SHAMBHUNATH INSTITUTE OF ENGINEERING & TECHNOLOGY

Subject: Design and analysis of algorithm Solution

Subject Code: (RCS-502)

Course: B.Tech **SEMESTER: 5th**

FIRST SESSIONAL EXAMINATION, ODD SEMESTER, (2019-2020)

Branch: Computer Science & Engineering

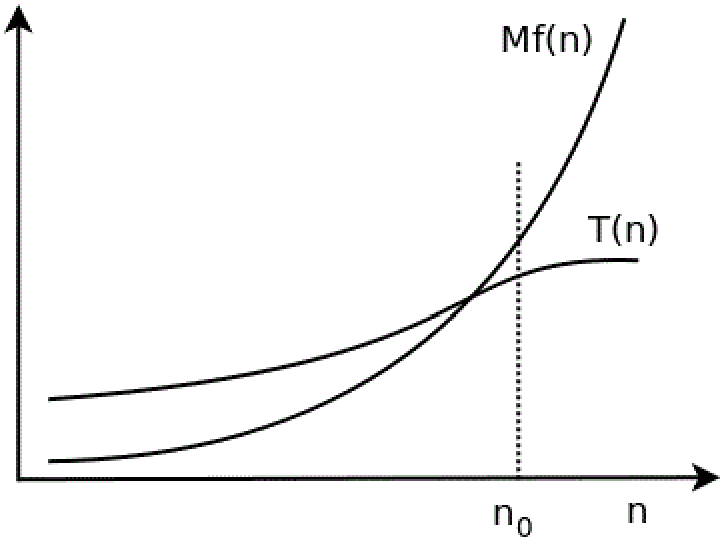
SECTION - A

Q-1) Attempt ALL parts.

a) Define Big O, small o and small ω notation with example?

Answer:

Big O: Let f(n) and g(n) be two positive functions. We write **f(n) ∊ O(g(n))**, and say that f(n) has order of g(n), if there are positive constants C and n_0 such that f(n) ≤ C·g(n) for all $n \geq n_0$.



T(n) = O(f(n))

Example-f(n) = $n^2/2 - n/2$. With Big O notation, this becomes **f(n) ∊ O(n^2)**, and we say that the algorithm has **quadratic time complexity**.

Small o: Let f(n) and g(n) be functions that map positive integers to positive real numbers. We say that f(n) is o(g(n)) (or f(n) E o(g(n))) if for **any real** constant c > 0, there exists an integer constant n0 ≥ 1 suchthat0≤f(n)<c*g(n).

Small ω: Let f(n) and g(n) be functions that map positive integers to positive real numbers. We say that f(n) is o(g(n)) (or f(n) E o(g(n))) if for **any real** constant c > 0, there exists an integer constant n0 ≥ 1 such that f(n)≤c*g(n).

b) What is the binomial tree? You are also required to give its properties.

Answer:

The ***binomial tree*** B_k is an ordered tree defined recursively. The binomial tree B_0 consists of a single node. The binomial tree B_k consists of two binomial trees B_k-1 that are ***linked*** together: the root of one is the leftmost child of the root of the other.

For the binomial tree B_k,

1. there are 2^k nodes,
2. the height of the tree is k,
3. there are exactly $\binom{k}{i}$ nodes at depth i for $i = 0, 1, \dots, k$, and
4. the root has degree k, which is greater than that of any other node; moreover if the children of the root are numbered from left to right by $k - 1, k - 2, \dots, 0$, child i is the root of a subtree B_i.

c) What do you mean by stable sort? Name two stable sort algorithms.

Answer:

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.

Example of stable sorting algorithm:

(I) Counting sort

(II) Merge Sort

d) Is the sequence < 23,17,14,6,13,10,1,5,7,12> a max heap ?

Answer:

No. Since PARENT (7) is 6 in the array. This violates the max-heap property.

e) Write the recurrence relation for binary search algorithm?

