Algorithm:

Algorithm is a step by step procedure for solving a computational problem

Need of writing the algorithms:

1) In order to work with the computer each instruction given by the user is in the form of programme
2) In order to write the program for a problem first of all we have to know the procedure how to solve that problem.
3) The procedure that is written step by step is called as Algorithm.
4) For a problem there exists several algorithms, the algorithms are analysed based on the time and memory requirement.
5) The algorithm is converted into a programme, the programme is executed in the computer to get the required output.

Ex: Algorithm for preparing a receipe (tea)

step 1 : start

step 2 : Take a bowl and pour some water boil the water and add the tea powder and sugar

step 3 : Add some milk

step 4 : Filter the mixture

step 5 : stop

Scanned with CamScanner

Characteristics of an algorithm:

Any algorithm must have the following characteristics.

* **Inputs:** Algorithm must accept zero (or) more inputs.
* **outputs:** Must produce atleast one output.
* **Finiteness:** Must terminate after the finite no. of steps
* **Definiteness:** Each instruction in the algorithm must be clear and unambiguous and should be understandable by the computer easily.
* **Effectiveness:** we should not use the unnecessary instructions in the algorithm.

Pseudo code for Expressing Algorithms:

* pseudo code is nothing but the way of expressing algorithm

(or)

pseudo code is representation of the algorithm in which instruction sequence can be given with the help of the programming constraints

* Algorithm is a pseudocode consisting of Heading and body, the heading consists of name of the procedure and parameter list.
* The syntax is:

```
Algorithm  name of the algorithm (parameter 1, parameter 2)
{
begin
    ----- ;
    ----- ;
    -----
}
end
```

Assignment: 1) =
2) :=
3) ←

- read (scanf)
- write (printf)

Scanned with CamScanner

* The Beginning and ending of the algorithm is indicated with open and closed flower brackets
* The delimeter / semicolon is used at the end of the each statement
* single line comments are written using two forward slashes (//)
* The identifier should be a combination (or) alpha-numeric string
* Using assignment operator, an assignment statement can be given.
* There are other types of operators, such as boolean operators, logical operators.
* The array indices are stored within [] square brackets
* The input-in and output-in can be done using read and write statements

The conditional statements if, then and if then else are written in the following form

* if condition then
  statement
* if condition then
  statement
  else
  statement

* The while statement can be written as

while (condition) do

{

statement 1 ;

statement 2 ;

- - - - -

- - - - -

}

Scanned with CamScanner

* The general form for writing for loop is

for variable = value 1, value 2, --- value n do

statement 1

statement 2

for variable = 1 to n do

statements

Ex:

```
Algorithm sum (a, n)
{
    s := 0;
    for i := 1 to n do
        s = s + a[i];
    return s;
}
```

Performance analysis:

The performance of an algorithm is measured by computing the time and space complexity of algorithms

Time complexity:

Time complexity of an algorithm is the amount of computer time required by an algorithm to run (or) completion. There are two types of computing times: compile time and run time

The time complexity is generally computed using run time / execution time. It is difficult to compute the time complexity in terms of physically clocked time. The time complexity is therefore in terms of frequency count.

Frequency count is basically a count denoting the no. of times execution of the statement.

Scanned with CamScanner

1) Algorithm for sum of 'n' numbers:

```
Algorithm sum (a, n)
{
    int s:=0;
    for i:=1 to n do
        s ← s + a[i];
    return s;
}
```

step / execution	frequency count
0	0
0	0
1	1
n+1	n+1
1	n
1	1
0	0

Time complexity: $1 + n+1 + n + 1 = 2n+3$

$\therefore O(n)$

2) Algorithm for matrix addition:

```
Algorithm matrix ( )
{
    for i:=1 to n do
        for j:=1 to m do
            s[i][j] = a[i][j]
                    + b[i][j];
}
```

step / execution	frequency count
0	0
0	0
1	n+1
1	n(m+1)
1	n.m
0	0

Execution time: $n+1 + mn + n + mn = 2mn + 2n + 1$

Time complexity: $O(m.n)$

Scanned with CamScanner

4) Algorithm for fibonacci series:

