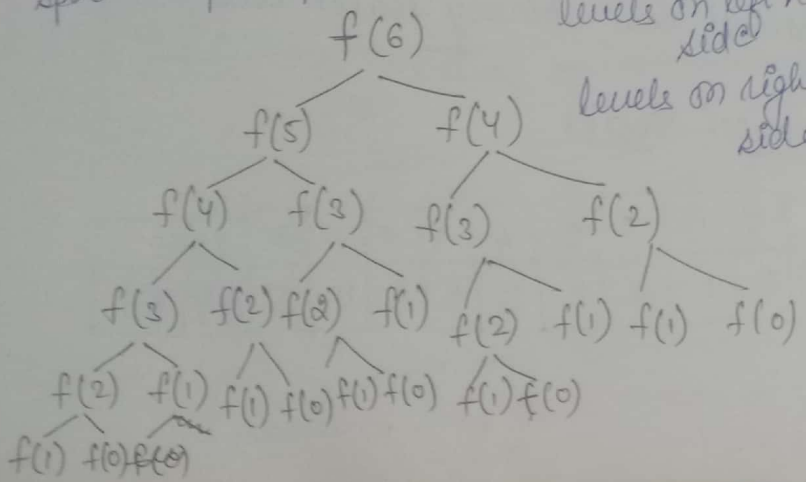
n	0	1	2	3	4	5	6	7	8	9	10
fib(n)	0	1	1	2	3	5	8	13	21	34	55

Recurrence Relation: $f(n) = \begin{cases} n & \text{if } n=0 \text{ or } n=1 \\ f(n-1)+f(n-2) & \text{if } n>1 \end{cases}$

Recursive prog

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

* For time complexity we need recursive relation of time of recursive prog.
* For a single recursive prog we can write many recursive prog like for time complexity, space complexity, value etc.



levels on left hand side = 6

levels on right hand side = 4

$f(n)$ = n-level complete binary tree (upper bound)

$= 2^n - 1$ nodes

$= 2^n - 1$ function call

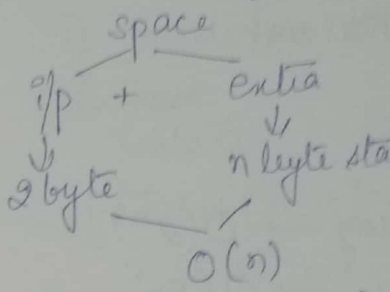
$= 2^n \times 1$ function calling cost $\Rightarrow 2^n \times 1$-addition

$= 2^n \times O(1) = O(2^n)$

Time complexity $= O(2^n)$ upper bound

Space: i/p (only one i/p i.e. 'n' of 2 bytes)

extra (stack = no. of level of tree



Brute force & Dynamic prog will always give correct answer

⇒ In the above recursive tree many function calls are repeated (overlapping subproblems) So what is the need of executing the same function again and again.

⇒ In the case of dynamic programming we will compute only distinct function calls. because as soon as we compute

any func. we will store its value in table so that we can re-use it further whenever it is needed.

imp Ques How many distinct func.n calls are there in fibanocci of 'n'?

Ans $f(6) \Rightarrow f(5), f(4), f(3), f(2), f(1), f(0)$

$= 6+1$ calls.

$f(n) = n+1$ distinct func. calls.

Time $= (n+1) * (\text{one func. call cost})$

$= n * 1$-addition cost

$= n * O(1)$

$= O(n)$.

without dynamic prog $\rightarrow O(2^n)$

with " " $\rightarrow O(n)$

Space using dynamic prog $\rightarrow$ i/p + extra

$= 2$ Byte + stack + Table

$= 2$ Byte + n Bytes + $(n+1)$ Byte (for storing distinct func. call)

$= 2n$ Bytes

$O(n)$

without dynamic prog $= O(n)$

Dynamic prog will give more level tree than Divide & conquer

In D&C we don't bother about distinct func. call whereas in Dynamic Prog we do.

Fibonacci prog with dynamic Prog

Table → global array.

```
fast-fib(n)
{
  if (n==0 || n==1)
    return (n)

  else
  {
    if (Table[n-1]==null)
      Table[n-1] = fast-fib(n-1);
    if (Table[n-2]==null)
      Table[n-2] = fast-fib(n-2);
    Table[n] = Table[n-1] + Table[n-2]
    return (Table[n]);
  }
}
```

LONGEST COMMON SUBSEQUENCE (LCS)

Subsequence of a given sequence is just the given sequence only. in which zero or more symbols are left out.

ex:

$S = (A, B, B, A, B, B)$
$\quad\quad 1\ 2\ 3\ 4\ 5\ 6$

$SS_1 = (B, B, B, B)$
$\quad\quad 2\ 3\ 5\ 6$

$SS_2 = (A, A)$
$\quad\quad 1\ 4$

$SS_3 = (B, A, B, A)$ X some other sequence
$\quad\quad 2\ 4\ 5\ ?$

$SS_4 = ( )$

$SS_5 = (A, B, B, A, B, B)$

$SS_6 = (A, B, A, B)$
$\quad\quad 1\ 2\ 4\ 5$

$SS_7 = (A, A, B, B)$ X

Common subsequence : Z is a common subsequence of ~~two~~ sequences x & y iff z is subsequence to x & Z is subsequence to y only.

ex:
$\quad\quad 1\ 2\ 3\ 4\ 5\ 6$
$x = (A, B, B, A, B, B)$

$y = (B, A, A, B, A, A)$
$\quad\quad 1\ 2\ 3\ 4\ 5\ 6$

$Z_1 = (A, A)$

$Z_2 = ( )$

$Z_3 = (A, B, A)$

$Z_4 = (B, A, B, A)$ X

$Z_4 = (A)$

4 length common subsequence is not possible
length is longest
so 3 is longest common subsequence.

Ques
$\quad\quad 1\ 2\ 3\ 4\ 5\ 6\ 7$
$x = (A, B, B, A, B, B, A)$

$\quad\quad 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$
$y = (B, A, B, A, B, A, B, A)$.

0 - length
$Z_1 = ( )$

1 - length
$Z_2 = (A)$

2 - length
$Z_3 = (AB)$

3 - length
$\quad\quad 1\ 2\ 4$
$Z_4 = (A\ B\ A)$
$\quad\quad 2\ 3\ 4$

~~5~~ - length
$Z_5 = (A, B, A, B, A)$

~~4~~ - length
$Z_6 = (A, B, AB)$

6 - length
$Z_7 = (B, B, A, B, B, A)$

longest common subsequence is of length 6

How to find out LCS using Dynamic programming

ex1 $x = (A, A, B, A, B)$ $\quad 1\ 2\ 3\ 4\ 5$

$y = (B, A, A, B, A)$ $\quad 1\ 2\ 3\ 4\ 5$

$LCS(4,5) = 1 + LCS(3,4)$

$= 1 + LCS(2,3)$

$= 1 + LCS(1,2)$

$= 1 + LCS(0,1)$

$= 1 + 0$

$= 1$

4 Ans

ex2 $x = (A\ B\ A\ A\ B\ A\ B)$ $\quad 1\ 2\ 3\ 4\ 5\ 6\ 7$

$y = (B\ B\ A\ A\ B\ B\ A\ B)$ $\quad 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$

$LCS(7,8) = 1 + LCS(6,7)$

$= 1 + LCS(5,6)$

$= 1 + LCS(4,5)$

$\max \{ LCS(3,5), LCS(4,4) \} \Rightarrow 4$ (3, 4)

$LCS(7,8) = 7$ Ans

Note

LCS of (m, n)

LCS(m, n) = length of the longest common subsequence possible with two seq. (x, y) where x contains m symbols and y contains n symbols.

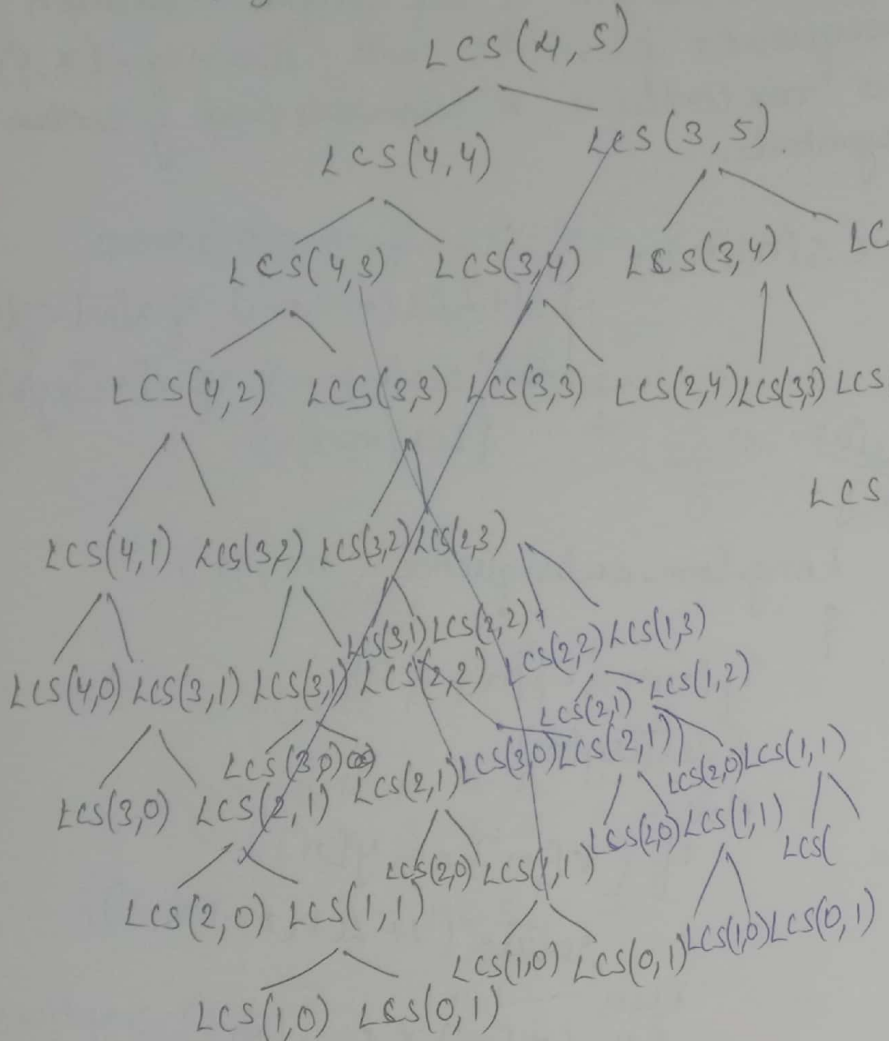
$$LCS(m,n) = \begin{cases} 0 & \text{if } m=0 \text{ or } n=0 \\ 1 + LCS(m-1, n-1) & \text{if } x[m] == y[n] \\ \max \begin{cases} LCS(m, n-1) \\ LCS(m-1, n) \end{cases} & \text{if } x[m] \neq y[n] \end{cases}$$

