14.04.22

Ques- What is algorithm?

In computer science for any problem we write programs for it, but before writing the programs we write some informal description of program which is known as algorithm.

For selection of better algorithm, we use the concept of space & time complexity of that particular algorithm.

To represent the space & time complexity of an algorithm we use asymptotic notation.

Needs of Asymptotic notations?

1) Effectiveness of problem
2) Correctness of programs
3) Compare the complexity
4) Making simple analysis

Q-
```
int fun1(int n)
{
  int m = 0;
for (int i = 0; i < n; i++)
  {
    m += 1;
  }
   return m;
}
```

Find T(n) & S(n).

Sol:

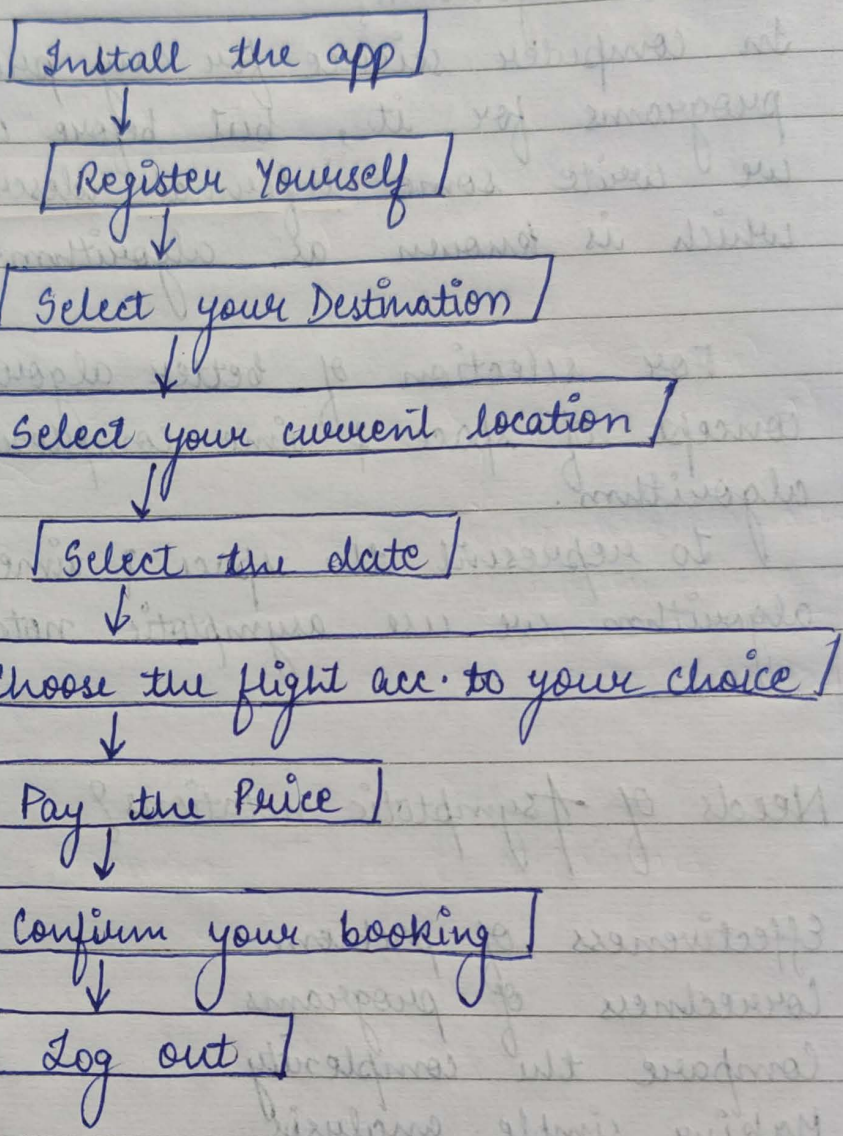
T(n) = O(n) [∵only one loop]

S(n) = constant O(1) [∵only one variable]

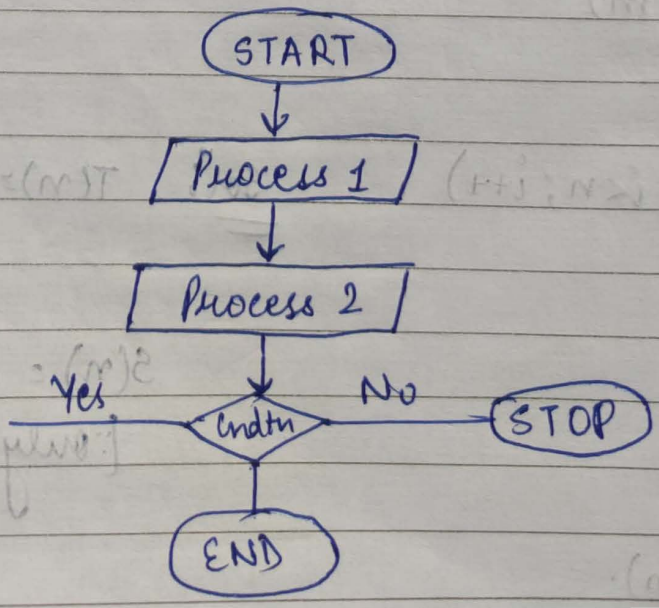
Page :
Date: / /

Ques- Draw a block diagram for flight ticket booking

Soln:



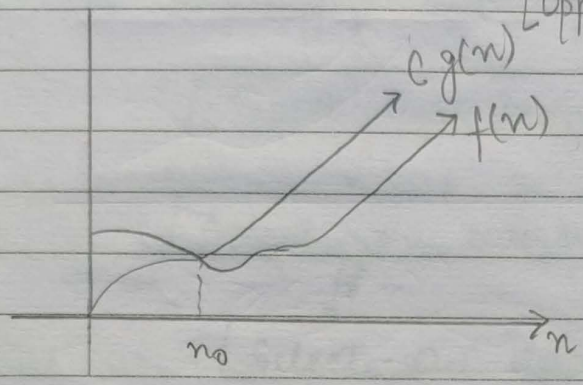
Flow Chart



16.04.22

ASYMPTOTIC NOTATIONS

1) Big - Oh (O) - Notation



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

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

[only when you satisfy this condition then only you can write this]

$n \geq n_0$, $c > 0$, $n_0 \geq 1$

★ For worst case

Ques- Represent the following f^n in O Notations

i) $f(n) = 3n+2$

$g(n) = n$

Sol^n:

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

$3n+2 \leq 1 \cdot n$ $\quad c = 1$

$3 \times 1 + 2 \leq 1 \cdot 1$ $\quad n = 1$

$5 \leq 1$ ✗

(If we take 1, 2, 3, then $3n+2$ becomes greater so take the closest upper bound) You can also take 5 here.)

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

$3 \times 2 + 2 \leq 4 \cdot 2$

$8 \leq 8$ ✓

$f(3n+2) = O(n)$

ii) $f(n) = 3n+2$

$g(n) = n^2$

Soln:- $3n+2 \leq c \cdot n$

$3\times1+2 \leq 1\cdot1$ $\quad c=1, n=1$

$5 \leq 1$ ✗

$3\times2+2 \leq 2\times2^2$ $\quad c=2, n=2$

$8 \leq 8$ ✓

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

iii) $f(n) = n$

$g(n) = 3n+2$

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

Soln:

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

$3+2 \leq 1$ $\quad c=1, n=1$

$5 \leq 1$ ✗

$3\times2+2 \leq 2\times2$ $\quad c=2, n=2$

$8 \leq 4$ ✗

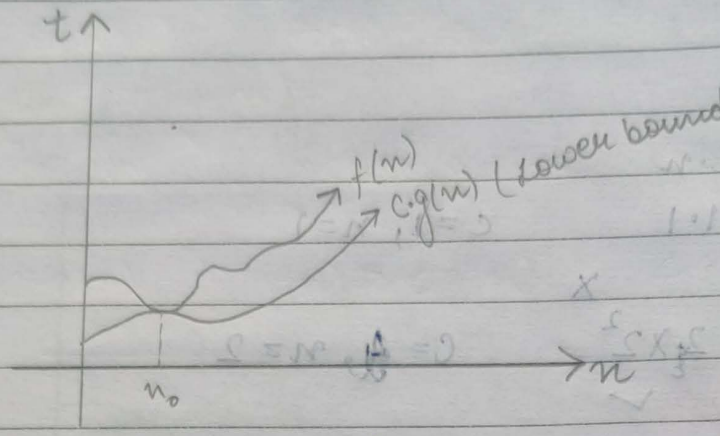
$3\times2+2 \leq 4\times2$ $\quad c=4, n=2$

$8 \leq 8$ ✗

$g(3n+2) = O(n)$

Page :
Date: / /

2) Omega (Ω) Notation



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

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

$c > 0$

$n \geq n_0$

$n_0 \geq 1$

★ **Best Case**

Ques. Represent the following function as $f(n) = \Omega(g(n))$

i) $f(n) = 3n+2$
$g(n) = n$

Sol^n:

$3n+2 \geq c \cdot n$

$3 \times 2 + 2 \geq 3 \cdot 2$ $\quad n=2, c=3$

$8 \geq 6$

in lower case keep min value

$f(3n+2) = \Omega(n)$

ii) $f(n) = 3n+2$
$g(n) = n^2$

Page :
Date: / /

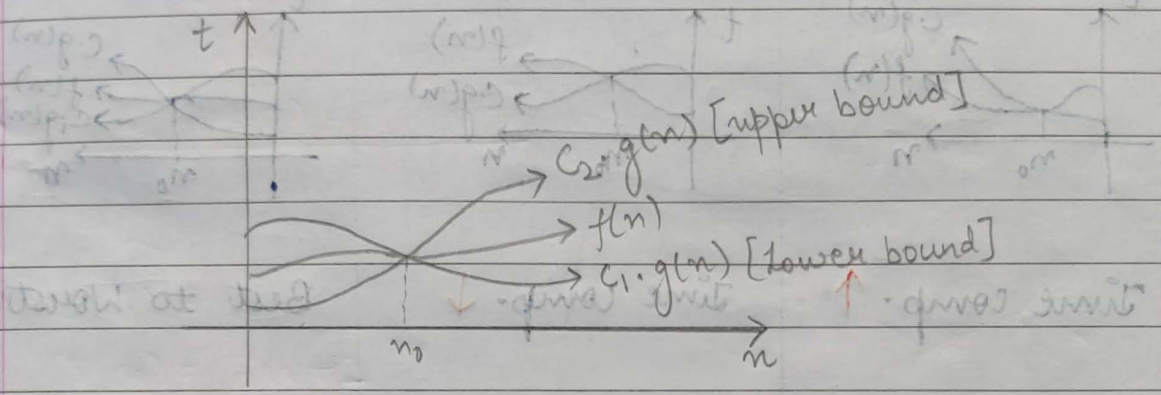
Sol^n: $3n+2 \geq c \cdot n^2$ OR $3n+2 \geq c \cdot n^2$

$3 \times 2 + 2 \geq 2 \cdot 2^2$ $\{c=2, n=2\}$ $3+2 \geq 1$

$8 \geq 8$ $5 \geq 1$

$\boxed{3n+2 = \Omega(n^2)}$

3) Theta (θ)- Notation :-



$C_1 g(n) \leq f(n) \leq C_2 g(n)$

$\boxed{f(n) = \theta(g(n))}$

$C_1, C_2 > 0$

$n_0 \geq 1$

$n \geq n_0$

★ For Average Case

Ques- $f(n) = 3n+2$ $g(n) = n$

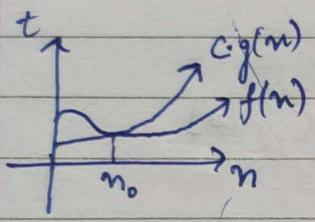
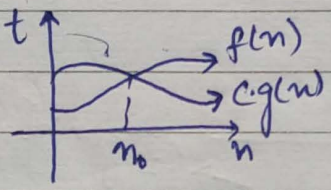
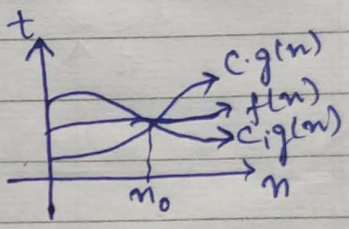
$C_1 \cdot n \leq 3n+2 \leq C_2 \cdot n$

1 2 4 2

$1 \cdot n \leq 3n+2 \leq 4n$ $\Rightarrow \boxed{f(3n+2) = \theta(n)}$

$1 \cdot 2 \leq 3(2)+2 \leq 4 \cdot 2$

$2 \leq 8 \leq 8$

	O-notation	Ω- notation	Θ- notation
1)	Worst case	Best case	Average case
2)	Upper bound	Lower bound	Upper to lower
3)	$f(n) \leq c \cdot g(n)$	$f(n) \geq c g(n)$	$c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$
4)			
5)	Time comp. ↑	Time comp. ↓	Best to Worst

21.04.22

Time-Complexity Order

1) Constant Time O(1)
Execution time of an algo is proportional to constant (independent to the I/P)
eg:- push and pop in stack.

2) Logarithmic Time Θ(log n)
Execution time of an algo is proportional to log time (divide input size into 2 or more than 2 way)
eg:- Binary search

3) Linear time $O(n)$

Execution time of an algo is directly proportional to the input size.

eg:- Find min/max in heap

4) n- $\log n$ time $(n + \log n)$

Execution time is directly proportional to product of input size and log part of input size.

eg:- Merge sort

5) Quadratic time $O(n^2)$

Execution time is directly proportional to square of the input size.

eg:- Bubble Sort

6) Polynomial time $O(n^k, k>1)$

Execution time is directly proportional to power of k to the input size.

eg:- Nested loops

7) Exponential Time $O(2^n)$

Used to finding all possible subsets of given elements.

8) Factorial time $O(n!)$

Used to find out all possible permutation of given elements.

Page :
Date: / /

Imp

$O(1)$
$O(\log n)$
$O(n)$
$O(n \log n)$
$O(n^2)$
$O(n^k, k>1)$
$O(2^n)$
$O(n!)$

Ques-1
```
int fun1 (int n)
{
  int m=0;
  for (int i=0; i<n; i++)
  {
    m+=1
  }
  return m;
}
```

Ans :- $T(n) = O(n)$

Ques-2
```
int fun2(int n){
  int i, j, m=0;
  for (i=0; i<n; i++)
  {
    for(j=0; j<n; j++)
      m+=1;
  }
}
```

Ans- $T(n) = O(n^2)$

Page :
Date: / /

Ques-3
```
A() {
  i=1;
for (i=1; i^2 <=n ; i++)
    print (CSW-2)
}
```

Ans- $i^2 \leq n$

$i \leq \sqrt{n}$

$T(n) = O(\sqrt{n})$

Ques-4
```
A()
{
for (i=1; i<n; i = i*2)
    print (CSW-2)
  }
```

→ i's no. se multiplied, it will become the base.

Ans- $T(n) = O(\log_2 n)$

Ques-5
```
A() {
for (i=1; i<n; i= i*5)
        print (CSW-2)
 }
```

Ans- $T(n) = O(\log_5 n)$

Ques-6
```
A() {
int i, j, k;
for (i=1; i<n; i++)          - n
  for (j=1; j<n; j*2)        - log2 n
    for (k=1; k<n; k=k*2)    - log2 n
      print (CSW-2)
}
```

Ans- $T(n) = O(n \log_2^2 n)$

Page :
Date: / /

Ques-7

```
A() {
int i, j, k;
for (i = n/2 ; i<n; i++)        n/2
  for (j = 1; j < n/2 ; j++)    n/2
    for (k = 1; k < n; k = k*2) log2n
      print (csw-2)
}
```

Ans- $T(n) = \frac{n}{2} \times \frac{n}{2} \times \log_2 n$

$= \frac{n^2}{4} \times \log_2^n$

$= O(n^2 \log_2^n)$

Ques-8

```
A() {
while (n>1)
  n = n/2 → jo no. hoga wahi log ka base.
}
```

Ans- $T(n) = O(\log_2^n)$

Ques-9

```
A(){
while (n>1)
  n = n/5
}
```

Ans- $T(n) = O(\log_5^n)$

Ques-10

```
A(int n) {
 int i, j, m = 0;
 for (i = 0 ; i<n; i++){
   for (j = i; j = 0; j--)
        m+=1;
 } return m;
}
```

Ans- $T(n) = O(n^2)$

Page :
Date: / /

Ques-11
```
A(int n)
{
  int i, j, k, m = 0;
  for (i = 0; i < n; i++){
    for (j = i; j < n; j++){
      for (k = j+1; k < n; k++){
        m += 1
    }
  }
}
```

Ans- $T(n) = O(n^3)$

Ques-12
```
A (int n)
{ int i, j = 0, m = 0;
  for (i = 0; i < n; i++){
    for ( ; j < n; j++){
      m += 1;
    }
  }
  return m;
}
```

Ans- $T(n) = O(n^2)$

Ques-13
```
int A(int n) {
  int i, j, m = 0;
  for (i = 1; i < n; i *= 2) {
    for(j = 0; j <= i; j++){
      m += 1;
    }
    return m;
  }
}
```

Ans :- $T(n) = O(n)$ //

22.04.22

Comparison b/w two functions

1) $n^2 < n^3$
2) $n^2 < 2^{n^2}$
3) $n > \log n$
4) $n^{2^n} > 2^{2^n}$
5) $n^2 > n \log n$
6) $n^{\log n} > n \log n$
7) $\sqrt{\log n} > \log \log n$

$n^2 \log n > n^2$

6) Taking $\log n$ on both sides

$\log n \log n > \log n + \log \log n$

Assume $n = 2^{2^{10}}$

$\log_2 2^{2^{10}} \quad \log_2 2^{10} \quad \log_2 2^{2^{10}} + \log_2 \log_2 2^{2^{10}}$