Answer:

Recurrence relation- T(n)=T(n/2)+1

Time complexity is O ($\log n$)

SECTION – B

Q 2) Attempt any TWO parts from this section.

a) Answer:

Compare Stassen's matrix multiplication algorithm with traditional multiplication algorithm related to time complexity. Multiply the two matrices using Stassen's formula-

$$\begin{pmatrix} 3 & 2 \\ 1 & 7 \end{pmatrix} * \begin{pmatrix} 4 & 3 \\ 7 & 2 \end{pmatrix}$$

Answer: Let us consider two matrices A and B. We want to calculate the resultant matrix C by multiplying A and B.

Naïve Method (Traditional multiplication algorithm)

Using Naïve method, two matrices can be multiplied if the orders of these matrices are $p \times q$ and $q \times r$.

Algorithm: Matrix-Multiplication (X, Y, Z)

```
for i = 1 to p do
  for j = 1 to r do
    C[i,j] := 0
      for k = 1 to q do
        C[i,j] := C[i,j] + A[i,k] × B[k,j]
```

The running time of traditional matrix multiplication is as follows after solving equation

$$T(n) = 8T\left(\frac{n}{2}\right) + 4n^2$$

is $(n) = O(n^3)$.

Strassen's matrix multiplication algorithm can be performed Divide and Conquer only on square matrices where n is a power of 2. Order of both of the matrices are n × n. u

P1 = (A11+ A22)(B11+B22)
P2 = (A21 + A22) * B11
P3 = A11 * (B12 - B22)
P4 = A22 * (B21 - B11)
P5 = (A11 + A12) * B22
P6 = (A21 - A11) * (B11 + B12)
P7 = (A12 - A22) * (B21 + B22)
C11 = P1 + P4 - P5 + P7
C12 = P3 + P5
C21 = P2 + P4
C22 = P1 + P3 - P2 + P6

Recurrence for new algorithm is $T(n) = 7T\left(\frac{n}{2}\right) + 14n^2$ and its running time is$T(n) = O(n^{2.81})$.

Multiply the two matrices using Stassen's formula-

$$\begin{pmatrix} 3 & 2 \\ 1 & 7 \end{pmatrix} * \begin{pmatrix} 4 & 3 \\ 7 & 2 \end{pmatrix}$$

We use the above formula and put value:

P1 = (A11+ A22)(B11+B22)
P2 = (A21 + A22) * B11
P3 = A11 * (B12 - B22)
P4 = A22 * (B21 - B11)
P5 = (A11 + A12) * B22
P6 = (A21 - A11) * (B11 + B12)
P7 = (A12 - A22) * (B21 + B22)
C11 = P1 + P4 - P5 + P7
C12 = P3 + P5
C21 = P2 + P4
C22 = P1 + P3 - P2 + P6

Get C = $\begin{pmatrix} 26 & 13 \\ 53 & 17 \end{pmatrix}$

b) Sort the following unsorted data set A using **gap=5**.

A=(5,77,32,45,56,68,75,45,76,52,44,36,56,24)

Answer: Take gap=5

5, 68, 44 $\dashrightarrow$ 5, 44, 68
77, 75, 36 $\dashrightarrow$ 36, 75, 77
32, 45, 56 --------.> already sorted
45, 76, 24 $\dashrightarrow$ 24,45,76
56, 52 $\dashrightarrow$ 52, 56

A= 5,36,32,24,52,44,75,45,45,56,68,77,56,76

Now gap = floor (gap/2) =floor (5/2) =2

5, 32, 52, 75,45,68,56 $\dashrightarrow$ 5,32,45,52,56,68,75
36, 24,44,45,56, 77, 76 $\dashrightarrow$ 24,36,44,45,56,76,77

A=5,24,32,26,45,44,52,45,56,56,68,76,75,77

Now gap=2/1=1

So finally we get A=(5,24,2632,44,45,45,56,56,68,76,77)

c) Solve **any two**-recurrence equation:

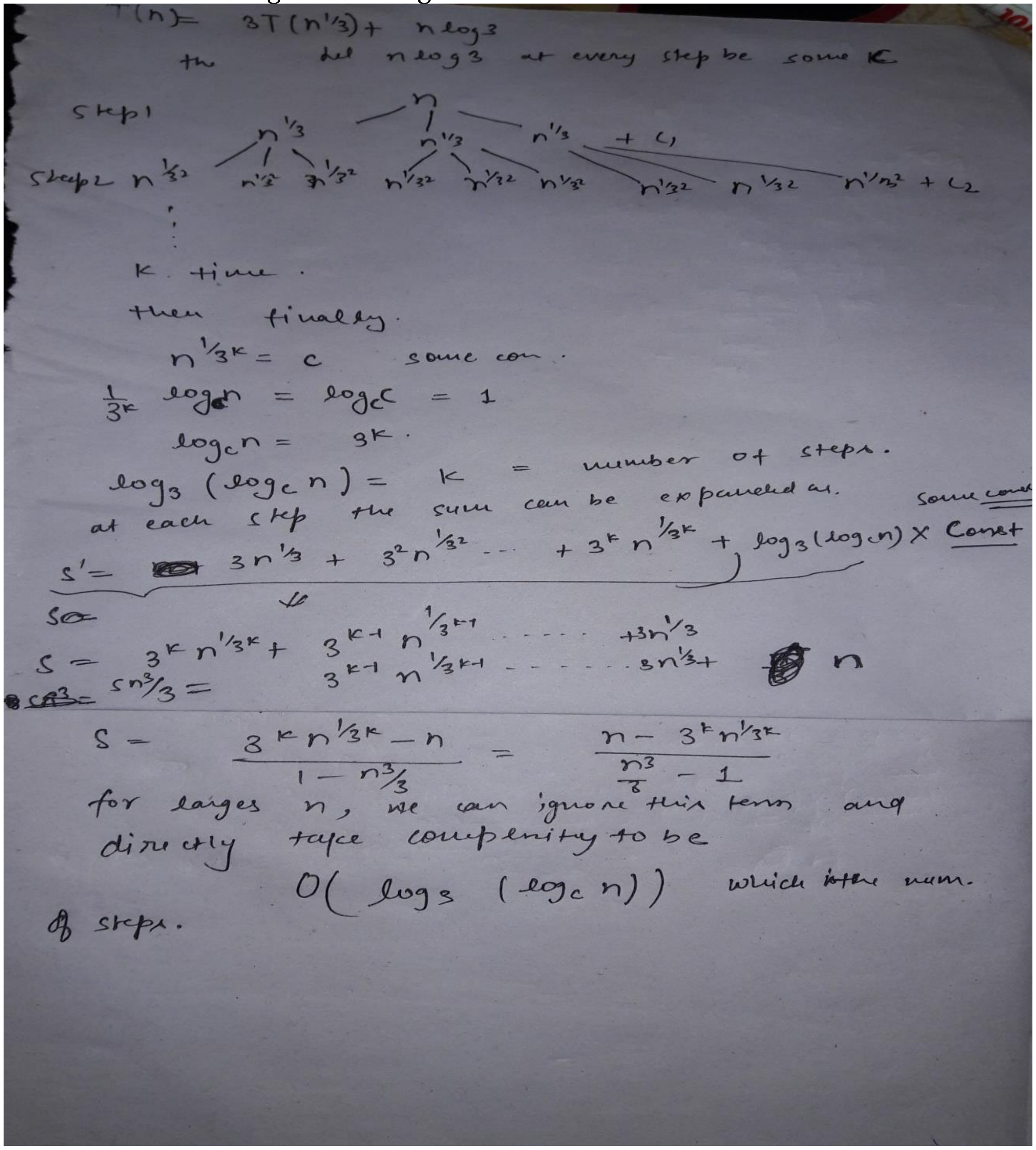
(i) $T(n) = 3\,T\left(n^{\frac{1}{3}}\right) + \log 3^n$

(ii) $T(n) = T(\propto n) + T((1-\infty)n)$

(iii) $T(n) = n + T({}^{n}/_{3}) + T({}^{2n}/_{3})$

Answer (I)

$$\log 3^n = n * \log 3$$



$T(n) = 3T(n^{1/3}) + n\log 3$

the let $n\log 3$ at every step be some K

step1 n

$n^{1/3}$ $n^{1/3}$ $n^{1/3}$ $+ c_1$

step2 $n^{\frac{1}{3^2}}$ $n^{1/3^2}$ $n^{1/3^2}$ $n^{1/3^2}$ $n^{1/3^2}$ $n^{1/3^2}$ $n^{1/3^2}$ $n^{1/3^2}$ $n^{1/3^2}$ $+ c_2$

⋮

K time.

then finally.