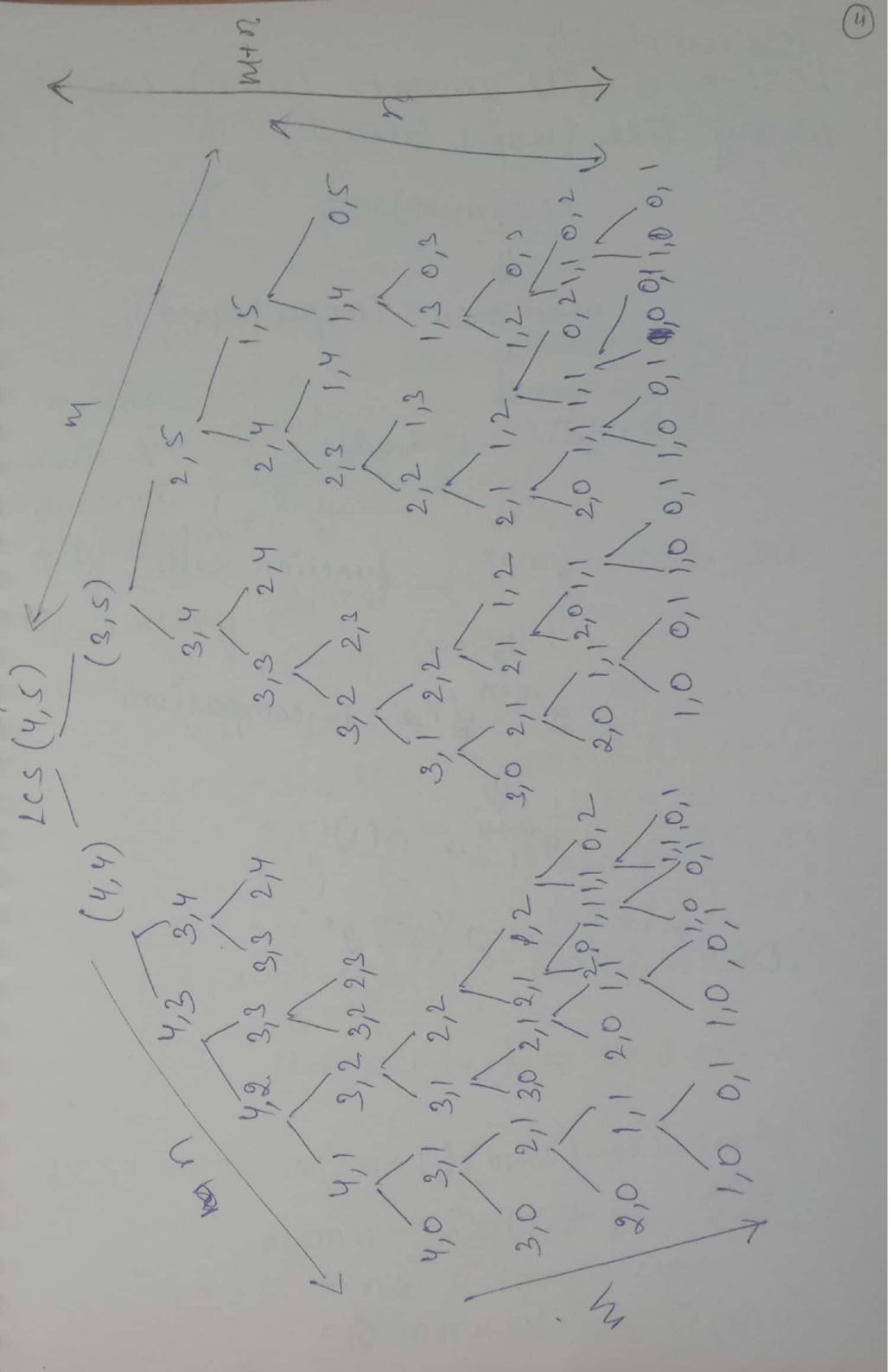
Recursive Prog

```
Long-Com-subsequence (m, n)
{
  if(m==0||n==0)
    return(0)
  else
    if (x[m]==y[n])
      return (1+LCS(m-1,n-1))
    else
    {if (x[m]!=y[n]) => optional
      a = LCS (m, (n-1));
      b = LCS (m-1,(n-1));
    } c = max(a,b)
    return c;
}
```

x = A A A A
    1 2 3 4

y = B B B B B
    1 2 3 4 5





Time complexity

LCS(m,n) will generate (m+n)-complete binary tree (upper bound)

LCS(m,n)

⇓

m+n → CBT (upper bound)

level

⇓

$2^{m+n}-1$ nodes.

⇓

2^{m+n} — function calls.

⇓

2^{m+n} * 1-comparison

⇓

2^{m+n} * O(1)

$O(2^{m+n}) \Rightarrow O(2^m \cdot 2^n)$

Space —

i/p + extra

⇓ ⇓

m & n stack

⇓

m+n

$\Rightarrow O(m+n)$.

In the above recursive tree many func^n calls are repeating so ~~many~~ we will go to dynamic programming which will solve only distinct function call.

imp Try How many distinct function calls are there in LCS of (m,n)?

LCS(4,5) = LCS(4,4), LCS(3,4), LCS(3,5),

= LCS(4,3), LCS(3,3) LCS(2,3), LCS(3,4)

= LCS(2,5), LCS(4,2) LCS(3,2), LCS(1,5)

LCS(2,4), LCS(4,1) LCS(3,1) (LCS(2,4))

LCS(1,4) LCS(1,3) (LCS(1,2) LCS(0,5)

LCS(0,4) LCS(0,3) LCS(0,2) LCS(0,1)

LCS(0,0)

(4+1) * (5+1) = 5 × 6 = 30

LCS(m,n) = (m+1)(n+1) → Distinct func. calls.

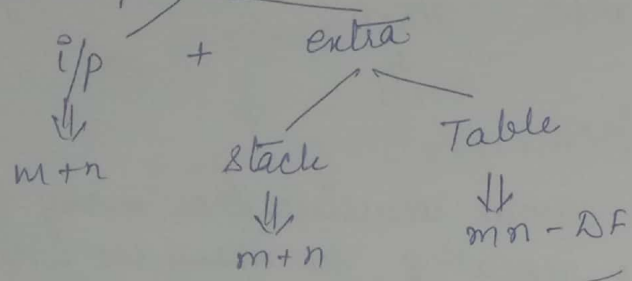
= mn → Distinct func calls.

= mn * 1-comparison

= mn * O(1) = O(mn)

$O(mn)$ → with Distinct func. call.

Space complexity



3. 0/1 KNAPSACK

ex: M=10

obj:	ob_1	ob_2	ob_3
profits:	70	38	58
weights:	7	4	6

For 0/1 knapsack problem greedy algo will fail because of this reason we are going for dynamic programming. which will cover every possibility exactly one time.

ex2 n=5

ob:	ob_1	ob_2	ob_3	ob_4	ob_5
profit	25	75	15	45	35
wt	5	3	2	1	2.

⇓

1) $0/1ks(n,m) = 0$ if $0,m = 0$, $n,0 = 0$, $0,0 = 0$

2) $0/1ks(n,m) = 0/1ks(n-1,m)$ if $w_n > m$.

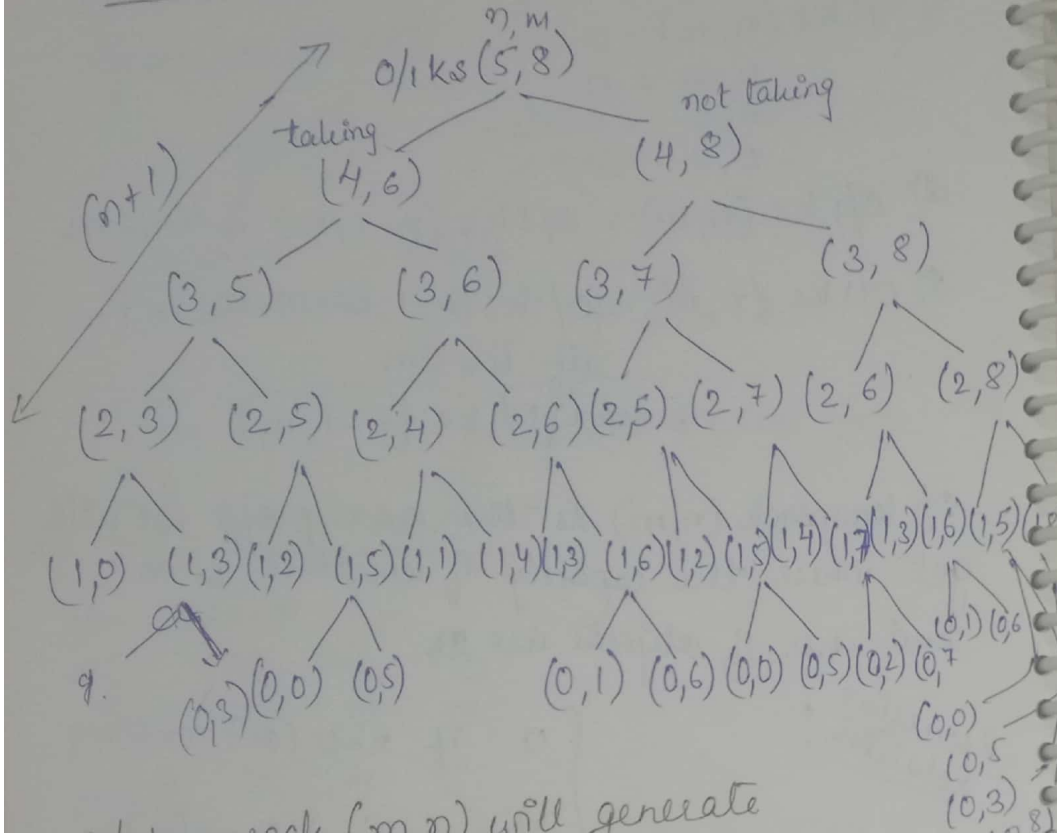
3) $0/1ks(n,m) = \begin{cases} 0/1ks(n-1, m-w_n) + P_n \\ \quad \text{if } w_n \le m \\ \max\{0/1ks(n-1,m)\end{cases}$

0/1 knapsack (n,m) is the max. profit we will get where the capacity of knapsack is m and no. of objects are n.