$2^{10} * 2^{10} \quad 2^{10} + \log_2 2^{10}$

$1024 * 1024 \quad 2^{10} + 10$

$1024 * 1024 \quad 1024 + 10$

$\therefore n^{\log n} > n \log n$

Ques-8 $f_1 = 2^n$

$f_2 = n^{3/2}$

$f_3 = n \log n$

$f_4 = n^{\log n}$

$f_3 < f_2 < f_4 < f_1$

Page :
Date: / /

Master's Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k \log^p n)$$

$a \geq 1$, $b > 1$, $k \geq 0$ & p is real number

i) if $a > b^k$ then $T(n) = \Theta\left(n^{\log_b^a}\right)$

ii) if $a = b^k$

(a) if $p > (-1)$ then $T(n) = \Theta\left(n^{\log_b^a} \log^{p+1} n\right)$

(b) if $p = (-1)$ then $T(n) = \Theta\left(n^{\log_b^a} \log \log n\right)$

(c) if $p < (-1)$ then $T(n) = \Theta\left(n^{\log_b^a}\right)$

iii) if $a < b^k$

(a) if $p \geq 0$ then $T(n) = \Theta(n^k \log^p n)$

(b) if $p < 0$ then $T(n) = \Theta(n^k)$

Q1 $T(n) = 6T\left(\frac{n}{3}\right) + n^2 \log n$

First check if it is in the form $T(n) = 6T\left(\frac{n}{3}\right) + \Theta(n^k \log^p n)$

After that check if $a \geq 1$, $b > 1$, $k \geq 0$ & p is real no.

$a = 6$
$b = 3$
$k = 2$

} Since it satisfies the condition

$b^k = 3^2 = 9$

$\therefore a < b^k$

Hence it is the case third and $p=1$ i.e. $p \geq 0$

$\therefore T(n) = \theta(n^k \log^p n)$

$$\boxed{T(n) = \theta(n^2 \log n)}$$

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

7) $T(n) = \sqrt{2}\,T\left(\frac{n}{2}\right) + \log n$

8) $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n$

9) $T(n) = 0.5T\left(\frac{n}{2}\right) + \frac{1}{n}$

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

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

$a = 2$, $b = 2$, $K = 1$, $p = -1$

$b^k = 2^1 = 2$

$a = b^k$

So, second case applies and $p = -1$

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

$$T(n) = \Theta\left(n^{\log_2^2} \log\log n\right)$$

$$\rightarrow \boxed{T(n) = \Theta(n \log\log n)}$$

7) $T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \log n$

$\sqrt{2} = 1.414$

$a = 1.414$, $b = 2$, $k = 0$, $p = 1$

$b^k = 2^0 = 1$

$a > b^k$

So, first case applies

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

$$\rightarrow T(n) = \Theta\left(n^{\log_2^{\sqrt{2}}}\right)$$

$$\rightarrow \boxed{T(n) = \Theta\left(n^{1/2}\right)}$$

[$\sqrt{2}$ can be written as $2^{1/2}$ so 2 and 2 in $\log_2^{\sqrt{2}}$ will get cancelled & power is left]

8) $T(n) = 64\, T\left(\frac{n}{8}\right) - n^2 \log n$

No solution because minus (−) is present. It should be always '+'.

9) $T(n) = 0.5\, T(n/2) + \frac{1}{n}$

No solution because a is less than 1

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

$a = 1, \; b = 2, \; K = 2, \; p = 0$

$b^K = 2^2 = 4$

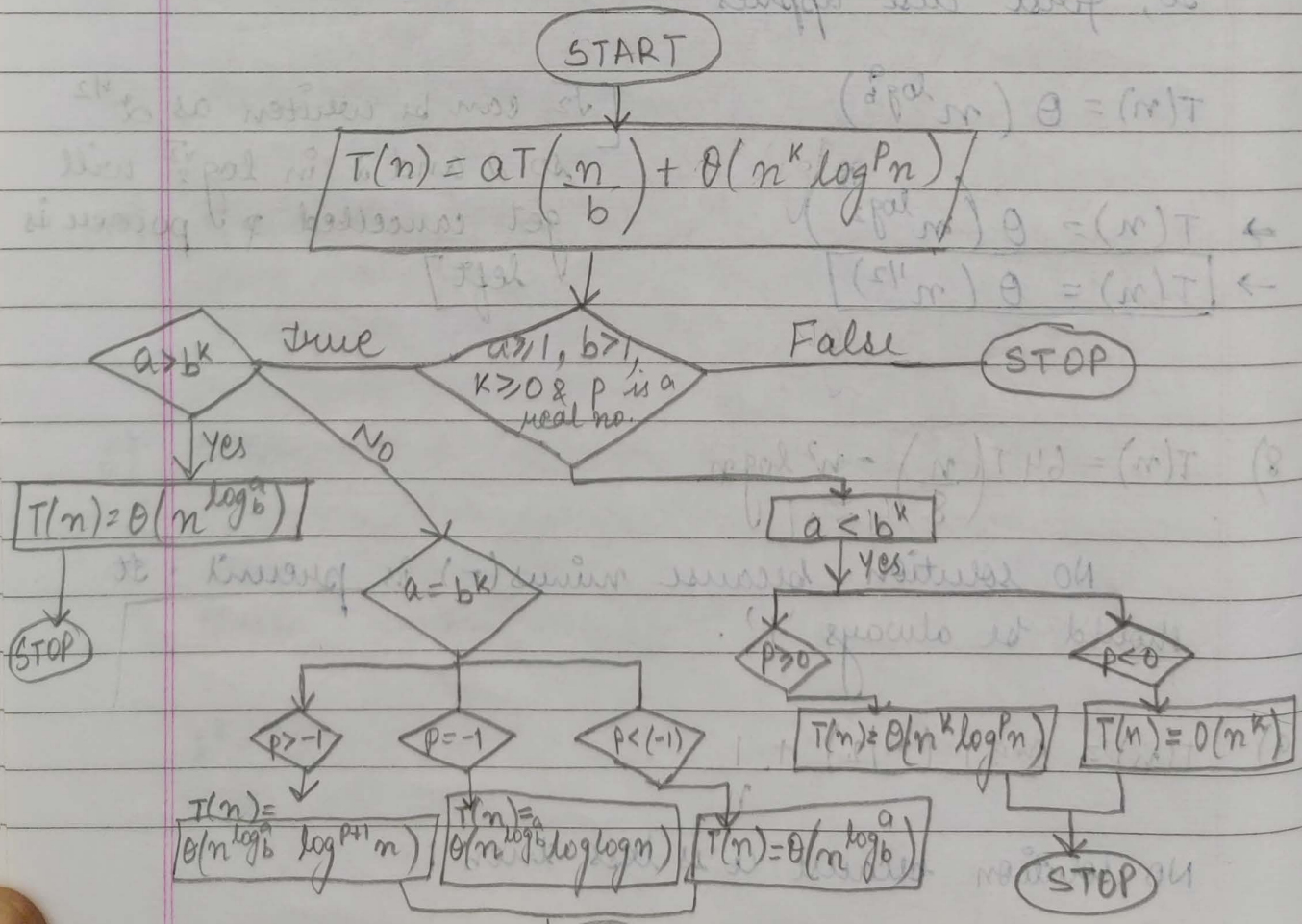
$a < b^K$

So, third case applies and $p = 0$

$T(n) = \theta(n^K \log^p n)$

→ $\boxed{T(n) = \theta(n^2)}$

Ques- Draw a flowchart of Master's Theorem



Page :
Date: / /

Approach to Solve Algorithm design Problems

→ Find out the constraints of the given problem
→ Develop idea to solve the problem
→ Analyze the complexity of the algorithm
→ Coding
→ Testing (if any error occured)

Q-1.22 $T(n) = 2T\left(\frac{n}{4}\right) + n^{0.5}$

$a = 2 \quad b = 4, \quad k = 0.5, \quad p = 0$

$b^k = 4^{0.5} = 2$

$a = b^k$

$p > -1$, second case applies

$T(n) = \theta(n^{\log_b^a} \log^{p+1} n)$

$T(n) = \theta(n^{\log_4^2} \log^1 n)$

→ $\boxed{T(n) = \theta(n^{0.5} \log n)}$

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

$a = 3, \quad b = 2, \quad k = 1, \quad p = 0$

$b^k = 2^1 = 2$

$a > b^k$

First case applies,

Page :
Date: / /

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

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

Back - Substitution Method

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

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

I) $a=1$ then $T(n) = O(n^{k+1})$

II) $a>1$ then $T(n) = O(a^n n^k)$

III) $a<1$ then $T(n) = O(n^k)$

Ques- Solve using back - substitution method

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

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

$a=1$, hence first case

→ $T(n) = O(n^2)$

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

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

Page :
Date: / /

a=1, Hence first case applies

→ $\boxed{T(n) = O(n^3)}$

Ques- Gift ques

```
A(n)
{
if (n >= 1)
{
  A(n-1),
Print ("Gift")
}
}
```

Total no. of recursive calls required for f(3). Find the space complexity.