Algorithm fib(n)	FC True	FC False	S/E True	S/E False
	0	0	0	0
{	0	0	0	0
if (n≤1) then	1	1	1	1
write n ;	1	0	1	0
else	0	0	0	0
{ $f_1 = 0$; $f_2 = 1$;	0	0	0	0
for i = 2 to n do	0	1	0	1
	0	1	0	1
{	0	n	0	1
$f_3 = f_1 + f_2$;	0	0	0	0
$f_1 = f_2$;	0	n-1	0	1
$f_2 = f_3$;	0	n-1	0	1
write f_3;	0	n-1	0	1
}	0	n-1	0	1
}	0	0	0	0
}	0	0	0	0
	0	0	0	0

Execution time:

False = (n-1) + (n-1) + (n+1) + (n-1) + n + 3 = 5n-1

True = 1+1 = 2

Time complexity: Best case (True) = O(1)

Worst case (false) = O(n)

5) sum of 'n' odd numbers:

Algorithm sum_odd(n)	S/E	FC
	0	0
{	0	0
for (i=1, i<n, i=i+2)	1	n/2 + 1
{	0	0
s ← s+i ;	1	n/2
}	0	0
return s;	1	1
}	0	0

Execution time: $\frac{n}{2} + 1 + \frac{n}{2} + 1 = \frac{2n}{2} + 2 = n + 2$

Time complexity: O(n)

Scanned with CamScanner

Space complexity :

It is the amount of memory needed to execute an algorithm. Memory is needed for the algorithm for three reasons :

Instruction space :

This is the space needed to store the instructions

Environmental stack space :

The memory needed to store the suspended inst- -ructions.

Ex : Let A() and B() are two functions.

Let us assume that function B() is called some- -where else in function A(), Before calling the function B() the instructions of function A() are stored in environmental stack space. Once the function B() is finished, then the function A() remaining instructions are executed by fetching the instructions from stack space

Data space :

Data space is the space needed to store the constants and variables of the given algorithm.

* space complexity is of two types :

1) Constant space complexity
2) Linear space complexity

Constant space complexity :

The space requirement doesn't vary with the variation of the input size.

Ex : Algorithm sum (a, b, c)

```
{
    return a+b+c ;
}
```

Scanned with CamScanner

In this algorithm the space complexity is the space needed to store the variables a, b, c. If the variables are type of 'int' then space requirement is 6 bytes.

Linear space complexity:

The space complexity varies with the variation of the input size.

Ex: Algorithm sum (a, n)

{

int s := 0;

for i := 1 to n do

s ← s + a[i];

return s;

}

The space complexity is the space required to store the variables. s, i, n, a[i].

Here the space requirement of a[i] is varies with the variation of the value 'n'.

Asymptotic Notations:

There are 5 types of notations. They are:

1) Bigh - oh - notation
2) Big - omega - notation
3) Theta notation
4) Little - oh - notation
5) Little - omega - notation

1) Asymptotic Notations are used to represent the time complexity of an algorithm

Scanned with CamScanner

* 1) Big-oh- Notation:

Used to represent the upper bound running time of the algorithm.

* Represents the largest time taken by algorithms.

Formal definition:

Let f(n) and g(n) be two non-negative functions If there exists an integer 'n_0' and a constant 'c' such that $f(n) \leq c \cdot g(n)$, $c > 0$ and for all integers $n >= n_0$ then

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

Ex: $f(n) = 3n+2$, $g(n) = n$ $f(n) = O(g(n))$?

To prove that f(n) is big-oh- of g(n) we have to follow a procedure for condition $f(n) \leq c \cdot g(n)$

Here we have to find 'c' and 'n'

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

put $c=1$ and $n=1$

$3n+2 \not\leq c \cdot n$ $(5 \not\leq 1)$

put $c=2$ and $n=2$

$3(2)+2 \not\leq 2 \cdot 2$

so if we put $c=4$ and $n=2$

$3(2)+2 = 4 \cdot 2$, so here $n_0 = 2$.