$n^{1/3^K} = c$ some con.

$\frac{1}{3^K} \log_c n = \log_c c = 1$

$\log_c n = 3^K$.

$\log_3(\log_c n) = K$ = number of steps.

at each step the sum can be expanded as. some const

$s' = 3n^{1/3} + 3^2 n^{1/3^2} \cdots + 3^K n^{1/3^K} + \log_3(\log_c n) \times$ Const

So

$S = 3^K n^{1/3^K} + 3^{K-1} n^{1/3^{K-1}} \cdots\cdots +3n^{1/3}$

$Sn^3/3 = 3^{K-1} n^{1/3^{K-1}} \cdots\cdots 3n^{1/3} + n$

$$S = \frac{3^K n^{1/3^K} - n}{1 - n^3/3} = \frac{n - 3^K n^{1/3^K}}{\frac{n^3}{3} - 1}$$

for larger n, we can ignore this term and directly take complexity to be

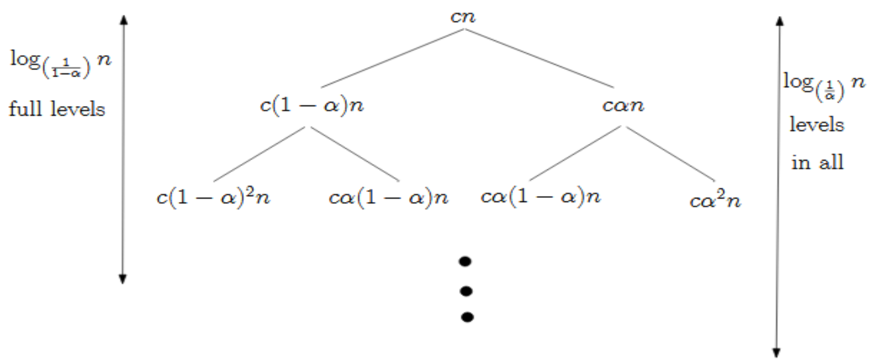
$O(\log_3(\log_c n))$ which is the num. of steps.

(ii)

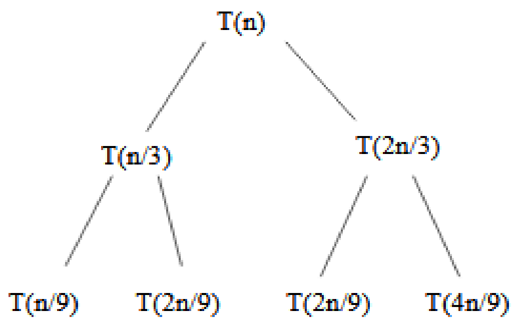
The recurrence tree is given below:

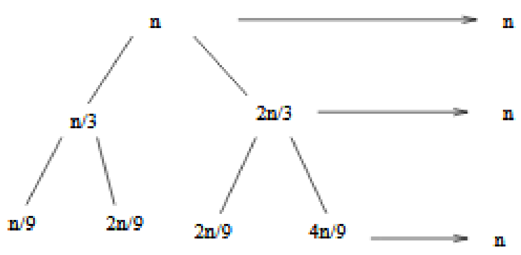
$$T(n) = \theta(cost * height)$$

$$T(n) = \theta(n * \log_{1-\alpha} n)$$



(iii)T(n) = T(n/3) + T(2n/3) + O(n)





Answer:
Note that the leaves are between the levels log3 n and log3/2 n From the computation tree, it is clear that the maximum height is log3/2 n. Therefore, the cost is at most log3/2 n· cn = O(n log n).
Cost is same so use

$$T(n) = \theta(cost * height)$$

$$T(n) = \theta(n * \log_{\frac{3}{2}} n)$$

d) Illustrate the operation of counting sort on the following array:
A= {4, 0, 2, 0, 1, 2, 1, 3}
In addition, check that it is a stable or not?
Answer:

i	1	2	3	4	5	6	7	8
A[i]	4	0	2	0	1	2	1	3

Here the highest number k = 4 so make a temporary array C [0……4] .