A(3)
↳ A(2)
↳ A(1)
↳ A(0)

b) Space Complexity = O(n)
one variable

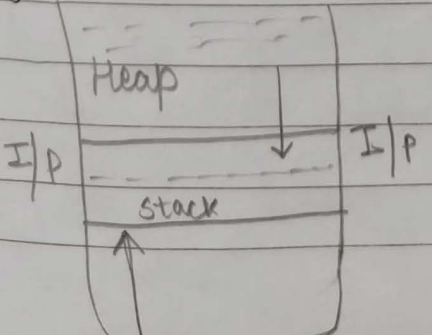
a) 4 times recursive call.

Space complexity for iterative program

Ques-

```
Algo (A, 1, n)
{
  int i;
for (i=1 to n)
  A[i] = 0;
}
```

Ans:- O(n)




```
Algo(A,1,n) → O(1)
{
  int i;
create(B,1,n) → O(n)
for(i=1 to n);
  A[i] = 0;
}
```

O(n) + O(1)

$\boxed{O(n)}$ is greater.

Page :
Date: / /

Ques-

```
Algo(A, 1, n)
{
  int i, j=10 ;
  for(i=1 to j)
  A[i]=0;
}
```

O(1)

Ques-

```
Algo(A, 1, n)
{
  Create B[n,n]
  int i, j;
  for(i=1 to n)
  for(j=1 to n)
  B[i,j] = A[i]
}
```

$O(n^2)$

Ques- Write a method that will return the sum of all the elements of the integer

[1, 2, 3, 4, 5]

RECAP

- → Definition of algorithm
- → Time complexity
- → Asymptotic Notation
- → Worst case, Best case, Avg case
- → Comparison
- → Time complexity order
- → Time complexity ques
- → Master's Theorem
- → Space Complexity
- → Back substitution

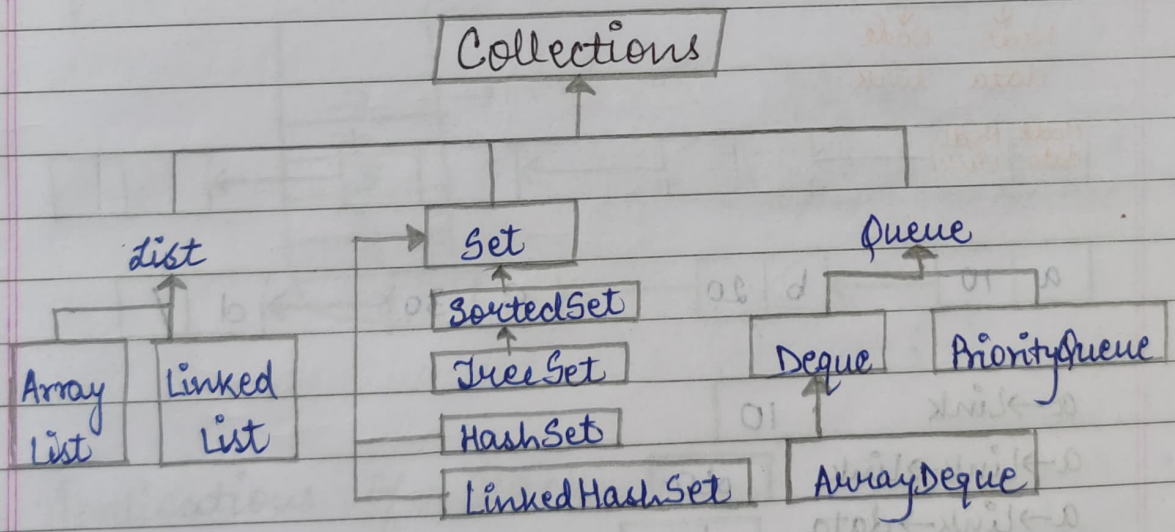
28.04.22

Chapter-3

Abstract Data Types & Java Collections

- ADT is the logical description of data and operations that are applied on it.
  Eg- Push & Pop a element into stack.
- ADT does not concern about actual implementation of the data structure.

★ Java Collection Framework



Data Structures

It represents how the data will be stored in the memory.
Eg:- Array, stack, Graph, Tree, Hashing

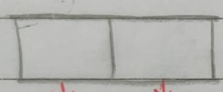
Page :
Date: / /

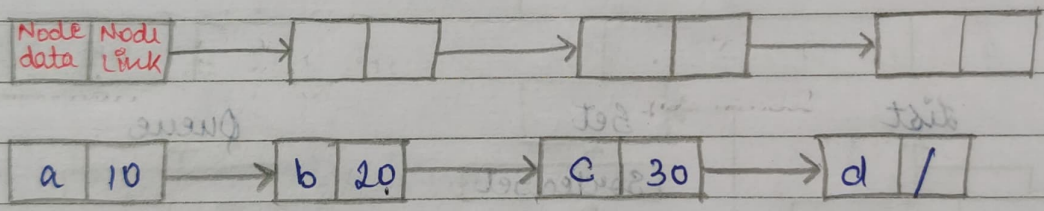
Array

Collection of multiple elements of same data type.

ADT Operations	T(c)
Add (k)	O(1)
Read (k)	O(1)
Substitute (k)	O(1)

Linked List





a→Link → 10

a→Link→Link → 20

a→Link→data → b

a→Link→Link→data → c

Linked list are dynamic data structures & memory is allocated at run time.

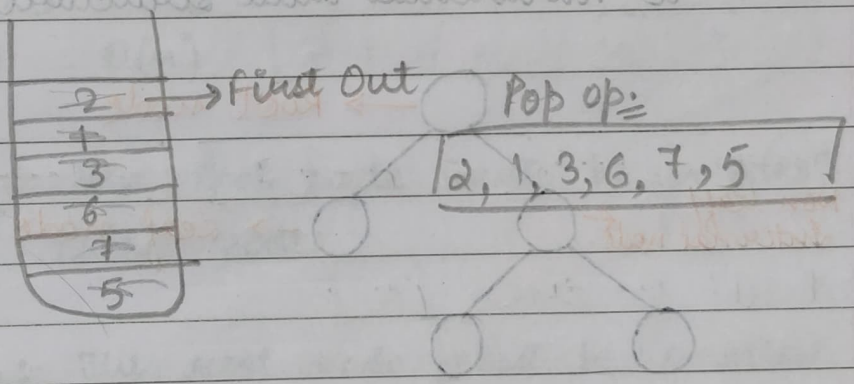
Page :
Date: / /

T.C

Insert()	O(1)
Delete()	O(1)
PrintList()	O(N)
Find K()	O(N)
Find K^{th}	O(N)
isEmpty()	O(1)

3) Stack

Last In First Out (LIFO)
First In Last Out (FILO)
eg :- 5, 7, 6, 3, 1, 2



Applications of Stack

1) Recursion
2) Infix to Postfix
3) Parsing
4) Editors
5) Tree traversals
6) Graph traversals
7) Postfix Expressions evaluation

Push()
Pop()
Top()
Size()
IsEmpty()
O(1)

Page :
Date: / /

4) Queue

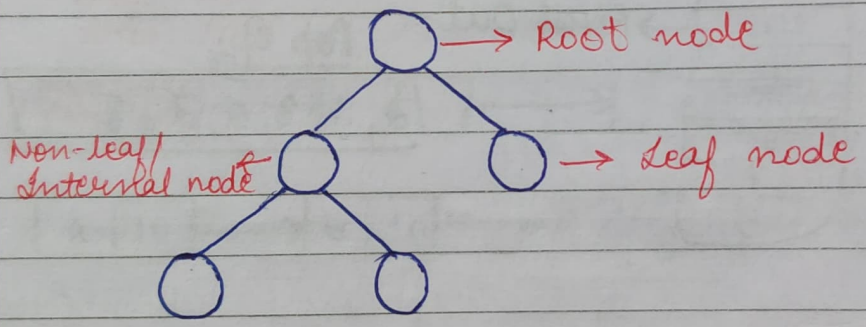
First in First Out (FIFO)

Insert (k)
Remove ()
Front ()
Size ()
IsEmpty ()

} O(1)

5) Tree

It is a hierarchial data structure



- Leaf node is the node which does not have any siblings.
- Non-leaf / Internal node which further has sibling node.

- Binary Tree (each node having atmost 2 child)
- Binary Search Tree
- B-Tree
- B+ Tree

Page :
Date: / /

Binary Search Tree (BST)

Insert (K)
Delete (K)
Search (K)
Find Max/Min ()

} O(log n) (when tree is balanced)
} O(n) (when tree is not balanced)

6) Heap

In heap the records are stored in an array

Insert () } O(log n)
Rename () }
Heapify () O(n)

a) Max-Heap :- The Root node must be greater than all its child nodes

b) Min-Heap :- The root node must be smallest than all its child nodes.