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

for $c=4$ and $n \geq 2$ $(n_0 = 2)$

2) $f(n) = 100n + 6$, $g(n) = n$ show that $f(n) = O(g(n))$

Ans: To prove the condition $f(n) \leq c \cdot g(n)$

we have to find 'c' and n_0.

put $c=1$, $n=1$ in

$100n + 6 \leq c \cdot n$

$106 \not\leq 1$

Scanned with CamScanner

put c = 50 and n = 1

$100(n) + 6 \leq 50 \cdot 2$

So the coefficient of 'c' should be greater than the coefficient of n in the f(n)

put c = 101, and n = 6

$100(6) + 6 \leq 101 \cdot 6$

$606 = 606$.

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

for $c = 101$ and $n \geq 6$ $(n_0 = 6)$

3) $f(n) = 10n^2 + 4n + 2$ and $g(n) = n$

show that $10n^2 + 4n + 2 \neq O(n)$?

Ans: Given $f(n) = 10n^2 + 4n + 2$ and $g(n) = n$;

to prove the condition $f(n) = O(g(n))$

g(n) should be the upper bound of f(n) but here g(n) is not the upper bound.

So we can't find the 'n_0' value for the given functions.

$\therefore f(n) \neq O(g(n))$

* Big - Omega - Notation :

Used to represent the lower bound of running time of the algorithm.

* Represents the smallest time taken by algorithm.

Formal definition :

Let f(n) and g(n) be the two non-negative functions. If there exists an integer 'n_0' and a constant 'c' such that $c > 0$ and for all integers 'n' is greater than (or) equal to n_0 and

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

Scanned with CamScanner

Examples :

1) Given $f(n) = 5n+10$, $g(n) = 2n$, show that $f(n) = \Omega g(n)$

Ans: In Big-omega notation, $g(n)$ is the lower bound

to prove that $f(n) \geq cg(n)$

we have to find 'c' and 'n_0'

put $c=1$ and $n=1$

$$5(1)+10 \geq 1 \cdot 2(1)$$

$$15 \geq 2$$

for every value of 'n' irrespective of 'c' value the condition $f(n) \geq c \cdot g(n)$ is satisfied.

$\therefore f(n) = \Omega g(n)$ for $c=1$ and $n \geq 1$

2) Given $f(n) = 2n+5$ and $g(n) = 2n$ show that $f(n) = \Omega g(n)$

Ans: To prove that $f(n) \geq c \cdot g(n)$, we have to find 'c' and 'n_0' values.

put $c=1$ and $n=1$

$$2(1)+5 \geq 1\ 2(1)$$

$$7 \geq 2$$

for every value of 'n' irrespective of 'c' and value the condition $f(n) \geq c \cdot g(n)$ is satisfied.

$\therefore f(n) = \Omega g(n)$ for $c=1$ and $n \geq 1$

3) Given $f(n) = n+1$ and $g(n) = n^2$ show that $f(n) = \Omega g(n)$

Ans: for the Big-omega notation, as we all know $g(n)$ degree should be lesser (or) $g(n)$ be the lower bound

Here $f(n) = n+1$ and $g(n) = n^2$

means $g(n)$ degree is greater than $f(n)$

Hence the condition is not satisfied.

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

Scanned with CamScanner

Theta Notation:

It represents the average running time taken by the algorithm.

Formal Definition:

Let $f(n)$ and $g(n)$ are two non-negative integers if there exists an integer n_0 and constants c_1 and c_2 such that $c_1 > 0$ and $c_2 > 0$ for all integers $n \geq n_0$ then

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

Examples:

1) $f(n) = 2n+8$, $g(n) = n$ show that $f(n) = \Theta\, g(n)$

Ans:

Big-omega-notation: $f(n) \geq c_2\, g(n)$

$$2n+8 \geq c.n$$

put $c=1$, $n=1$

$$2(1)+8 \geq 1$$

$$10 \geq 1$$

$\therefore c_1 = 10$ and $n=1$

Big-oh-notation: $f(n) \leq c_1 \cdot g(n)$

$$2n+8 \leq c \cdot n$$

put $c=1$, $n=1$

$$10 \not\leq 1$$

put $c=10$, $n=1$

$$2(1)+8 \leq 10 \cdot 1$$

$$10 \leq 10$$

$\therefore c_2 = 10$ and $n=1$

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

$\therefore c_1 \cdot g(n) \leq f(n) \leq c_2\, g(n)$

$$n \leq 2n+8 \leq 10n$$

Scanned with CamScanner

2) $f(n) = n^2+n$, $g(n) = 5n^2$ then show that $f(n) = \Theta g(n)$

Ans:

Big - omega - notation : $c_1 \cdot g(n) \leq f(n)$

$$c_1 \cdot 5n^2 \leq n^2+n$$

put $c = 1/5$ and $n = 1$

$$1/5 \cdot 5(1)^2 \leq 1+1$$

$$1 \leq 2 \qquad \therefore c_1 = 1/5, \; n = 1$$

Big - oh - notation : $f(n) \leq c_2 \cdot g(n)$

$$n^2+n \leq c_2 \cdot 5n^2$$

put $c = 1, \; n = 1$

$$1+1 \leq 1 \cdot 5(1)$$

$$2 \leq 5 \qquad \therefore c_2 = 1, \; n = 1$$

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

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

$$\boxed{1/5 \cdot 5n^2 \leq n^2+n \leq 5n^2}$$

Little - oh - notation :

Let $f(n)$ and $g(n)$ be two non-negative functions

then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ such that $f(n) = o\,g(n)$

That is $f(n) = o\,g(n)$ and $f(n) \neq \Theta(g(n))$

Ex 1 : $f(n) = n$, $g(n) = n^2$ $f(n) = o\,g(n)$?

Ans:

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \left(\frac{n}{n^2}\right)$$

$$= \frac{1}{\infty} = 0.$$

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

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

Scanned with CamScanner

Little omega notation:

Let $f(n)$ and $g(n)$ be two non-negative functions

then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ (or) $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$

such that $f(n) = \omega g(n)$ that is $f(n) \neq \Theta g(n)$

Ex 1: $f(n) = 3^n$ and $g(n) = 2^n$ $\quad f(n) = \omega g(n)$?

Given $f(n) = 3^n$ and $g(n) = 2^n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3^n}{2^n}$$

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

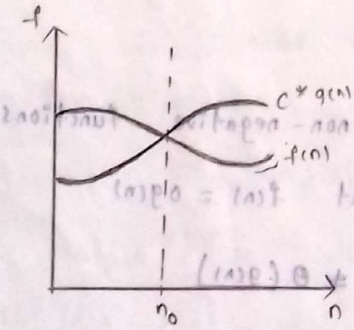
$$= \infty$$

$$\therefore \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

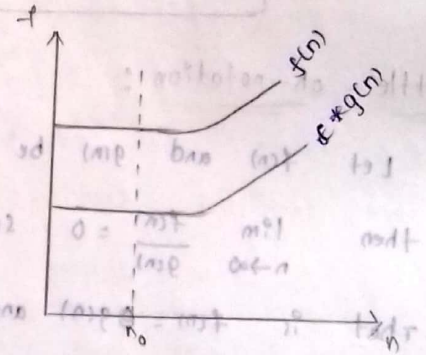
$f(n) = \omega g(n)$

1. show that $f(n) = 8n+128 \neq \Omega(n^2)$
2. $4n^2 - 64n + 288 = \Omega(n^2)$
3. write an alg to find largest of given 'n' no's. derive its time complexity using big-oh notation

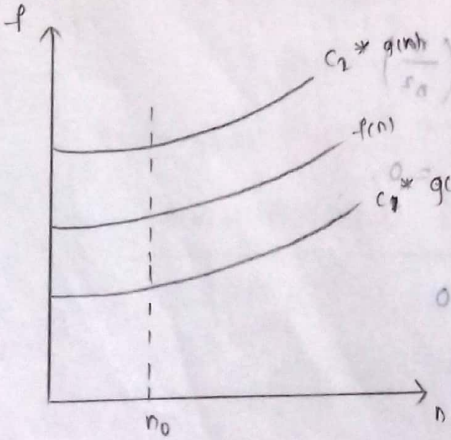
Big-oh-notation:



Big-omega-notation:



Theta notation:



Note:

* Any notation is used to represent the time complexity of an algorithm in any case (best, average, worst)
* $1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 \_\_\_ < 2^n < 3^n \_\_\_ < n^n$

Best case :- min time taken by the alg ex:- linear search → 1st element found at 1st pos

Avg case :- Avg " " " " " " → found at somewhere else in middle

worst case :- max " " " " " " → found at last or not found

Scanned with CamScanner

probability :

It is defined as how likely the things to be happen.