For i= 0 to we assign C[i] = 0.

i	0	1	2	3	4
C[i]	0	0	0	0	0

Now count the frequency of index j = 1,2,3,4.5,6,7,8   using   C[A[j]] =C[A[j]] +1

So

i	0	1	2	3	4
C[i]	2	2	2	1	1

Now we prepare for i = 1 to 4 cumulative frequency using C[i] =C[i]+C[i+1]

i	0	1	2	3	4
C[i]	2	4	6	7	8

Now for j= 8 to 1 using B[C[A[j]]] =A[j] and decrease the frequency using C[A[j]] =C[A[j]] -1

I	1	2	3	4	5	6	7	8
B[i]	0	0	1	1	2	2	3	4

Yes, it is a stable sort.

SECTION – C

3. Attempt any ONE part of the following.

a) Explain the properties of red black tree**.** Show the red black trees that result after successively Inserting the Keys 41, 38, 31, 12, 19, 8 into an initially empty red black tree.

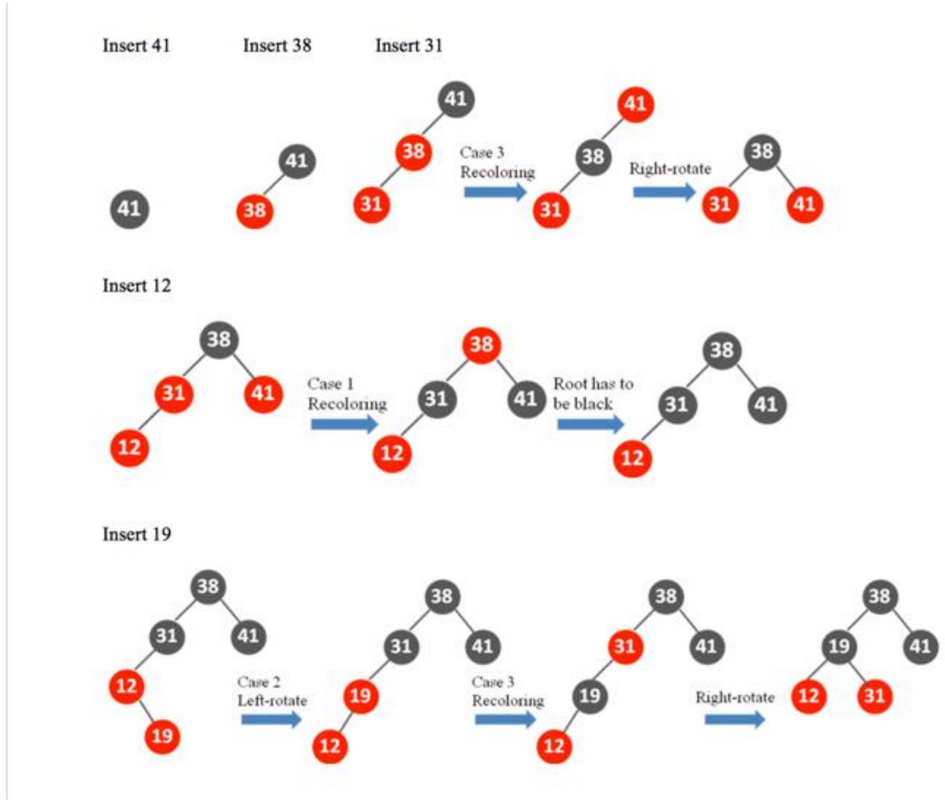
Answer:

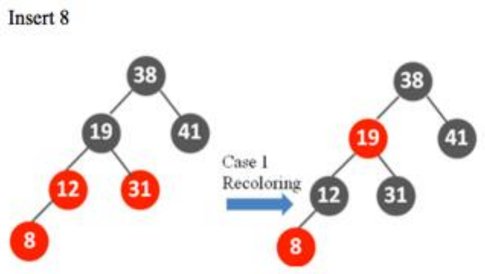
A ***red-black tree*** is a binary search tree with one extra bit of storage per node: its ***color***, which can be either RED or BLACK. By constraining the way nodes can be colored on any path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately ***balanced.***

Each node of the tree now contains the fields *color, key, left, right*, and *p.* If a child or the parent of a node does not exist, the corresponding pointer field of the node contains the value NIL. We shall regard these NIL'S as being pointers to external nodes (leaves) of the binary search tree and the normal, key-bearing nodes as being internal nodes of the tree.

A binary search tree is a red-black tree if it satisfies the following ***red-black properties:***

1. Every node is either red or black.
2. Every leaf (NIL) is black.
3. If a node is red, then both its children are black.
4. Every simple path from a node to a descendant leaf contains the same number of black nodes. Inserting the **Keys 41, 38, 31, 12, 19, 8 into an initially empty red black tree**:





b) Why don't we allow a minimum degree of t=1? Show the result of inserting the Keys ***F, S, Q, K, C,L, H, T, V, W, M, R, N, P, A, B, X,Y, D,Z ,E*** in order into an empty b-tree with minimum degree 2. Only draw the configurations of the tree just before some node must split , and also draw the final configuration.

Answer:

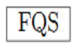
Minimum degree =t => Minimum t-1 keys , Maximum 2t-1 keys
=> Minimum t child , Maximum 2t child
so,
Minimum degree =1 => Minimum 0 keys , Maximum 1 key
=> Minimum 1 child , Maximum 2 child

Minimum case doesn't exist i.e., no node exists with o keys and no node with only 1 child exists because a node with 1 key has 2 child ....

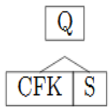
Inserting the Keys F***, S, Q, K, C,L, H, T, V, W, M, R, N, P, A, B, X,Y, D,Z ,E*** in order into an empty b-tree with minimum degree 2.

We have t = 2, therefore the most a node can hold 2t – 1 = 3 (node is full when number of key = 3) and the least t –1 = 1.

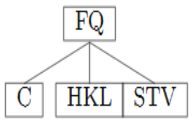
1. Insert *F*, *Q*, *S*



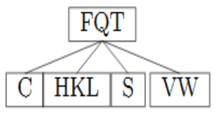
2. Insert *K*, *C*



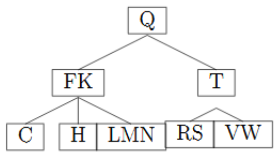
3. Insert *L*, *H*, *T*, *V*



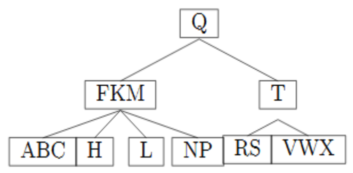
4. Insert *W*



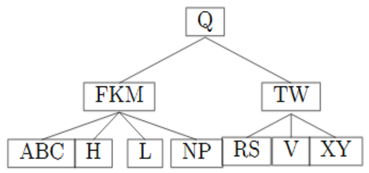
5. Insert *R*, *M*, *N*



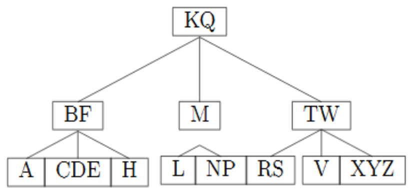
6. Insert *P*, *A*, *B*, *X*



7. Insert *Y*



8. Insert *D*, *Z*, *E* **FINAL CONFIGURATION**



4. Attempt any ONE part of the following. (1*5 = 5)

a) Write the pseudo code for PARTITION OF QUICK SORT and run the quick sort upon following data item: **A = <15,31,1,9,80,12,14,7,24>.**

Answer:

Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray *A*[*p* . . *r*] in place.

```
PARTITION(A,p,r)
    X ← a[r]
    i← p-1
    for j ← p to r-1
    do if A[j] ≤ x
    then i = i + 1
    exchange A[i] ↔A[j]
    exchange A[i+1] ↔A[r
    return i +1.
```

i	1	2	3	4	5	6	7	8	9
A[i]	15	31	1	9	80	12	14	7	24

Here p = 1 and r = 9 so X=A[9] = 24 and i = 0

J = 1 to 8 now A[1] ≤ 24 true so i=i+1=1 so A[1] ↔ A[1].