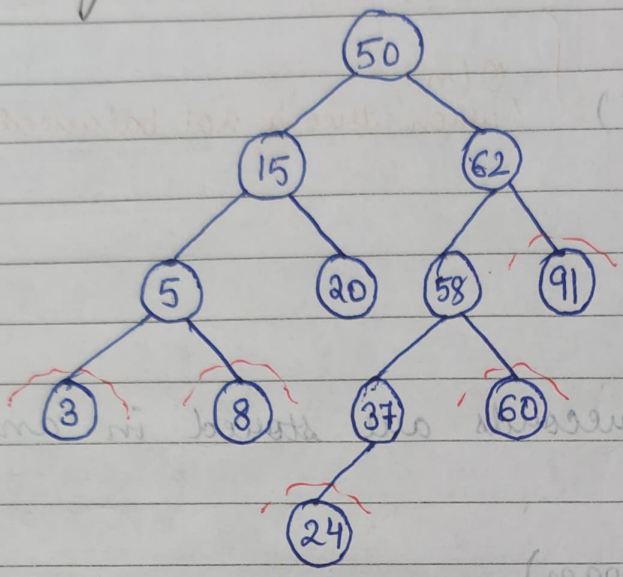
Ques- 24, 16, 9, 8, 17, 6, 5 . You want to Insert 11 . What will be the T.C.

Time complexity of 24, 16, 9, 8, 17, 6, 5 is O(n)
time complexity for insertion is O(log n)
So, total time complexity will be O(n) + O(log n) = O(n log n) //.

Page :
Date: / /

29.04.22

Ques- Insert 50, 15, 62, 5, 20, 58, 91, 38, 37, 60, 24 to build Binary Search Tree.



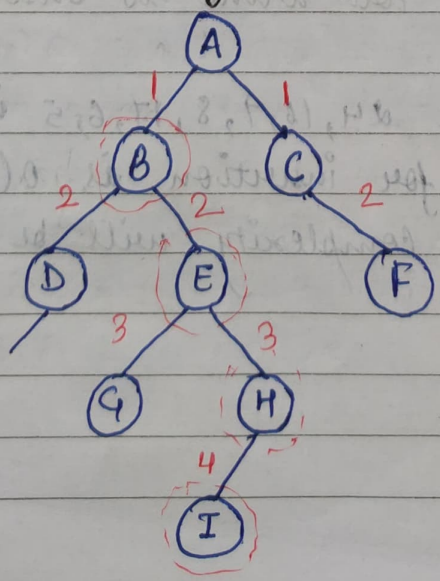
T.C. O(N)

No. of leaf node = 5

No. of non-leaf nodes = 7 [total nodes - leaf nodes]

Height of tree = 4

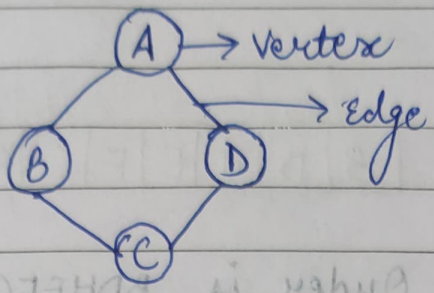
How to find height of tree



A & D = 2
A & G = 3
A & I = 4 → Height
A & F = 2

Page :
Date: / /

7) Graph

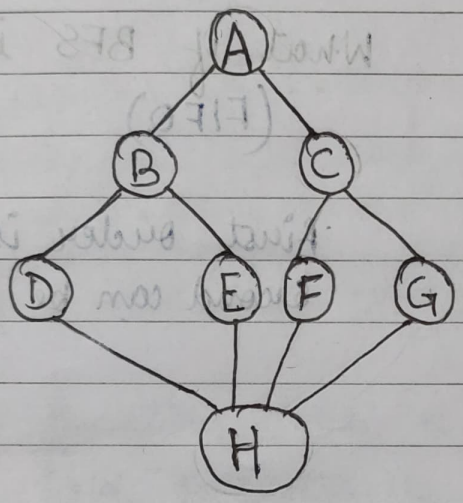


Traversal
- (stack) → Depth First Search (DFS)
- (Queue) → Breadth First Search (BFS)

Graph is represented with the set of vertices & edges.

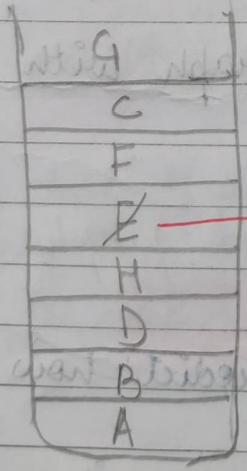
Ques-

There can be more than one DFS possible.



What is the order if DFS applied.

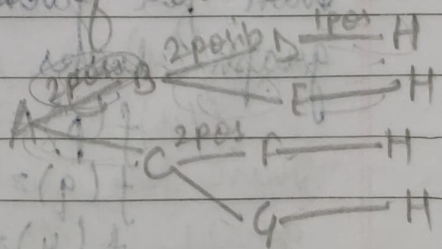
[After B is processed, from B we can go to A, D or E so we will go to D or E because A is already processed]



because the above and below element are already processed so it will backtrack

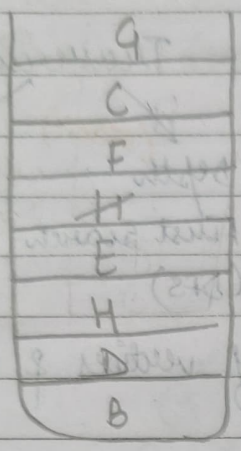
A	B	D	H	E	F	C	G

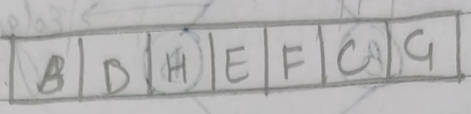
How to take out total no. of outcomes



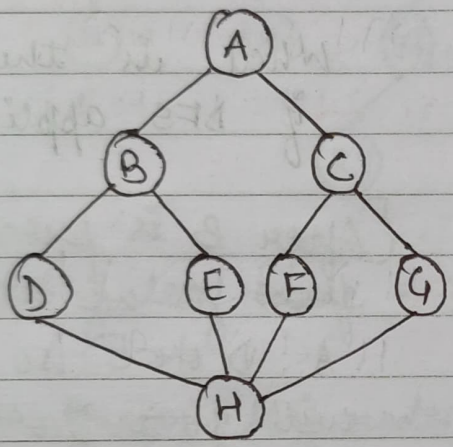
Order is ABDHEFCG
Second can be ACFHGEBD
Third can be ABEHFCGD

If it start from B





Order is BDHEFCG

Ques- 

What if BFS is applied? (FIFO)

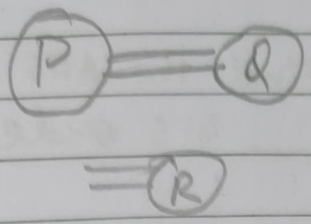
First order is ABCDEFGH
Second can be ACBFGDEH

Ques- Consider a DFS of an undirected graph with 3 vertices P, Q and R.

Q Let f(u) = first visited time
l(u) = last visited time
Then for given time instances predict how the graph looks like.

f(p) = 5 units l(p) = 12 units
f(q) = 6 units l(q) = 10 units
f(R) = 14 units l(R) = 18 units

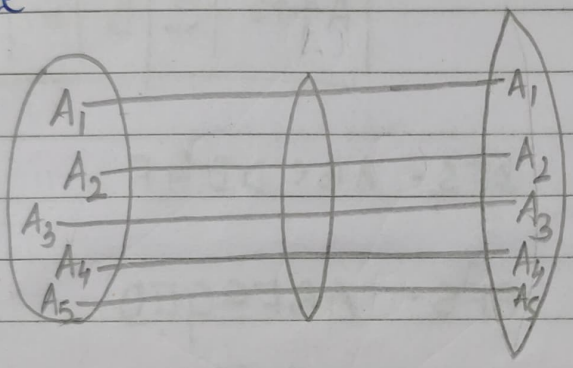
f(p) = 5 units l(p) = 9 units
f(q) = 6 units l(Q) = 10 units
f(R) = 14 units l(R) = 18 units



[∵ Q node is accesed at 6 mins so it will be connected to

8) Hashing

This technique is used to minimize the search time.



A hash function takes a message of arbitrary length and generates a fixed length code.

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

If size of the hash table is 'm' & we can insert only 'n' elements then,

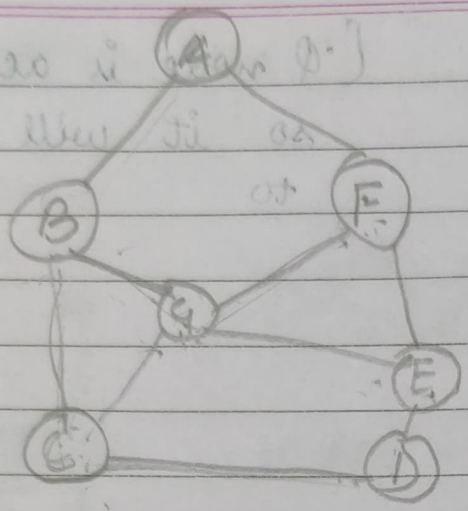
$$\text{Load factor } (\alpha) = \frac{m}{n}$$

$$0 \leq \alpha \leq 1$$

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

Page :
Date: / /

5.05.2022

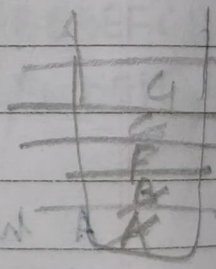


Find the BFS & DFS order

A	B	C	D	E	G	F

DFS - ABCDEGF

BFS - ABFCGED



Page :
Date: / /

5.05.2022

Chapter- 4 & 5

Sorting & Searching

BUBBLE SORT :- Bubble the element and swap if one number is less than the other.