Recursive Relation:

$$0/1ks(n,m) = \begin{cases} 0 & \text{if } n=0 \text{ (or) } m=0 \\ 0/1ks(n-1,m) & \text{if } w[n] > m \\ \max\begin{cases} 0/1ks(n-1, m-w[n]) + P[n] \\ \qquad \text{if } w[n] \le \\ 0/1ks(n-1,m) \end{cases} \end{cases}$$

Recursive Tree



0/1 knapsack (m,n) will generate

⇓

n+1 — complete Binary Tree (upper bound)

⇓

2^{n+1} — 1 nodes.

⇓

2^n — function.

⇓

2^n * 1 - comparison

⇓

$2^n * O(1) \Rightarrow O(2^n)$ without Dynamic prog.

Here Structure is used to store the information.

Space:

i/p + extra

⇓

n + stack

⇓

n+1

O(n) without dynamic prog.

In the above recursive tree some (very less) function calls are repeating. because of this reason we are going for dynamic prog. which will call only distinct func call

imp How many distinct func.n call are there in 0/1 knapsack? with Dynamic prog.

Ans 0/1 knapsack (n, m)

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

(n+1) (n+1)

(n+1).(n+1) — Distinct func. call

⇓

mn — Distinct Func call.

⇓

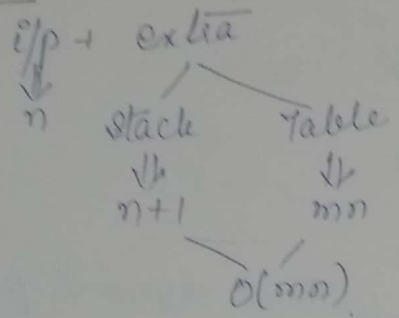
mn * O(1)

⇓

O(mn)

Time:

Space: (with Dynamic Prog)



In 0/1 knapsack problem because of very less repeatations time complexity O(mn) is almost equal to $O(2^n)$ so, it is one of the N-P complete problem.

problem taking more the polynomial time is called NP complete.

Note-2

Sum of subset problem is polynomialy reducable to 0/1 knapsack pro

Note-2

0/1 knapsack is polynomial time reduce-able to sum of subset. So sum of subset is also N.P complete problem.

(If one problem is reducable to another type of problem means that they are similar)

MATRIX CHAIN MULTIPLICATION

matrices

Ex: $A_{2\times4}$ $B_{4\times3}$

$C_{2\times3} = A_{2\times4} \times B_{4\times3}$

$A = \begin{bmatrix} - & - & - & - \\ - & - & - & - \end{bmatrix}_{2\times4}$ $B \begin{bmatrix} - & - & - \\ - & - & - \\ - & - & - \\ - & - & - \end{bmatrix}_{4\times3}$.

multiplication

$C \begin{bmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \end{bmatrix}_{2\times3} \Rightarrow$ 2×3 elements

each element ⇓ multiplication

2 * 3 * 4 ⇓

24 multiplication

Ex 2

$A_{10\times2}$ $B_{2\times5}$

C = AB

⇓

10×5 element

⇓

10×5×2 multiplications

= 100 mul

Ex 3

$A_{2\times4}$ $B_{4\times5}$ $C_{5\times3}$.

$AB_{2\times5} \times C_{5\times3} \Rightarrow ABC_{2\times3}$

2×5×4 + 2×5×3.

40 + 30 = 70 multiplication.

ex 4

$A_{2\times3}\ B_{3\times5}\ C_{5\times2}\ D_{2\times4}$

$E = ABCD$

$((AB)C)D$

$AB_{2\times5} = 2\times5\times3 = 30$ mul

$(ABC)_{2\times2} = 2\times5\times2 = 20$ mul.

$(ABCD)_{2\times4} = 2\times2\times4 = 16$ mul

66 mul.

$(A(B(CD)))$

$(CD)_{5\times4} = 5\times4\times2 = 40$ mul

$(BCD)_{3\times4} = 3\times4\times5 = 60$ mul.

$(ABCD)_{2\times4} = 2\times4\times3 = 24$ mul

124 mul.

$(A((BC)D))$

$BC_{3\times2} = 3\times5\times2 = 30$ mul

$BCD_{3\times4} = 3\times2\times4 = 24$ mul.

$ABCD_{2\times4} = 2\times3\times4 = 24$ mul

78 mul.

$((A(BC))D)$

$BC_{3\times2} = 3\times2\times5 = 30$ mul

$ABC_{2\times2} = 2\times3\times2 = 12$ mul.

$ABCD_{2\times4} = 2\times2\times4 = 16$ mul.

~~58~~ 58 mul

$((AB)(CD))$

$AB_{2\times5} = 2\times3\times5 = 30$ mul

$CD_{5\times4} = 5\times2\times4 = 40$ mul.

$ABCD_{2\times4} = 2\times5\times4 = 40$ mul

110 mul.

ex 5

$A_{1\times2}\ B_{2\times1}\ C_{1\times3}\ D_{3\times1}$ ABCD

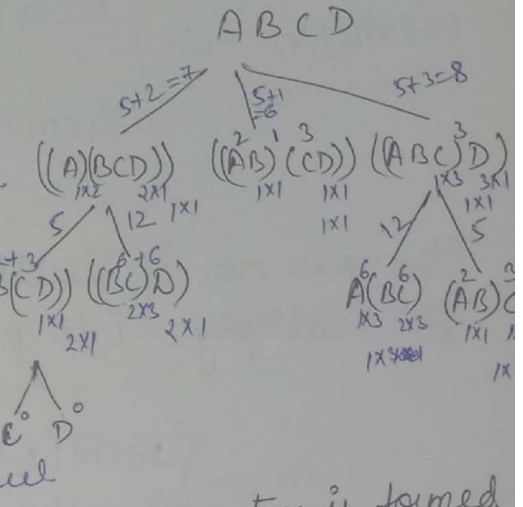
$((AB)C)D$

$AB_{1\times1} = 1\times2\times1 = 2$

$ABC_{1\times3} = 1\times3\times1 = 3$

$ABCD_{1\times1} = 1\times1\times3 = 3$

8 mul



$(A(BC))D$

$BC_{2\times3} = 2\times3\times1 = 6$

$ABC_{1\times3} = 1\times3\times2 = 6$

$ABCD_{1\times1} = 1\times1\times3 = 3$

15 mul

$((AB)(CD))$

$AB_{1\times1} = 1\times2\times1 = 2$

$CD_{1\times1} = 1\times3\times1 = 3$

$ABCD_{1\times1} = 1\times1\times1 = 1$

6 mul.

$(A((BC)D))$

$BC_{2\times3} = 2\times3\times1 = 6$

$BCD_{2\times1} = 2\times1\times3 = 6$

$ABCD_{1\times1} = 1\times1\times2 = 2$

14 mul

$A(B(CD))$

$CD_{1\times1} = 1\times3\times1 = 3$

$BCD_{2\times1} = 1\times2\times1 = 2$

$ABCD_{1\times1} = 1\times1\times2 = 2$

7 mul

Multiplication of 4 matrices (ABCD) $\rightarrow P_1 * P_4$

$$MCM(1,4) = \min \begin{cases} MCM(1,1) + MCM(2,4) + P_0 * P_1 * P_4 \\ MCM(1,2) + MCM(3,4) + P_0 * P_2 * P_4 \\ MCM(1,3) + MCM(4,4) + P_0 * P_3 * P_4 \end{cases}$$

($P_0 * P_1$, $P_1 * P_2$, $P_3 * P_4$; $P_0 * P_2$, $P_2 * P_4$)

The min no. of multiplications required to multiply i to j matices :-

$$MCM(i,j) = \begin{cases} 0 & \text{if } (i=j) \\ \min_{i \le k \le j-1} \begin{cases} MCM(i,k) + MCM(k+1,j) + P_{i-1} * P_k * P_j \end{cases} & \text{if } i<j \end{cases}$$

($P_{i-1} * P_k$, $P_k * P_j$)

'MCM(1, n) will generate n-level n-array tree (upperbound)

Total func^n calls $= (n-1)(n-2)(n-3)\ldots 1$

$= (n-1)!$

Time complexity $= (n-1)! \times$ cost of 1 func call

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

$= n! = O(n^n)$

To find the min b/w 23 $\rightarrow$ 2-1 comparison

To find the min b/w 100 = 100-1 "

So % " " " " n = n-1

coz of this cost of 1 func call = $n-1$

$= O(n-1) \equiv O(n)$

Space : without

i/p + extra

n n level tree

n n

$O(n)$

imp

In the above recursive tree many func calls are repeating. so we will go to dynamic prog.

Ques How many distinct func calls are there in MCM(i, j)?

Soln MCM(i, j)

change n way change n ways

n^2 – DFC.