* probability of an event is found by using the formula $\frac{\epsilon}{s}$ where 'ϵ' is event and 's' is the sample space.

* The set of possible outcomes of an event is called as sample space and each outcome is called as sample point.

Ex: probability of getting a head on tossing a coin

$s = \{H, T\}$

$\epsilon = H$

$\text{probability} = \frac{\epsilon}{s} = \frac{1}{2}$

2) probability of getting even numbers on rolling a die?

Ans: $s = \{1, 2, 3, 4, 5, 6\}$ & $\epsilon = \{2, 4, 6\}$

$P(\epsilon) = \frac{3}{6} = \frac{1}{2} = 0.5$

Mutually Exclusive:

Two events are said to be mutually exclusive if they don't have any element in common.

$\therefore \epsilon_1 \cap \epsilon_2 = \emptyset$

Ex: ϵ_1 = getting 2 heads on tossing two coins

ϵ_2 = getting 2 tails on tossing two coins

$\epsilon_1 = \{HH\}$ and $\epsilon_2 = \{TT\}$

$\therefore \epsilon_1 \cap \epsilon_2 = \emptyset$

Independent events:

Occuring of one event doesn't effects the occuring of another event.

Ex: ϵ_1 = on rolling a die and ϵ_2 = tossing a coin

the probability of getting head (or) tail for a coin and the probability of getting (1 - 6) on the faces of die are independent.

Scanned with CamScanner

Conditional probability :

Happening of the previous event effects the happening of the present event.

Ex : E_1 is raining out side there is 30% probability raining today

E_2 = planning to go outside.

conditional probability : $P\left(\frac{E_1}{E_2}\right) = \frac{P(E_1 \cap E_2)}{P(E_2)}$

Amortized Analysis :

Assume that there are 'n' operations an average cost of an operation to be small even though a single operation in the sequence might be expensive some operations may have higher cost and some operations may have lower cost. Lower cost operations are called worst case operations.

* If some part in the algorithm is slow and the remaining parts are fast. The average time complexity is better than the slower part.

18/07/2014

Set : set is a collection of elements with similar property

Disjoint sets :

Two sets are said to be disjoint if they don't have any element in common.

Ex : $S_1 = \{1,2,3,4\}$, $S_2 = \{5,6\}$

$S_1 \cap S_2 = \emptyset$

$\therefore S_1, S_2$ are disjoint sets

Disjoint set operations :

Union : The union of the two sets is a set containing all the elements of the first set and all the elements of the second set.

Ex : $S_1 = \{1,7,8,9\}$, $S_2 = \{5,2,10\}$

Scanned with CamScanner

Algorithm:

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

→ In insertion sort we will start from the 2nd element in the list. it is compared with 1st element if 2nd ele is smaller than 1st element then 2nd element is placed in 1st position.

→ In general in insertion sort say if the element is 4th element and has to be compared with all the previous elements, if the 4th ele is greater than 3rd ele then it is not compared with 2nd & 1st ele bcz 1st & 2nd elements are less than 3rd element. If the 4th element is smaller than 3rd element then 3rd element is placed in 4th element place, 4th element is not placed in 3rd element place (4th element is placed after finding the correct position to it until it is in temp variable), next temp is compared with 2nd ele if temp is smaller than 2nd then 2nd ele is copied to 3rd element place then temp is compared with 1st element if temp is smaller than 1st ele then 1st element is moved to

Scanned with CamScanner

2nd place and temp is stored in 1st place.