Ques- 17 4 6 1 5 28

4	17	6	1	5	28	} Pass 1
4	6	17	1	5	28	
4	6	1	17	5	28	
4	6	1	5	17	28	
4	1	6	5	17	28	} Pass 2
4	1	5	6	17	28	
1	4	5	6	17	28	Pass 3

Ques-

3	5	7	2	6	1	16
3	5	2	7	6	1	16 } (1)
3	5	2	6	7	1	16
3	5	2	6	1	7	16
3	2	5	6	1	7	16 } (2)
3	2	5	1	6	7	16
2	3	5	1	6	7	16 } (3)
2	3	1	5	6	7	16
2	1	3	5	6	7	16 (4)

[i → Counter
Index is defined by j]

Bubble Sort

→ created a class named sorting

```
class sorting {          → for Print created a fn.
  Public static void PrintArray (int arr[]){
    for(int i=0; i<arr.length; i++) {
      System.out.print( arr[i] + " ");
    }
  System.out.println();
  }
                          → created an array inside main fn.
  Public static void main (string args[]) {
    int arr[] = { 9, 8, 2, 7, 5, 6};
    // bubble sort
Outer Loop ←  for(int i=0; i<arr.length-1; i++){   → (n-1) times
      for(int j=0; j<arr.length-i-1; j++) {
        if(arr[j] > arr[j+1]) {
          //swap
          int temp = arr[j];
          arr[j] = arr[j+1];
          arr[j+1] = temp;
        }
      }
    }
    PrintArray(arr);
  }
}
```

1 2 3 5 6 7 16 — (5)

No. of Passes – 5

```
void bubble_sort(int arr[], int n)
{
    for (int i=0; i<n; i++)
    {
        for( int j=0; j<n-i-1 ; j++)
        {
            if (arr[j] > arr[j+1])
                swap( arr[j], arr[j+1])
        }
    }
}
```

Insertion Sort

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
5	2	15	9	30	18	11

2, 5, 15, 9, 30, 18, 11 — Pass 1

2, 5, 15, 9, 30, 18, 11 — Pass 2

2, 5, 9, 15, 30, 18, 11 — Pass 3

2, 5, 9, 15, 30, 18, 11 — Pass 4

2, 5, 9, 15, 18, 30, 11 — Pass 5

2, 5, 9, 11, 15, 18, 30 — Pass 6

Worst case $O(n^2)$

Best case $\Omega(n)$

[when array is already sorted]

Page :
Date: / /

Inplace Algo : When an algo doesn't use extra space i.e., constant space complexity.

Outplace Algo : vice - versa

Pseudo - code

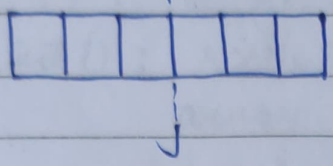
(suppose we have an array A).

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into sorted A[1 ... j-1]
    i = j - 1
    while i > 0 AND A[i] > key
        A[i+1] = A[i]
        i = i - 1
    A[i+1] = key.
```

ALGORITHM

- Array is split into two parts sorted & unsorted array.
- A key element is picked.
- Compare the key element with its predecessor.
- If the key element is smaller that its predecessor compare it to elements before. All greater elements are moved one position ahead of their current position.
- Index is increased by 1, & loop is iterated.

Special cases :



By Linear Search

No. of comparison → $O(n)$
No. of movements → $O(n)$
} $O(n^2)$

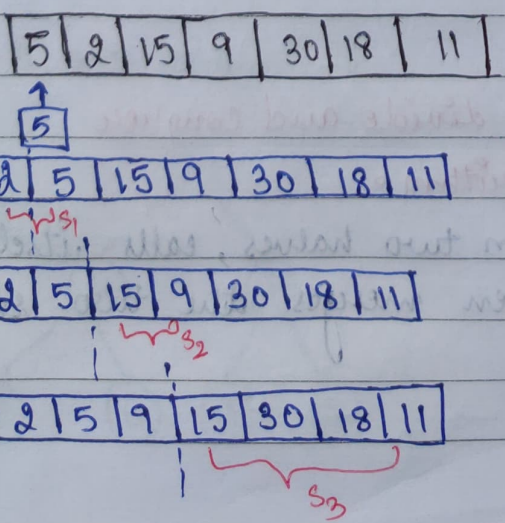
Note :-

Either we can apply linear search or binary search on left sub-array elements. We cannot reduce worst case time complexity for insertion sort i.e., $O(n^2)$

Selection Sort

→ select
→ compare
→ swap

Q-



Page :
Date: / /

④ | 2 | 5 | 9 | 11 | 30 | 18 | 15 |

⑤ | 2 | 5 | 9 | 11 | 15 | 18 | 30 |

⑥ | 2 | 5 | 9 | 11 | 15 | 18 | 30 |