Time $= n^2 \times n = O(n^3)$

Space (with dynamic prog)

i/p + extra

i/p → n

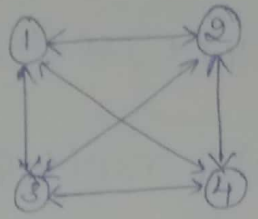
extra → stack → n

extra → Table → n^2 – Distinct Function call

$O(n^2)$

TRAVELLING SALESMAN PROBLEM

ex



	1	2	3	4
1	0	10	20	30
2	5	0	70	80
3	7	60	0	50
4	6	15	25	0

$$TSP(1,\{2,3,4\}) = \min \begin{cases} C(1,2)+TSP(2,\{3,4\}) \\ C(1,3)+TSP(3,\{2,4\}) \\ C(1,4)+TSP(4,\{2,3\}) \end{cases}$$

$$TSP(1,\{2,3,4\}) = \min \begin{cases} C(1,2)+C(2,3)+C(3,4)+C(4,1) \\ C(1,2)+C(2,4)+C(4,3)+C(3,1) \\ C(1,3)+C(3,2)+C(2,4)+C(4,1) \\ C(1,3)+C(3,4)+C(4,2)+C(2,1) \\ C(1,4)+C(4,2)+C(2,3)+C(3,1) \\ C(1,4)+C(4,3)+C(3,2)+C(2,1) \end{cases}$$

$$\min \begin{cases} 10+30+50+6 = 96 \\ 10+80+25+7 = 122 \\ 20+60+80+6 = 166 \\ 20+50+15+5 = 90 \\ 30+15+70+7 = 122 \\ 30+25+60+5 = 120 \end{cases}$$

$$TSP(2,\{3,4\}) = \min\begin{cases} C(2,3) + TSP(3,\{4\}) \\ C(2,4) + TSP(4,\{3\}) \end{cases}$$

$TSP(3,\{4\}) = C(3,4) + TSP(4,\phi)$

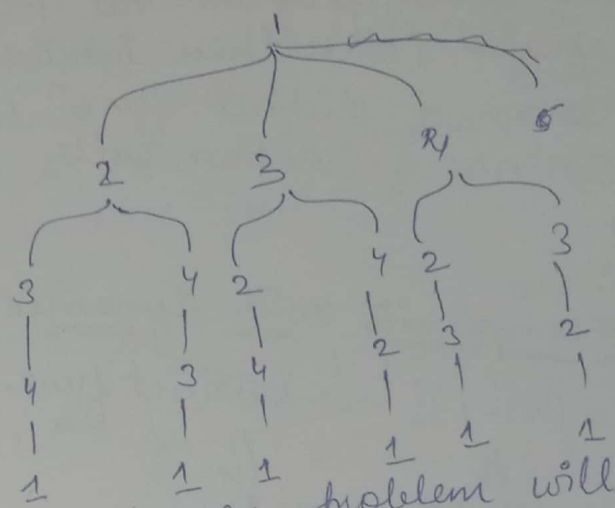
$TSP(4,\{3\}) = C(4,3) + TSP(3,\phi)$

$TSP(4,\phi) = C(4,1)$

$TSP(A,R)$ = Min. cost required to go from 'A' to all other remaining vertices in R exactly once and coming back to source S.

Recurrence Relation

$$TSP(A,R) = \begin{cases} C(A,S) \text{ if } R = \phi \\ \min\{C(A,K) + TSP(K,R-K)\} \\ \forall K \in R \end{cases}$$



Travelling salesman problem will create n-level n-array tree (upperbound)

Total no. of function calls = $(n-1)(n-2)\ldots 1$

$= (n-1)!$

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

$= n! \Rightarrow O(n^n)$

Space = i/p + extra

i/p ⇓ n^2 (matrix)

extra ↓ stack ⇒ n level

$O(n^2)$

In the above problem no. function call is repeated (other than terminations)
So no. of distinct func^n call = Total no. of function calls.

Time complexity with dynamic Prog

$(n-1)! \longrightarrow$ Distinct func. call.

$(n-1)! * n$

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

Space $\Rightarrow$ i/p + extra

i/p $\Rightarrow n^2$

extra $\Rightarrow$ stack + Table

stack $\Rightarrow n$

Table $\Rightarrow (n-1)! = O(n^n)$

no repeation so elements just storing but not used again

$\Rightarrow O(n^n)$.

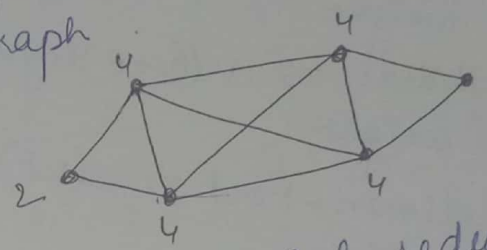
Note:

TSP is one of the NP-complete problem.

Hamiltonian graph/cycle is also a NP-complete problem. $\longrightarrow$ traversing every vertices edge of a graph exactly once & coming back to source

Euler graph $\longrightarrow$ cover every edge of a graph traversed exactly once and come back to source point

If a graph have only positive even (degree) edges. of every vertices. then it is definitly a eular graph

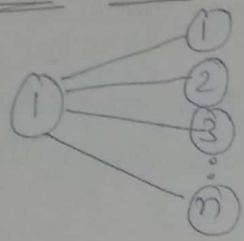


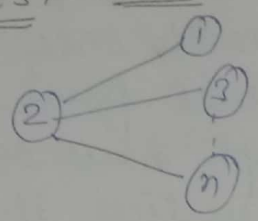
even degree edges.

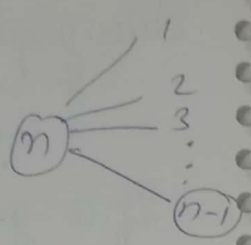
TSP is polynomial reduced to Hamiltonian graph. So checking Halmiltonian graph is a Hamiltonian graph

Checking given graph is Eular graph or not? $O(v^2)$ algo is possible so it is P-class problem.

6. ALL PAIRS SHORTEST PATH







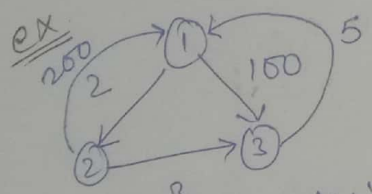
① +ve edge weight
$|V| *$ Dijkstra's $\Rightarrow V((V+E)\log V)$

② -ve edge weight
$|V| *$ Bellman-Ford $\Rightarrow V(VE)$

③ Unweighted Graph
$|V| *$ BFT $\Rightarrow V(V+E)$

④ Floyd warshall Algo
$\Rightarrow O(V^3)$ for +ve wt or for -ve wt

Floyd-Warshalls Algo



$\Rightarrow$

A^0 → directly.

A^0	1	2	3
1	0	2	100
2	200	0	3
3	5	∞	0

↑ no edge b/w (3→2)

3 → b/w 2 vertex, '1' no. of vertex

A'	1	2	3
1	0	2	100
2	200	0	3
3	5	7	0

$A'(3,2) = \min\{A^0(3,2), A^0(3,1)+A^0(1,2)\}$
$= \min\{\infty, 5+2=7\}$
$A'(3,2) = 7$

A^2	1	2	3
1	0	2	5
2	200	0	3
3	5	7	0

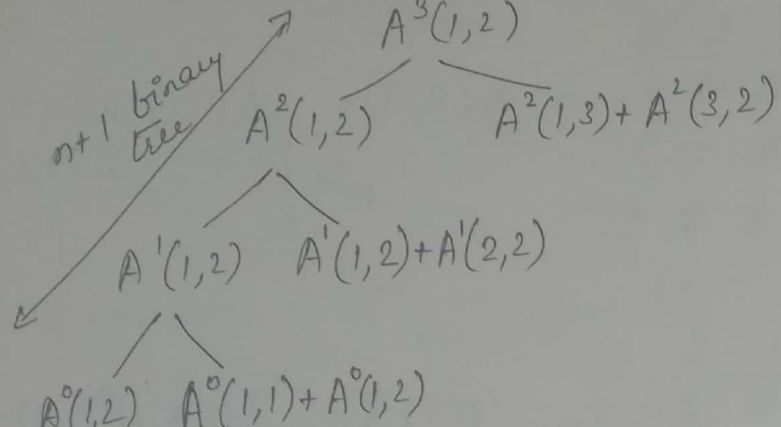
$A^2(2,3) = \min\{A^1(2,3), A^1(2,1)+A^1(1,3)\}$

A^3	1	2	3
1	0	2	5
2	8	0	3
3	5	7	0

$A^3(2,3) = \min\{A^2(2,3), A^2(2,3)+A^2(3,1)\}$
$= \min\{\underbrace{200}_{A^2(2,3)}, \underbrace{5+3}_{A^2(2,3)+A^2(3,1)}\}$
$= 8$

Let $A^k(i,j)$ = the min. cost required to go from vertex i to vertex j to which all intermediate vertices present in the set $\{0 \ldots k\}$

$$A^k(i,j) = \begin{cases} 0 & \text{if } (i=j) \\ \min\{A^{k-1}(i,j),\ A^{k-1}(i,k)+A^{k-1}(k,j)\} & \end{cases}$$



Floyd Warshalls algo on n-vertices generates $(n+1)^{th}$ level binary tree Then no. of func? call

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

$= 2^n * O(1)$

Time $= O(2^n)$

Space = i/p + extra

n^2 , $n+1$ tree

$O(n^2)$

In the above recursive tree some repeatition is there so we will go for dynamic prog. which will call distinct func call.

Ques How many distinct func call are there in $A^k(i,j)$

Soln $A^k(i,j)$

n n n

n^3 Distinct func. calls.