For j =2 now A [2] ≤ 24 as 31≤ 24 false so I not change

Finally for j = 3 to 8 the complete pseudo code executes we found that all the number less than X=24 is in left part while more than X=24 is in right part.

15	*12*	*1*	*9*	*14*	*12*	**<u>24</u>**	*31*	*80*

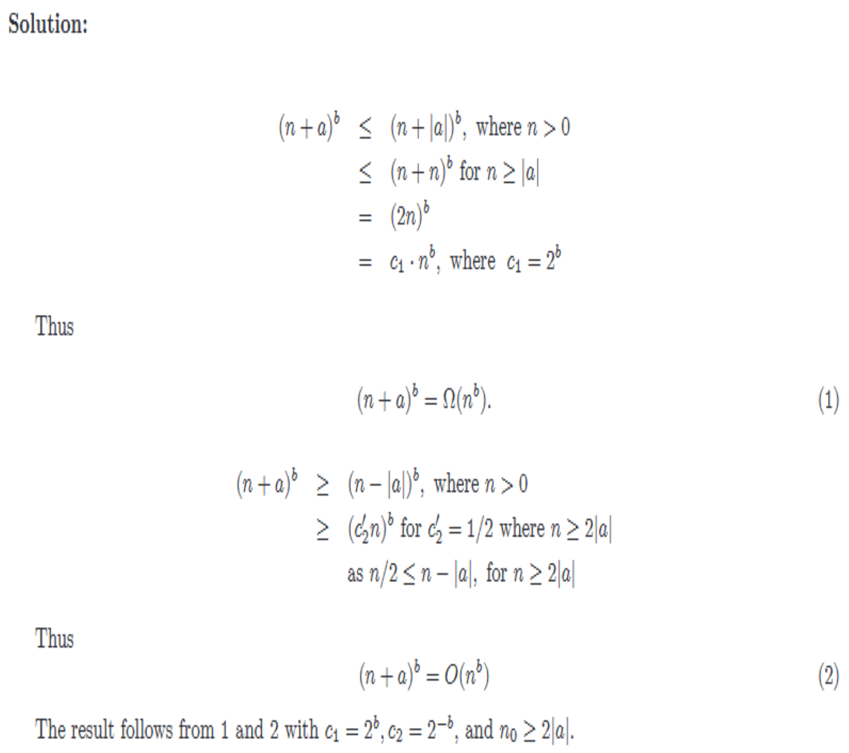
c) **(i)** Given $f(n) = (n+a)^b$ then show that $f(n) = \theta(n^b)$.

Answer: $\lim_{n\to\infty} (n+a)^b / n^b = \lim_{n\to\infty} n^b \left(1+\frac{a}{n}\right)^b / n^b$

put $if \quad n \to \infty$ **then** $\frac{1}{n} = 0$

Solving this limit we find the value c=1 so we write$f(n) = \theta(n^b)$.

OR



Solution:

$$\begin{aligned}(n+a)^b &\le (n+|a|)^b, \text{ where } n>0\\ &\le (n+n)^b \text{ for } n \ge |a|\\ &= (2n)^b\\ &= c_1 \cdot n^b, \text{ where } c_1 = 2^b\end{aligned}$$

Thus

$$(n+a)^b = \Omega(n^b). \qquad (1)$$

$$\begin{aligned}(n+a)^b &\ge (n-|a|)^b, \text{ where } n>0\\ &\ge (c_2'n)^b \text{ for } c_2' = 1/2 \text{ where } n \ge 2|a|\\ &\text{as } n/2 \le n-|a|, \text{ for } n \ge 2|a|\end{aligned}$$

Thus

$$(n+a)^b = O(n^b) \qquad (2)$$

The result follows from 1 and 2 with $c_1 = 2^b, c_2 = 2^{-b}$, and $n_0 \ge 2|a|$.