```
void sort (int arr[])
 {
    int n = arr.length
     int min;
  for( int i=0; i< n-1; i++)
     { min = i;
      for (int j=i ; j< n ; j++)
        {
          if (arr[j] < arr[min])
                min = j;
        }
    int temp= arr[min];
     arr[min] = arr[i];
     arr[i] = temp;
    }
}
```

7.05.22

Merge Sort

is a divide and conquer algorithm.

It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

Page :
Date: / /

Two functions are divided in this algo:

- merge () : merging two halves
- MergeSort () : recursively calls itself to divide the array till size becomes one.

l = left index r = right index

38 (0), 27 (1), 43 (2), 3 (3), 9 (4), 82 (5), 10 (6)

Is $l < r$

Yes

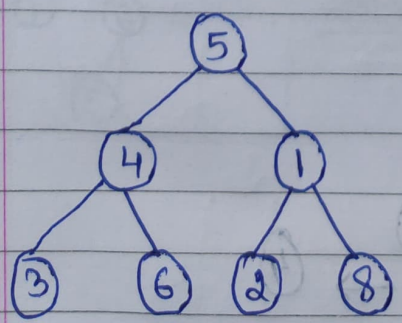
$m = l + (r-1)/2$

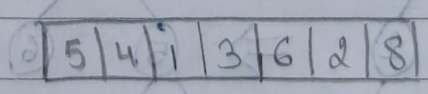
Divide the array in two parts.

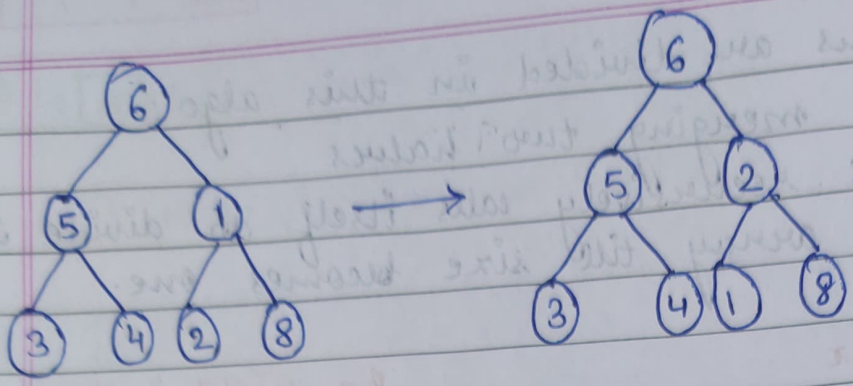
Heap Sort

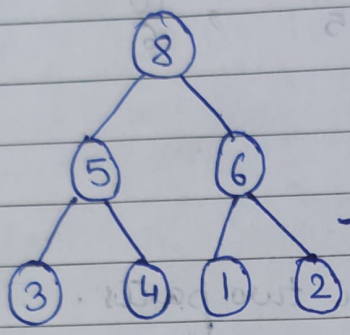
- Build heap-tree
- Build max-heap
- Delete the root
- Update the array and repeat step 2 until sorted

Q 5, 4, 1, 3, 6, 2, 8





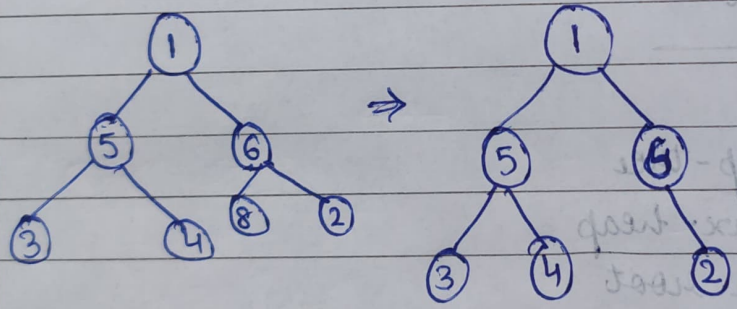




→ max heap

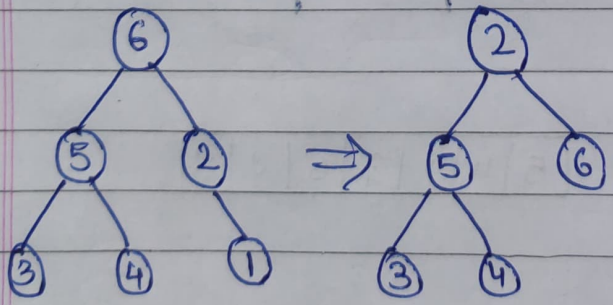
8	5	6	3	4	1	2

Delete Root Node



1					6

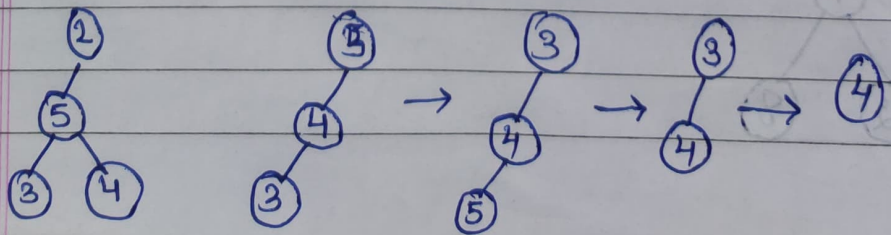
After del 8



1	5	6	3	4	2	8

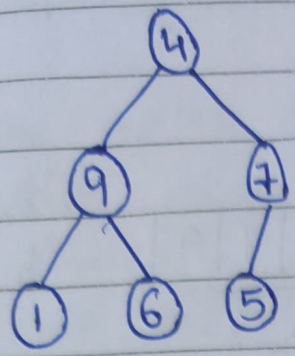
After del 6

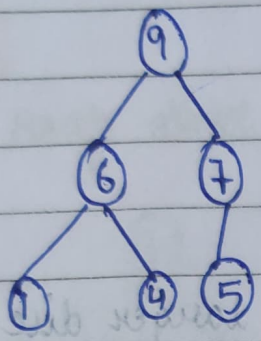
→



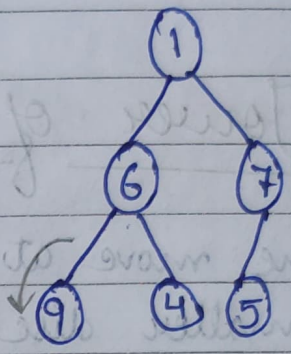
Eg:- 4, 9, 7, 1, 6, 5

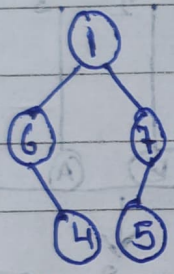
4	9	7	1	6	5

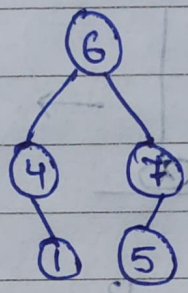


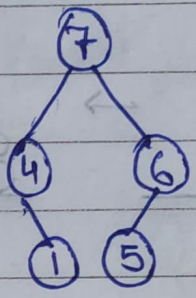


Delete root 9:

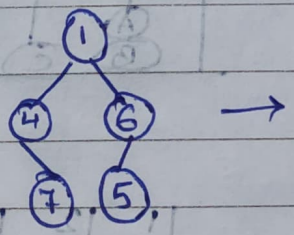


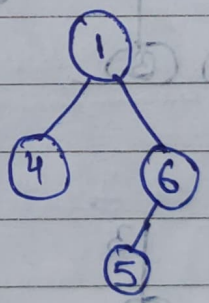




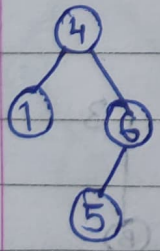


Delete Root 7:

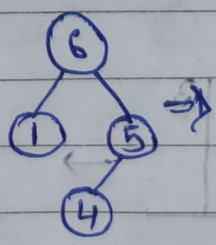


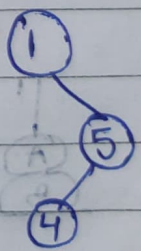


1	4	6	5	7	9



Delete Root 6:

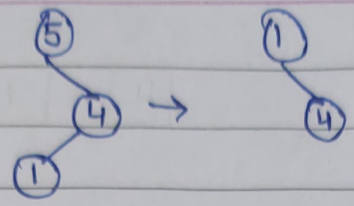




1	5	4	6	7	9

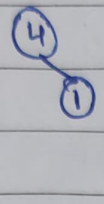
Page :
Date: / /

Delete Root 5 :



1	4	5	6	7	9

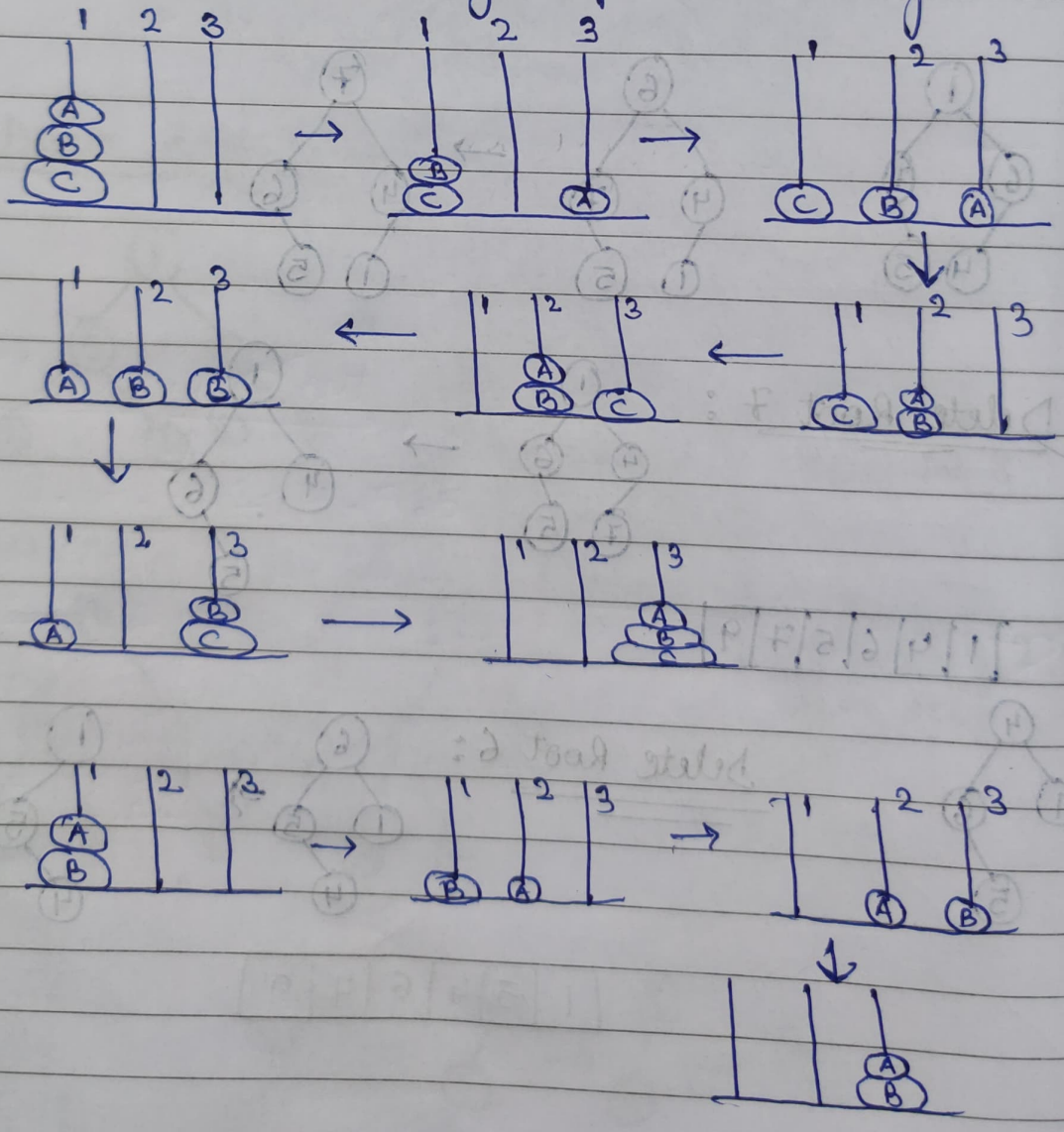
Delete Root 4 :



1	4	5	6	7	9

Tower of Hanoi

1. One move at every step
2. Smaller disc can only be placed on larger disc.



Page :
Date: / /

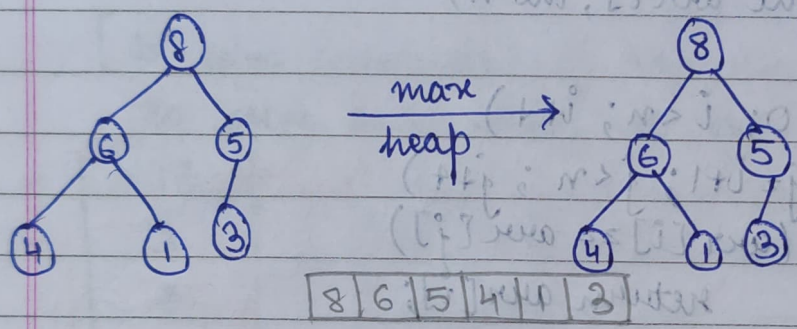
$2^n - 1$

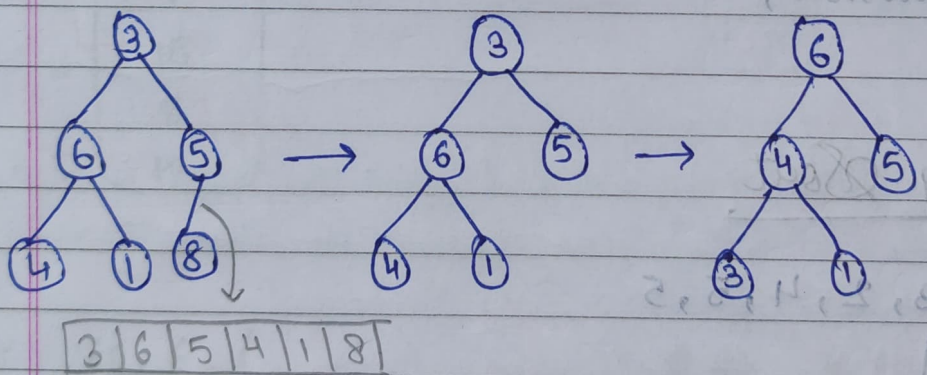
n = no. of disc

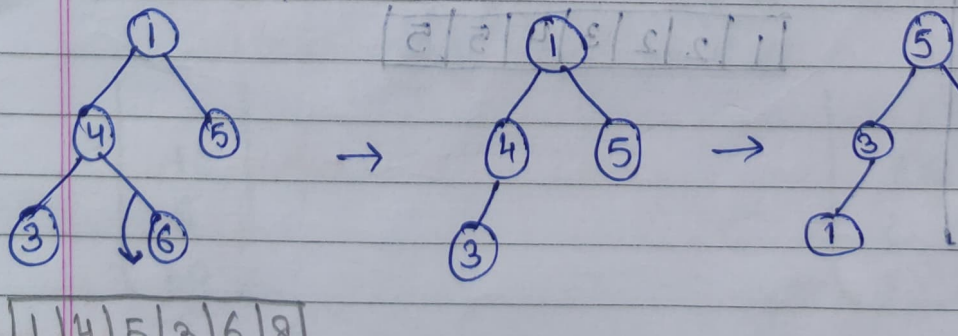
→ In tower of Hanoi problem if there are n no. of disc then the optimal movement required is $2^n - 1$

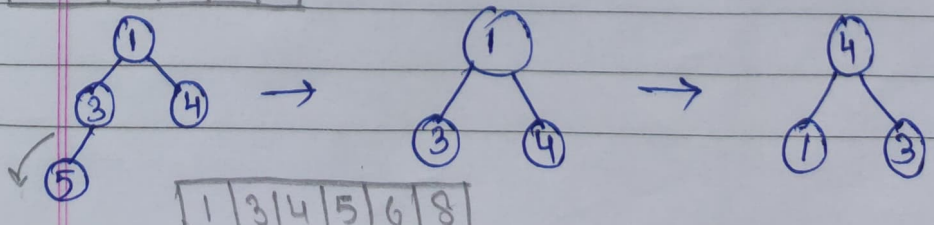
12.05.22

Heap sort: 8, 6, 5, 4, 1, 3



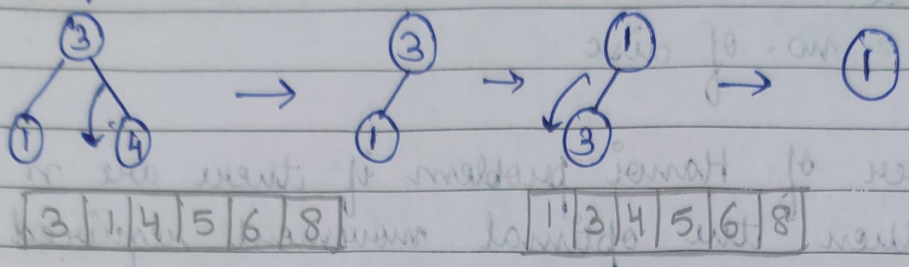






Page :
Date: / /

MAX



Q1 WAP to print the first repeated no. from an unsorted array-

Input - 6, 4, 5, 3, 3, 8, 9, 9

O/P - 3

Sol:-