$n^3 * O(1) = O(n^3)$

Space = i/p + extra

n^2 , stack n^2 , Table

n tables of size n^2

$= n \times n^2 = n^3$

$O(n^3)$

Note

In floyd warshell., instead of storing every time in the new table, store in the same table, with this modification

space = i/p + extra

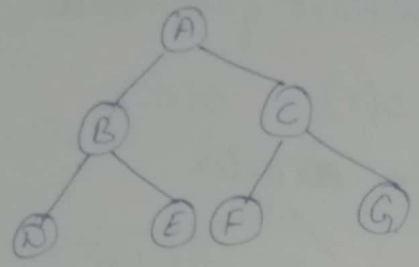
n^2 , only stack n

$O(n^2)$

$A^k(i,j) = \begin{cases} 0 & \text{if } i==j \\ \min\{A^0(i,j),\ A^0(i,k)+A^0(k,j)\} \end{cases}$

TREE TRAVERSAL & GRAPH TRAVERSAL

1. Preorder (Root, Left subtree, Right subtree)
2. Postorder (LST, RST, Root)
3. Inorder (LST, Root, RST)



Preorder : A B D E C F G

Postorder : D E B F G C A

Inorder : D B E A F C G

preorder (root) $\Longrightarrow T(n)$

```
{
    pf (root → data)              ⟹ O(1)
    preorder (root → left)        ⟹ T(n/2)
    preorder (root → right)       ⟹ T(n/2)
}
```

$$T(n) = 2T(n/2) + \text{Constant}$$
$$= O(n)$$

```
postorder (root)
{   postorder (root → left)
    postorder (root → right)
    pf (root → data)
}
```

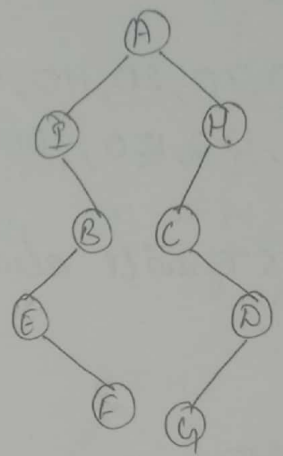
```
Inorder (root)
{   Inorder (root → left);
    pf (root → data);
    Inorder (root → right);
}
```

Note

Inorder, preorder and postorder on a binary tree takes $O(n)$ time always. (Best case, Worst case, Average case)

n nodes

Ques

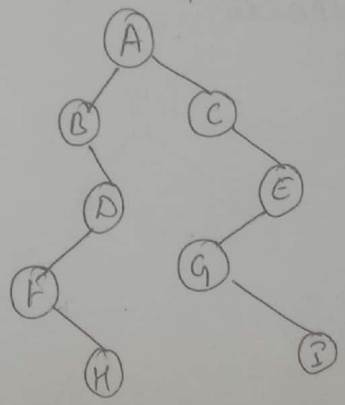


Inorder → I E F B A C G D H

preorder → A I B E F H C D G

postorder → F E B I G D C H A

Ques

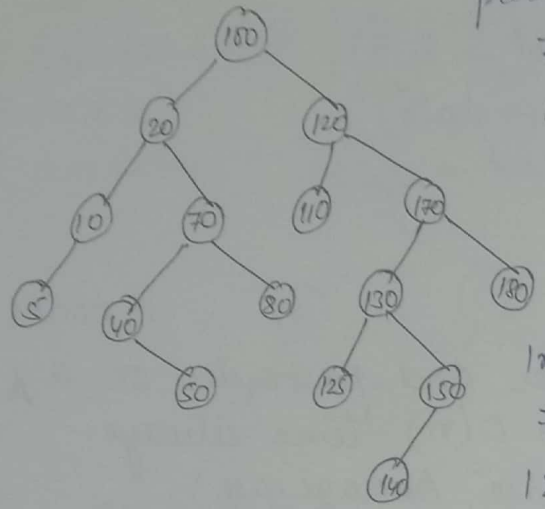


preorder = A B D F H C E G I

postorder = H F D B I G E C A

Inorder = B F H D A C G I E

(3)



preorder – 100, 20, 10, 5,
70, 40, 50, 80,
120, 110, 170, 130, 125,
150, 140, 180

Inorder = 5, 10, 20, 40, 50,
70, 80, 100, 110, 120,
125, 130, 140, 150, 170,
180

postorder = 5, 10, 50, 40, 80, 70, 20, 110, 125,
140, 150, 130, 180, 170, 120, 100

Note

* Inorder traversal of a BST will always produce ascending order.

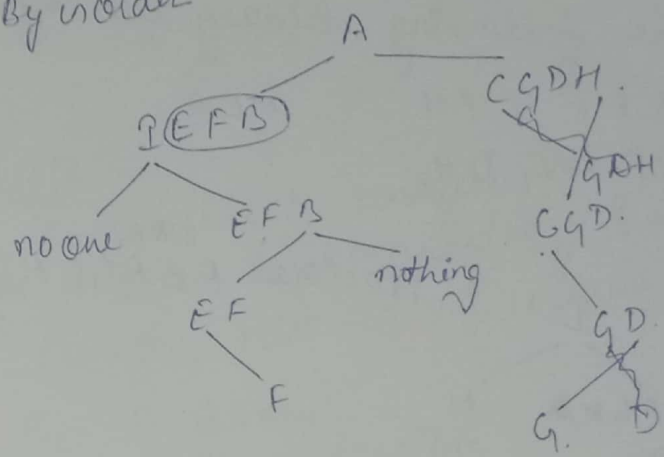
Left Root Right
small middle Greater

* BST with n nodes,

Q Consider the following Binary tree info?

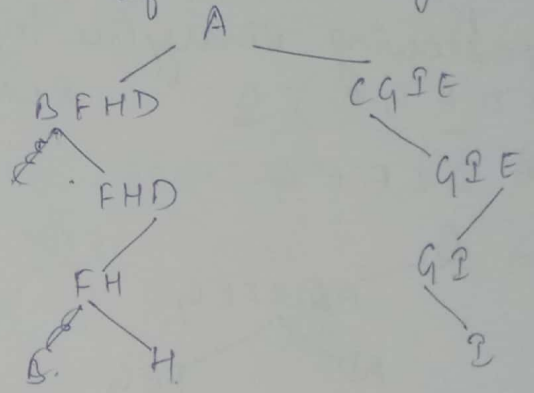
Preorder – (A) I B E F H C D G
Inorder – I E F B (A) C G D H.
Postorder ?

By inorder



Ans Post H F D B I G E C A — start from last

In B F H D A C G I E
left Right



Note

To construct binary tree for the given pre-order, post order and inorder: ~~Inord~~ $O(n^2)$ time is req. (n-times linear search for finding.

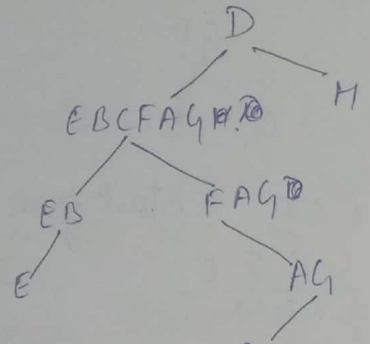
Q Consider the following binary tree info.

Pre: [D] C B E F G A H — NLR.

In: E B C F A G D H.

post order? — LRN

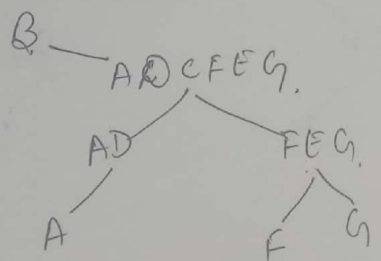
postordu E B A G C H D



Q Consider the following Binary tree inf. — RLN

Post: A D F G E C B

In: B A D C F E G.

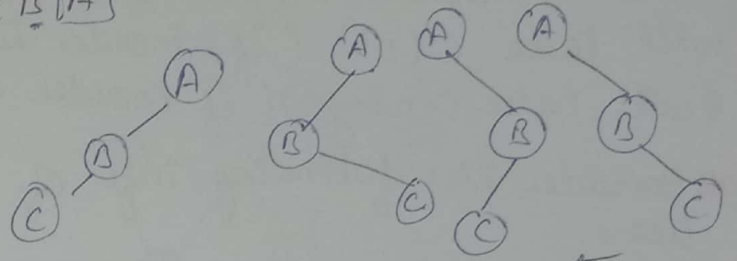


Ques Consider the following BT info.

Pre: [A] B C

post C B [A]

↳ many guess.



Note:

To construct unique binary tree.

~~Pre~~ In-order is compulsory.

Pre-in, Post-in } unique binary tree.

Pre, Post } Binary tree is possible but not unique

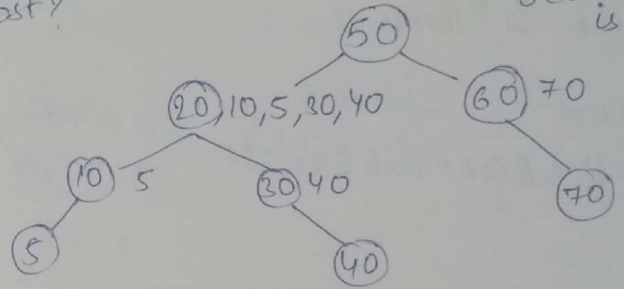
Ques Consider the following BST info.

Pre: 50 20 10 5 30 40 60 70

post?

because of BST inorder is ascending order

5 10 20 30 40 50 60 70



5, 10, 40, 30, 20, 70, 60, 50.

Note:

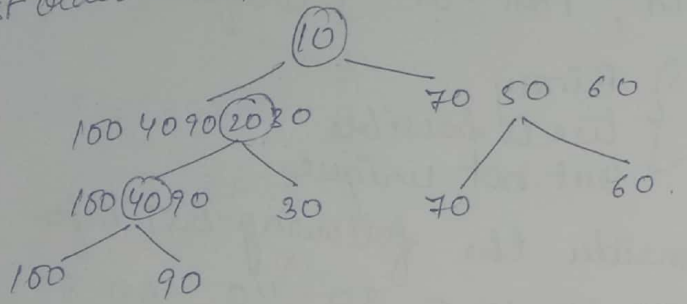
To construct binary tree for the given (preorder, Inorder) or (post-order, Inorder) will take $O(n^2)$ (if inorder is not sorted) & will take $O(n \log n)$ (if inorder is sorted)

Ques Consider the following info of min heap tree.

~~Preorder~~ – ~~10, 20, 40, 100, 90, 30, 50, 70, 60~~

Inorder – 100 40 90 20 30 10 70 50 60.

post order – 100, 90, 40, 30, 20, 70, 60, 50, 10



Note

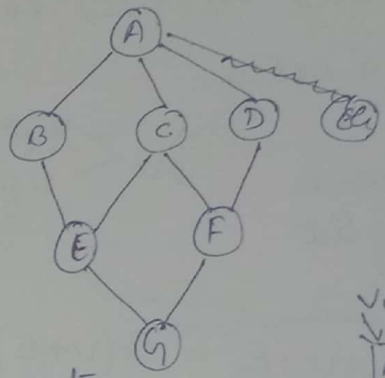
Inorder gives Left sub tree and Right sub tree if root node is known.