Analysis:-

```
for(i=1;i<n;i++) ⟶ n
{
  for(j=i;j>0 && temp<a[j-1];j--) →(n-1)(n-1)
  {
  }
}
```

$O(n^2)$ in worst case & Avg case

Best case :- here, inner for loop doesn't enter inside if the previous elements are already sorted hence time complexity is $O(n)$.

Scanned with CamScanner

Insertion Sort:

0	1	2	3	4
30	10	40	20	50

index = 1 temp = 10

10 compared with 30, 10 < 30 True so 30 moved to right

30 30 40 20 50

replace index 0 with temp.

10 30 40 20 50 ⟶ I iteration.

index = 2
temp = 40

40 compared with all the previous elements

40 < 30 False so don't change positions here we need not to compared 40 with 10 as 10 and 30 are in sorted order

index = 3, temp = 20

20 compared with all the previous elements

20 < 40 True
40 moved to index 3

10 30 40 40 50

20 < 30 True
30 copied to index 2

10 30 30 40 50

20 < 10 False no change

so copy 20 at index 1.

10 20 30 40 50

index = 4 temp = 50

50 < 40 False No change

Scanned with CamScanner

Heap sort:

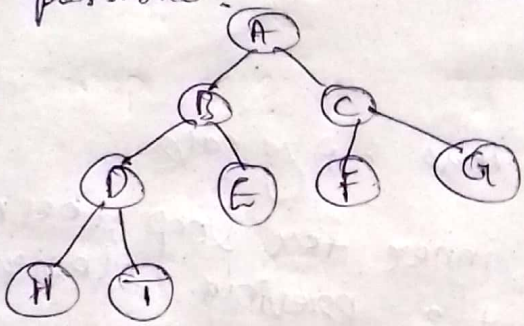
A heap is a complete binary tree

- Build the comple binary tree for the given elements
- apply max heap

complete binary tree:

- A complete binary tree is a binary tree in which every level except, possibly the last, is completely filled, and all nodes are as far left as possible.

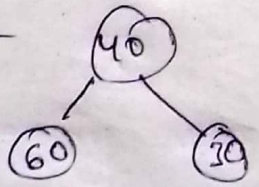
→ elements should be present in left right manner



ex:- 40, 60, 30, 10, 20, 50 build complete binary tree.

→ while building complete binary tree we have to follow the order "Root left right"

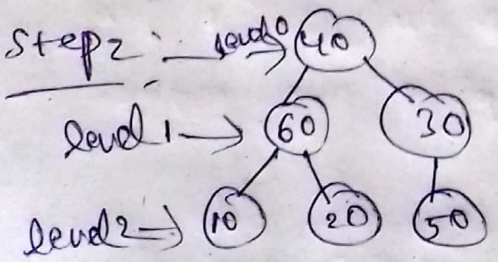
step1:-



step2:- level 0 → 40

level 1 → 60, 30

level 2 → 10, 20, 50



max heap:- parent node should be greater than or equal to child nodes.

→ Delete and replace root node with the last node

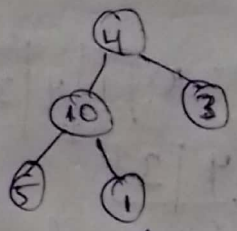
→ deleted node will be placed at the last position of the array.

Scanned with CamScanner

ex → heap sort ex:-

0	1	2	3	4 ← index
4	10	3	5	1 ← input

→ Build complete binary tree

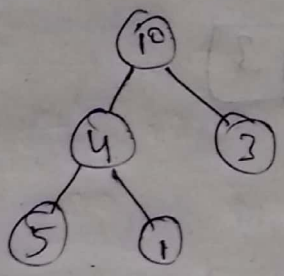


0	1	2	3	4
4	10	3	5	1

→ Transform to max-heap

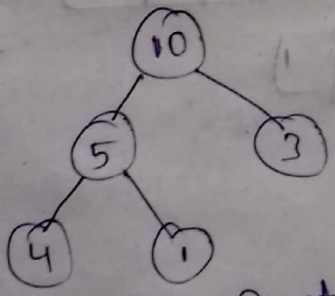
* 5 & 1 are less than 10
* 10 is greater than 4, so swap 4 & 10

0	1	2	3	4
10	4	3	5	1



* 5 is greater than 4, so swap 5 & 4

0	1	2	3	4
10	5	3	4	1

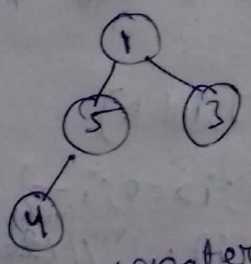


* swap first and last node and delete last node from heap.