```
int repeat (int arr[], int n)
{
  for (int i=0; i<n; i++)
    for (int j= i+1; j<n ; j++)
        if (arr[i] == arr[j])
            return arr[i];
  return 0;
}
```

Counting Sort

1, 2, 3, 2, 4, 5, 5

1	1
2	X 2
3	1
4	1
5	X 2

1	2	2	3	4	5	5

(one swap in one iteration)

Selection Sort

```
for(int i=0; i<arr.length-1; i++) {
    int smallest = i;
    for(int j=i+1; j<arr.length; j++){
        if(arr[smallest] > arr[j]){
            smallest = j;
        }
    }
    int temp = arr[smallest];
    arr[smallest] = arr[i];
    arr[i] = temp;
}
```

Insertion Sort

```
for(int i=1; i<arr.length; i++){
    int current = arr[i];
    int j = i-1;
    while(j>=0 && current<arr[j]){
        arr[j+1] = arr[j];
        j--;
    }
//Placement
    arr[j+1] = current;
}
```

Bucket Sort

Bucket Sort (A)

1.) Let B[0 --- n-1] be a new array

2.) n = A.length

3.) for i = 0 to n-1

4.) make B[i] an empty list

5.) for i = 1 to n

6.) insert A[i] into list B[⌊n A[i]⌋]

7.) for i = 0 to n-1

8.) sort list B[i] with insertion sort

9.) Concatenate the lists together

B[0], B[1] — —

13/5/22

Page :
Date: / /

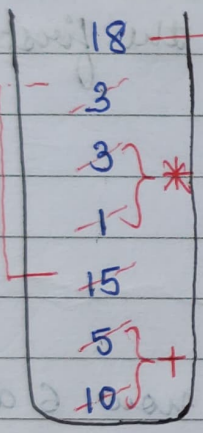
Evaluation of Postfix Expression

Operator ⇒ +, -, *, /
Operand ⇒ Number/ digit, Variables

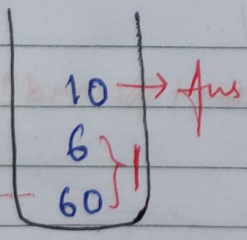
Prefix ⇒ +ab
Infix ⇒ a+b
Postfix ⇒ ab+

Ques- 10, 5, +, 1, 3, *, +

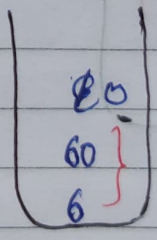
[To solve conversion of exp use ⇒ operator stack
To solve evaluation of exp use ⇒ operand stack



Ques- 60, 6, /

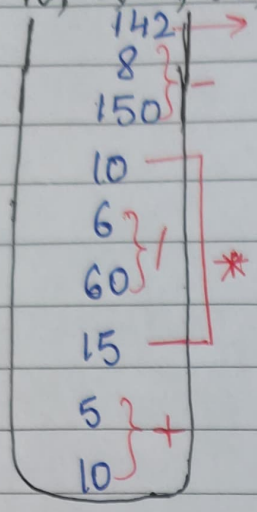


Ques- 6, 60, /



Page :
Date: / /

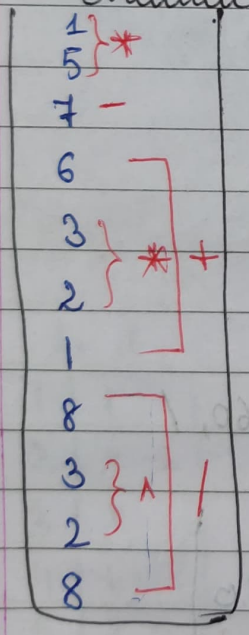
Ques- 10, 5, +, 60, 6, /, *, 8, -

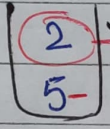


Ques- 8, 2, 3, ^, /, 2, 3, *, +, 5, 1, *, -

i) What is the top two elements after the first * operation?

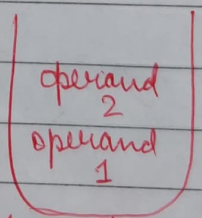
ii) Evaluate the final expression





i) 6 and 1 because now 6 and 1 are there in the stack.

Note :-



operand 1 + operator + operand 2