1) Finding root → n } $2n$ * ntimes.
2) Finding left & Right sub tree } n ⇒ $O(n^2)$.

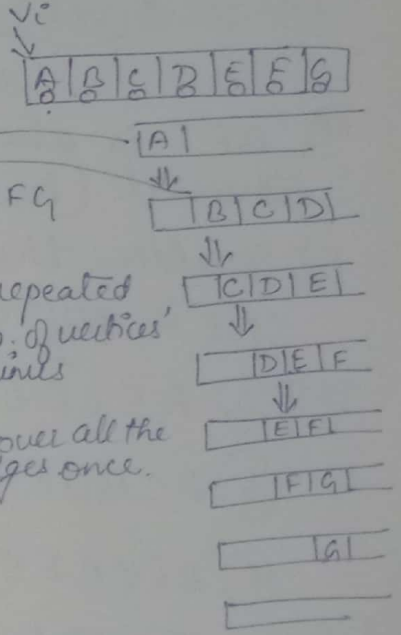
GRAPH TRAVERSAL

1) Breadth First Traversal & Depth First Traversal.

1) BFT




```
BFT(v)  → any vertex
{
  visited(v) = 1
  Add(v, Q)
  while (Q is not empty) }→ repeated 'no. of vertices' times
  {
    x = delete(Q)
    for all w adj to x. }→ cover all the edges once.
    {
      if (w is not visited)
      {
        visited(w) = 1
        add(w, Q);
      }
    }
  }
}
```



Note

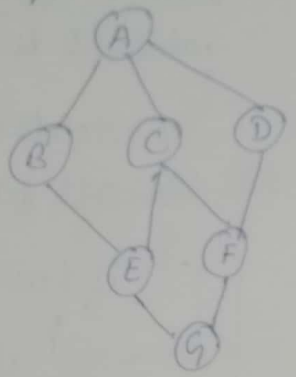
1. To implement BFT we are using 'Queue' Data structure
2. Time complexity = O(V+E) i.e. all cases [B C / W C / A C]
3. Space = i/p + extra
   - i/p: if graph is given as adjacency list
   - extra: 2V < V for array + V for queue

O(V+E)

3V+E = O(V+E)

4. BFT is also known as level-order traversal (level-by-level printing)
5. Consider the following graph

Give 4 diff" BFT's.



Sol^n (a) A B C D E F G

(b) A C B D E F G

(c) A D C B F E G

(d) A C B D F E G

(e) A C D B E F G

(f) C A E F B D G

(g) G E F B C D A

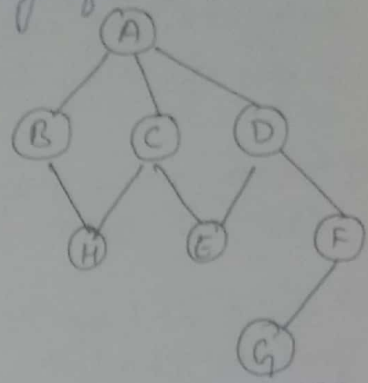
(h) E B G A F C D X

C

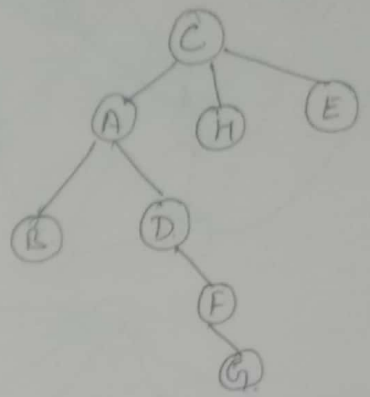
Ques Consider the following graph

Give 4 diff" BFT starting from G

1. G F D E A C B H
2. G F D A E B C H
3. G F D A E C B H
4. ~~G F D E A C H~~
5. ~~G F D A E A~~



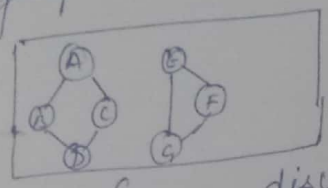
BFT (tree) is a spanning tree (covering all the vertices with (n-1) edges)



(BFT) of above graph

Applications of BFT

1. using BFT we can check a graph is connected or not
2. using BFT we can find out no. of connected components in the given graph.



G_1 disconnected

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

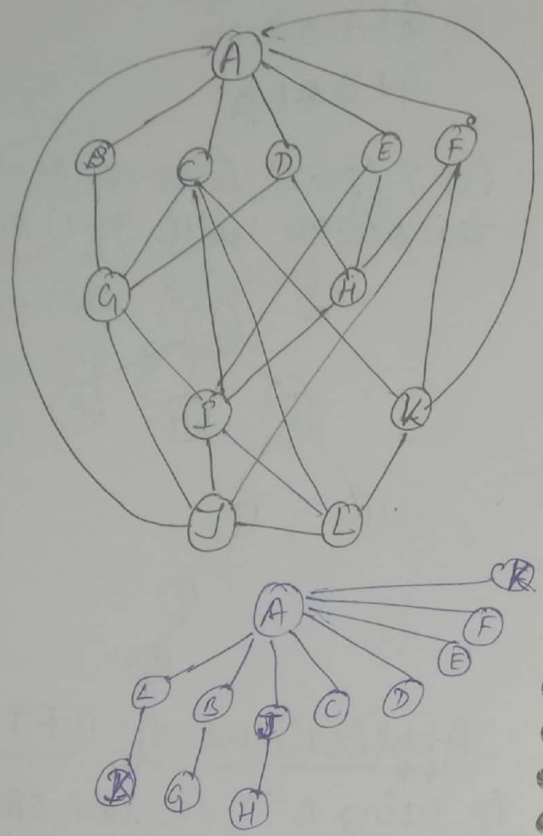
Applied BFT one time from

For the above example we applied BFT 4 times but the complexity will be (V+E) only coz only (V+E) are covered as a whole. and 1 can of away to judge the application of BFT.

3. For a given graph contain cycle or not we can use BFT. It is a P-class problem

4. Using BFT we can find out shortest path from the given source to every other vertex in the unweighted graph O(V+E)

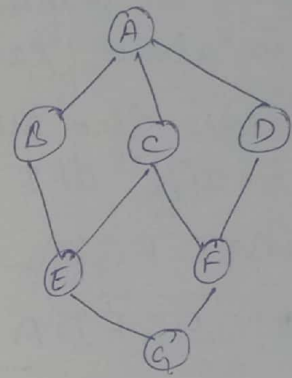
7. Using BFT we can verify a given graph is bi-partite graph or not



DEPTH FIRST TRAVERSAL

```
DFT (v)
{
1. visited (v) = 1
2. pf (v);
3. for all w adj to v
   {
     if (w is not visited)
     DFT (w);
   }
}.
```

each function call will occupy 1 stack place return of function call free the stack space



A

ø	ø	ø	0	ø	0	0
A	B	C	D	E	F	G

DFT(A)
1. ✓
2. A
3. w = B(1)
1. B

DFT(B)
1. ✓
3 w = AE
1. A ✓ E. DFT(E)

DFT(E)
1. ✓
3. w = BC G
1. B covered ✓
2. C DFT(C)

DFT(C)
1. ✓
3. w = AEF
1. A covered ✓
2. E covered ✓
3. F DFT(F)

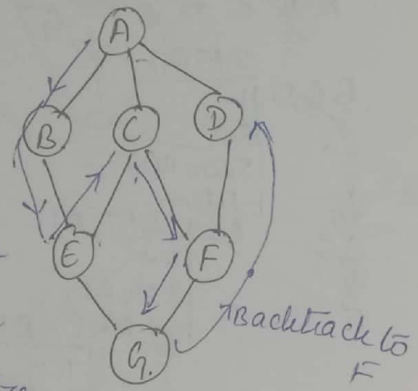
DFT(F)
1. ✓
3. w = C D G
1. C covered
2. D DFT(D)

DFT(D)
1. ✓
3. w = A, F
1. A covered
2. F covered

1% To implement DFT, we all using stack
2% For loop is covering all edge and recursion is covering all vertices so time complexity is $O(V+E)$ {all cases i.e. BC, WC, AC}
3% Space complexity = input + extra
input $(V+E)$
extra: stack size max 'V' ; Array of size V
$3V+E \equiv O(V+E)$

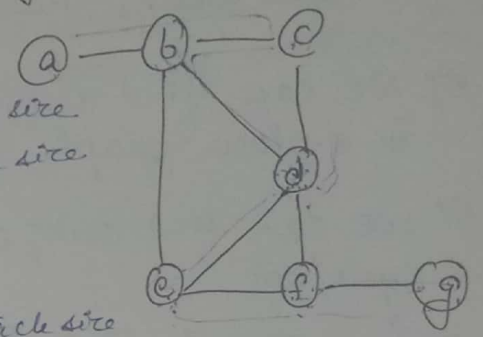
Max size of stack can be used = 'V' size
According to graph size of stack can vary.
Like is above case 'V-1' was the stack size

Ques Consider the graph.
4-diff^{n} DFT



1% A B E C F G D → 6 stack size.
2% ~~A B~~ C E G F D A B → 7 stack size
3% A D F G E C B. → 6 stack size
4% A B E G F D C → 6 stack size
backtrack
5% F C E G B A D → 6 stack size
backtrack
6% D F C A B E G

Ques Consider the following graph :-

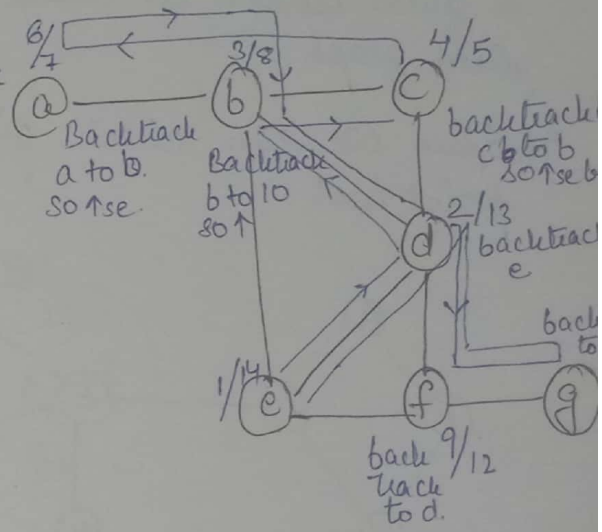


DFT (T/F)
B.T

1% e d b c a f g ✓ 4 stack size
2% g f e d b a c ✓ 6 stack size
3% b d c a f e g X
4% a b c d f g e ✓ 6 stack size
B.T.

whenever you backtrack ↑se a no.