0	1	2	3	4
1	5	3	4	10 (delete)

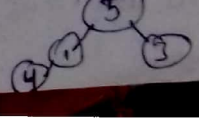
* now construct the maxheap with

0	1	2	3
1	5	3	4



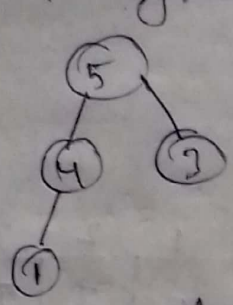
* 5 is greater than 4
* 1 is less than 5 so swap 1 & 5

0	1	2	3
5	1	3	4



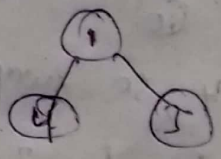
Scanned with CamScanner

* 4 is greater than 1 so swap 1 & 4

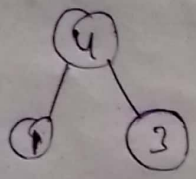


5	4	3	1

* swap 1st & last nodes | 1 | 4 | 3 | 5 |
* delete last node | 1 | 4 | 3 |
* construct max heap with the elements 1 4 3



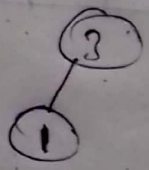
swap 4 & 1 | 4 | 1 | 3 |



swap 1st & last nodes | 3 | 1 | 4 |

remove last node

construct max heap with | 3 | 1 |



already max heap

swap 1st & last nodes | 1 | 3 |

remove last nodes | 1 |

only one element left in the heap.

Algorithm:-

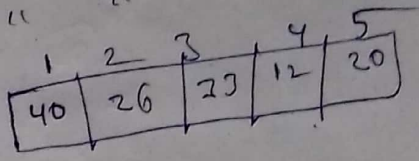
```
void heapsort(int a[], int n)
{
    for(int i = n/2 - 1; i >= 0; i--)        → n/2
        heapify(arr, n, i);                  → n/2 (log n)   // create max-heap
    for(int i = n-1; i >= 0; i--)            → n
    {
```

Scanned with CamScanner

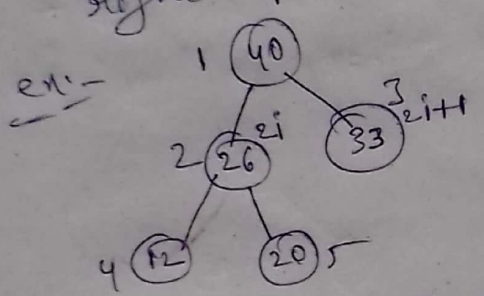
swap(arr[0], arr[i]); → // swap 1st & last node
heapify(arr, i, 0); → // creates max heap on reduced array.
}

How to store heap in array

main root will be stored at index = 1
for a root/parent with index = i
left successor/child stored at index = 2i
right " " " = 2i+1



ex:-



→ If a child is at index = i, then its parent will be at index = lower bound(i/2) ex: child is 5, Parent = LB(5/2) = 2

Heapification (or) heapify algorithm:-

Heapify(A, i, n) → array, → index of a node, → no. of elements.

```
{ left = 2i;
  right = 2i+1;
  if(left<n and A[left] > A[i])
  { max = left;
  }
  else
  {max = i;
  }
  if(right<n and A[right] > A[max])
    { max = right;
    }
    if(i != max)
     { swap(A[i], A[max]) } heapify(A, max n)
    }
```

Scanned with CamScanner

Time complexity :- $O(n \log n)$

The complexity of heapification is algorithm is $\log n$ (height of the binary tree i.e $\log_2 n$ (max no. of distance that a node can cover is equal to the height of the binary tree).