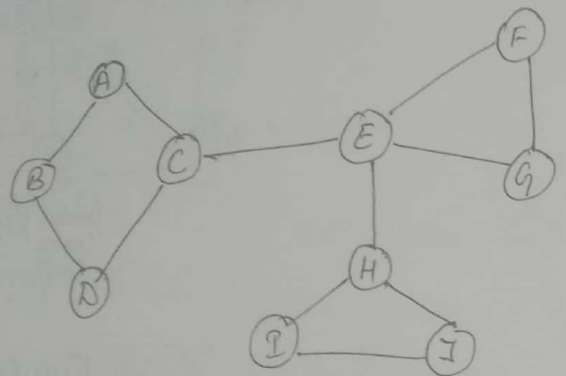
1% e d b c a f g (1/14, 10/11)



'1' starting time
14 finishing time

Application of DFT

1. we can verify given graph is connected/not
2. we can find out no. of connected components in a given graph
3. we can find out, a given graph contain a cycle/not
4. we cannot find out shortest path from a given vertex to every other vertex in a the given graph.
5. Given We can check given vertex is articulation point/not



articulation pt: C, E, H
deleting on which the graph becomes disconnected.

To check 'X' is articulation pt. or not
1. with 'X' on graph apply DFT & if all vertexs covered
2. then without 'X' apply DFT if all vertex are not covered (i.e. ~~their~~ respective values of some vertex remains zero then we came to know that that graph has becomed disconnected) and the 'X' was articulation pt.

6. we can check the given graph is strongly connected / not.

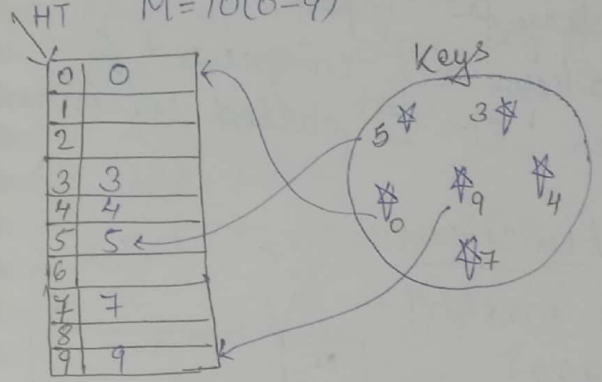
strongly connected — In Directed graph if there is path b/w every pair of vertices → strongly connected graph.

P, NP, NP

Hashing

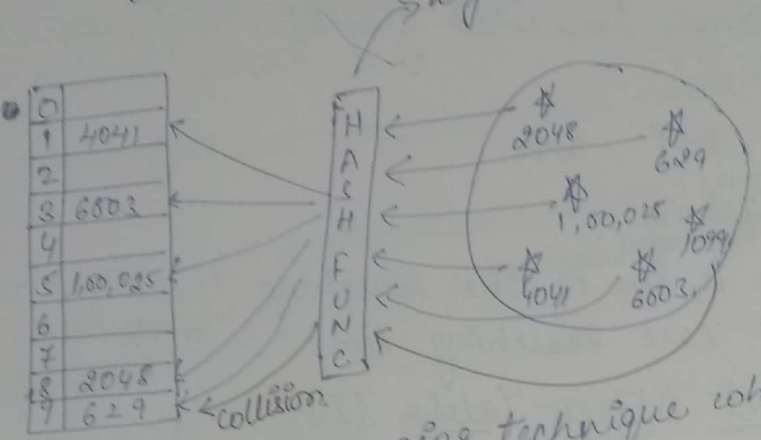
It is one of the searching technique where worst case searching time is O(1)

Direct Address Table (DAT)



(1) In direct adds table key itself is the address without any calculation
(2) worst case searching time is O(1)
(3) The drawback with the DAT is that even when though the no. of keys very less but one key may be very large (2^{1000}) then there

there is a need of Hash table approx. 2^{1000} i.e. $2^{1000}-1$. To eliminate this drawback we are going to the hash func.



Hash func is a mapping technique which takes very large key as input.

Collision – if two keys are compressed in same manner then it is called collision.

Types of Hash Functions

1. Divison modulo method –

M = 1000 (0-999)

Key = 13265487 9

$hf(k) = k \bmod m$

= 879

0	
1	
2	
⋮	
879	13265487 9
⋮	
999	

ex-2

$M = 8 (2^3)$

$2^3 \equiv 8$ → 0-7 remainder after dividing no. by 8 so to represent 0-7 → 3 bits are used to

$10101116000010101 \bmod 2^3 = 101$

ex-3

$m = 2^k$

$1150111160001010\underbrace{10010101}_{\text{suppose k bits}} \bmod 2^k = 10010101$

Note

1. Don't choose m values exactly powers of 2 because if $m = 2^k$ then hash func of keys = LSB 'k' bits always.
2. Pick 'm' value which is a prime and which is not close to powers of 2

511 → don't take coz it is close to $2^9 = 512$

1023 → don't " " " " " $2^{10} = 1024$

2. Digit Extraction Method (TRUNCATION METHOD)

M = 1000 (0-999)

K = 1 2 3 4 5 6 7 8 9
13 2 65 4 8 7 9

hf(k) = 249

0	
1	
2	
⋮	
249	13265487 9
⋮	
999	

3) MID-SQUARE METHOD

M = 1000 (0-999)

Key = 8492617

$(Key)^2 = (8492617)^2$

⇓

abcdefghijklm

↳ Take middle 3 digits

0	
1	
2	
3	
⋮	
ghi	8492617
⋮	
999	

4) Folding Method

(i) Fold ~~shifting~~ Boundary

M = 1000 (0-999)

key = 132654798

132
+ 798
930 index

0	
1	
2	
3	
⋮	
930	132654798
⋮	
999	

(ii) Fold shift Boundary

M = 1000 (0-999)

k = 132654798

132
654
+ 798
1584

158
+ 4
162

Either 158

0	
1	
2	
⋮	
162	132654798
⋮	
999	

COLLISION RESOLUTION TECHNIQUES

- chaining (outside)
- open addressing (inside)
  - linear probing
  - Quadratic probing
  - Double Hashing

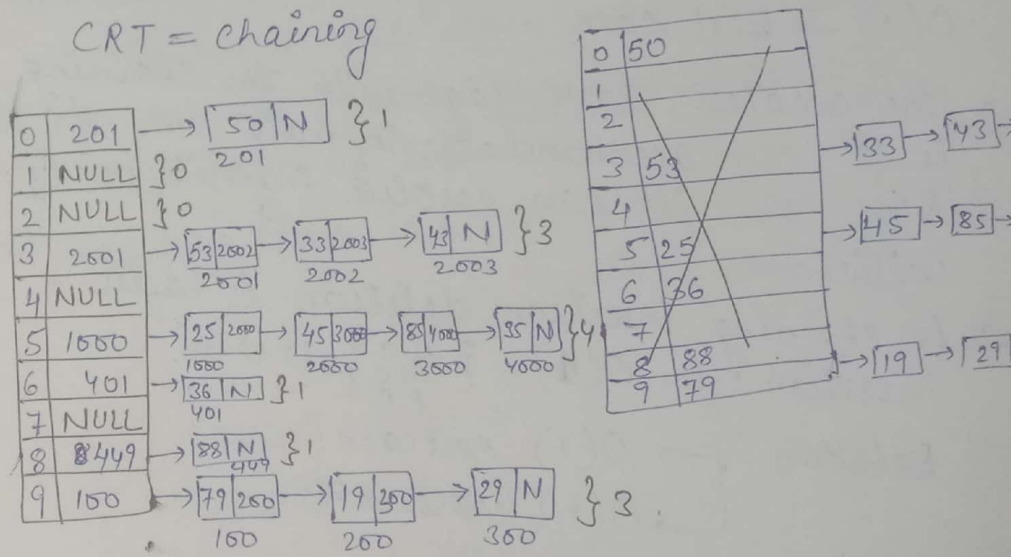
Chaining

ex: m = 10 (0-9)

hf(k) = k mod m

keys = 25, 79, 50, 45, 85, 53, 36, 19, 35, 29, 88, 33, 43

CRT = chaining



length of longest chain = 4

In the worst case, the length of longest chain of n elements = n.

Average chain length = $\frac{1+0+0+3+0+4+1+0+1+3}{10}$

$\frac{13}{10} = 1.3$.

Note

The lengths of the longest chain possible in worst case is 'n' because of this reason the worst case of searching is $O(n)$.

Drawback

The drawback with the chaining is we are wasting lot of space in the form of linked list even those place available inside the table.

$O(1) \rightarrow$ best case

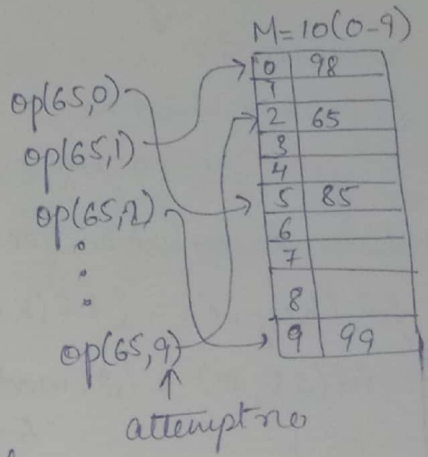
⇒ The greatest advantage with the chaining is it can accomodate infinite no. of keys. or it can resolve infinite no. of collisions.

⇒ In chaining insertion, deletion is easier.
Insertion is $O(1)$ { B C, W C, A C } all cases.

Deletion → $O(1)$ Best case
→ $O(n)$ worst case.

Open Addressing

1. If I hash to a particular slot & if it is already full then rehash again with diff^n hash func. (same as hash func^n with diff^n variable) until an empty slot is found



Continue this step max 'm' times.

2. we are not wasting space unnecessarily in the form of link-list.
3. Searching time is $O(n)$ max (worst case)
" " " $O(1)$ min (Best case)

m slots n-keys.

m-slots = n-keys.

1 slot = $\frac{n}{m}$ keys/slot

(load factor (α))

$0 \leq \alpha \leq 1$ [open addressing]

$0 \leq \alpha \leq \infty$ [chaining

Linear Probing

ex: M=10(0-9)

$hf(x) = x \bmod m$

keys = 25, 79, 43, 55, 67, 90, 143, 78, 95

Collision Resolution Technique = ~~LP~~ Linear Probing

$hf(key, i) = (hf(key) + i) \bmod m$

① $i=0$ $hf(25,0) = (25 \bmod 10 + 0) \bmod m$

$= 5+0 = 5^{th}$ ~~index~~

$= (5+0) \bmod 10$

$= 5^{th}$ index

$\forall i \in \{0, 1, 2, \ldots m-1\}$

soln LP(25,0)

(i) $(25 \bmod 10 + i) \bmod 10$

$(5+0) \bmod 10$

$= 5$

(ii) LP(79,0)

$(79 \bmod 10 + i) \bmod 10$

$(9+0) \bmod 10 = 9$

(iii) $(43 \bmod 10 + i) \bmod 10$

$(3+0) \bmod 10 = 3$

(iv) $(55 \bmod 10 + i) \bmod 10$

(a) LP(55,0) $5+0 \bmod 10 = 5^{th}$ is full

so (b) LP(55,1) $(5+1) \bmod 10$

$6 \bmod 10 = 6$

0	90
1	95
2	
3	43
4	143
5	25
6	55
7	67
8	78
9	79

(v) $(67 \bmod 10 + i) \bmod 10$

$= (7+0) \bmod 10$

$= 7$

$(143 \bmod 10 + i) \bmod 10$

$(3+0) \bmod 10$

3 is full

$3+1 \bmod 10 = 4 \bmod 10$

$= 4$

1) we will attempt (probe) one-by-one slot in the sequence.

2) Worst case searching time O(n)

best case " " O(1)

3) Deletion difficult - but we can manage with the special symbol '$'. If more '$'s then rehash the table again.

Quadratic Probing

ex:

$m = 10(0-9)$

$hf(x) = x \bmod m$

keys = 25, 79, 43, 55, 67, 90, 143, 78, 95

CRT = Quadratic Probing $c_1=1$ $c_2=1$

$QP(key, i) = (hf(key) + c_1 i + c_2 i^2) \bmod m$

$c + bx + ax^2$, $a=2$

$\forall i \in \{0, 1, 2, \ldots m-1\}$.

soln

$QP(25,0) = 5 + 1.0 + 1.0^2 = 5$

$QP(79,0) = 9 + 1.0 + 1.0^2 = 9$

$QP(43,0) = 3 + 1.0 + 1.0^2 = 3$

$QP(55,0) = 5 + 1.0 + 1.0^2 = 5$ full

$5 + 1.1 + 1.1^2 = 7$

0	90
1	95
2	
3	43
4	
5	25
6	
7	55
8	78
9	79

$QP(67,0) = 7 + 1.0 + 1.0^2 = 7$ full

$QP(67,1) = 7 + 1.1 + 1.1^2 = 9$ full

$QP(67,2) = 7 + 1.2 + 1.2^2 = 13 \bmod 10 = 3$ full

$QP(67,3) = 7 + 1.3 + 1.3^2 = 29 \bmod 10 = 9$ full

$QP(67,4) = 7 + 1.4 + 1.4^2 = 7 + 4 + 16 = 27 \bmod 10 = 7$ full

$QP(67,5) = 7 + 1.5 + 1.5^2 = 7 + 5 + 25 = 37 \bmod 10 = 7$ full

$QP(67,6) = 7 + 1.6 + 1.6^2 = 7 + 6 + 36 = 49 \bmod 10 = 9$ full

$QP(67,8) = 7 + 1.8 + 1.8^2 = 7 + 8 + 64 = 79 \bmod 10 = 9$ full

$QP(67,9) = 7 + 1.9 + 1.9^2 = 7 + 9 + 81 = 97 \bmod 10 = 7$ full

67 cannot be stored.

If we are probing in quadratic equation manner

2. worst case searching time $= O(m)$

best case " " $= O(1)$

3. Deletion is ~~deletion~~ difficult but we can manage with help of '$'. If more dollars rehash it.

Double Hashing

ex: $M = 10(0\text{-}9)$

$hf(x) = x \bmod m$

keys = 25, 79, 43, 55, 67, 90, 143, 78, 95

CRT = Double Hashing.

$$DH(key, i) = (\{hf_1(key) + i \cdot hf_2(key)\}) \bmod m$$

$\forall i \in \{0, 1, 2, \ldots m-1\}$.

$hf_2(key) = 1 + (key \bmod (m-2))$

Soln $DH(25,0) = 5 + 0 \cdot hf_2(25)$

$= 5$

$DH(79,0) = 9 + 0 \cdot hf_2(79)$

$= 9$

$DH(43,0) = 3 + 0 \cdot hf_2(43)$

$= 3.$

$DH(55,0) = h_1(55) + 0 \cdot h_2 55$

$= 5$ (full)

$DH(55,1) = 5 + 1.(55 \bmod 8 + 1)$

$= 5 + 1.(7+1) = (5+8) \bmod 10 = 3.$

$DH(55,2) = 5 + 2.8 = (5+16) \bmod 10 = 1$

$DH(67,0) = 7 + 0.hf_2(67)$

$= 7^{th}$ place

$DH(90,0) = 90 + 0.hf_2(90)$

$= 0$

0	90
1	55
2	
3	43
4	
5	25
6	
7	67
8	
9	79

$DH(143,0) = 3 + 0.hf_2 = 3$ full

$DH(143,1) = [3 + 1.(143 \bmod 8 + 1)] \bmod 10$

$= 3 + 1.1(7+1)$

$= 11 \bmod 10$

$= 1$

$DH(143,2) = [3 + 2.(143 \bmod 8 + 1)] \bmod 10$

$= 3 + 2.(8)$

$= 3 + 16 = 19 \bmod 10 = 9$

$DH(143,3) = 3 + 3.8$

$= 3 + 24 = 27 \bmod 10 = 7$

$DH(143,4) = 3 + 4.8$

$= 3 + 32 = 35 \bmod 10 = 5$

ex

$M = 10\ (0-9)$

$hf(k) = k \bmod m$.

keys = 54, 70, 93, 82, 71, 10, 20

CRT = L.P

$L.P = 54 \bmod 10 = 4$

$70 \bmod 10 = 0$

$L.P = 10 \bmod 10 = 0$

started from '0' $0+0=0$, $0+1=1$, $0+2=2$, $0+3=3$, $0+4=4$, $0+5=5$ done 6 attempts

$L.P = 20 \bmod 10 = 0$

started from '0' $0+0=0$, $0+1=1$, $0+2=2$, $0+3=3$, $0+4=4$, $0+5=5$, $0+6=6$ done 7 attempts

0	70
1	71
2	82
3	93
4	54
5	10
6	20
7	
8	
9	

Primary clustering

If two keys started from same Hash address. then those two key follow the same path unnecessarily to find an empty slot. in linear manner.

Because of this reason average searching time uses. This problem is known as. primary clustering

eg: In above example, '10' did 6 attempts and after '20' did '6 + 1 inertia' ie, 7 attempts

1. Linear probing is suffering with primary clustering

2. Average searching time of linear probing

$$= \frac{1+2+3+4+5+\ldots+m}{m} = \frac{\frac{m(m+1)}{2}}{m} = \frac{m+1}{2} = O(m)$$

Quadratic probing

ex: M = 10 (0-9)

h(k) = k mod m

keys 54, 70, 93, 82, 71, 10, 20

CRT = Quadratic probing

QP(10,0) = 0 + 1.0 + 1.0² = 0

QP(10,1) = 0 + 1.1 + 1.1² = 2

QP(10,2) = 0 + 1.2 + 1.2² = 6

0	70
1	71
2	82
3	93
4	54
5	
6	10
7	
8	
9	

QP(20,0) = 0 + 1.0 + 1.0² = 0

QP(20,1) 0 + 1.1 + 1.1² = 2

QP(20,2) 0 + 1.2 + 1.2² = 6

0 + 1.3 + 1.3² = 12 mod 10 = 2.

0 + 1.4 + 1.4² = (4 + 16) mod 10 = 20 = 0

0 + 1.5 + 1.5² = (5 + 25) mod 10 = 30 = 0

0 + 1.6 + 1.6² = (6 + 36) mod 10 = 42

⋮

⋮

Secondary clustering

1. If two keys are started from same hash address. They both follow the same path unnecessarily to find an empty slot in Quadratic manner.

Because of this address searching time is used. (less than the linear probing.

this avg searching time)

This problem is known as secondary clustering

2. Quadratic probing is suffering from secondary clustering.

3) Avg searching time = O(m)

[mathematically decreased but order of n]

Ques ex: M = 10 (0-9)

$h_2(k) = 1 + (k \bmod (m-2))$

$h_1(k) = k \bmod m$

keys = 54, 70, 93, 82, 71, 10, 20

0	70
1	71
2	82
3	93
4	54
5	20
6	10
7	
8	
9	

DH(10,0) = $h_1(10) + 0.h_2(10)$

(10,0) = 0 + 0.3 = 0

(10,1) = 0 + 1.3 = 3

(10,2) = 0 + 2.3 = 6.

DH(20,0) = $h_1(20) + 0.h_2(20)$ (20,1) = 0 + 1.5 = 5

(20,1) 10 + 0.5 = 0

$DH(30,0) = 0 + 0.7$

1. Avg case of double hashing. searching time $= O(1)$ because all the keys will follow diff^n path.

2. In double hashing 99% clustering is eliminated but 1% is still left out.

Note:

1. Using perfect hashing we will get worst case searching time $O(1)$

For every slot a new hash func^n is used. So for a table of m partitions (m+1) hash function (1 for a whole table and m for slots). So m hash tables are formed.

→ perfect Hashing.