(ii) Compare the order of growth of $n + nlogn \; and \; n.n^{1/2}$

Answer: $n * n^{1/2} = n^{3/2}$

n	n+nlogn	$n^{3/2}$
1	1	1
100	300	1000
10000	50000	1000000

So we say that order of growth $n^{3/2}$ is greater than $n + nlogn$.

5. Attempt any ONE part of the following. **(1*5 = 5)**

a) Prove that quick sort algorithm takes $O(n^2)$ time to sort an array of elements in the worst case.

Answer:

The worst-case behavior for quick sort occurs when the partitioning routine produces one region with *n* - 1 elements and one with only l element. (This claim is proved in Section 8.4.1.) Let us assume that this unbalanced partitioning arises at every step of the algorithm. Since partitioning costs $\Theta(n)$ time and $T(1) = \Theta(1)$, the recurrence for the running time is

$$T(n) = T(n - 1) + \Theta(n).$$

Solving by iteration method $T(n) = O(n^2)$.

b) Write the pseudo code of HEAPSORT .Illustrate the operation of HEAPSORT on the array

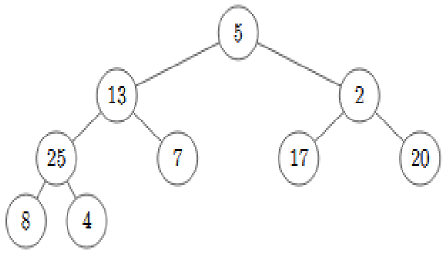
A = <5, 13, 2, 25, 7, 20, 8, 4>.

Answer:

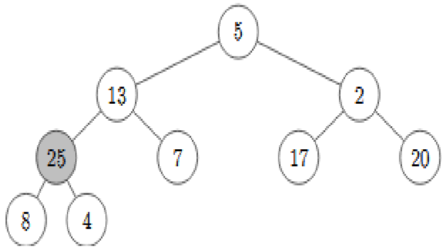
```
HEAPSORT(A)
1. BUILD-MAX-HEAP(A)
2. for i ← length[A] downto 2
3. do exchange A[1] A[i]
4. heap-size[A] ← heap-size[A] – 1
5. MAX - HEAPIFY(A, 1)
```

Since *A.length* = 9, the command MAX-HEAPIFY(A, i) is called for $i = 4, 3, 2, 1$. The action of BUILD-MAX-HEAP is as follows (these first few diagrams are not required for a correct answer), with the nodes exchanged at each step shaded:

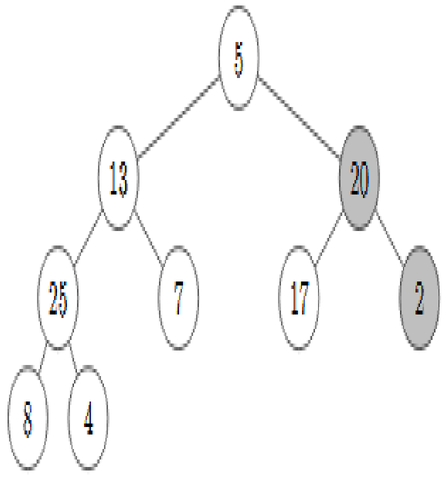
BUILD-MAX-HEAP(A):



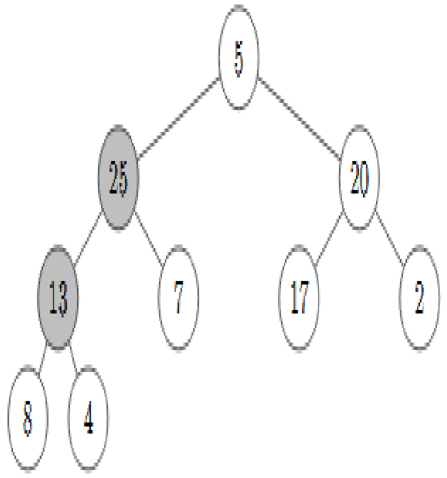
MAX-HEAPIFY(A, 4):



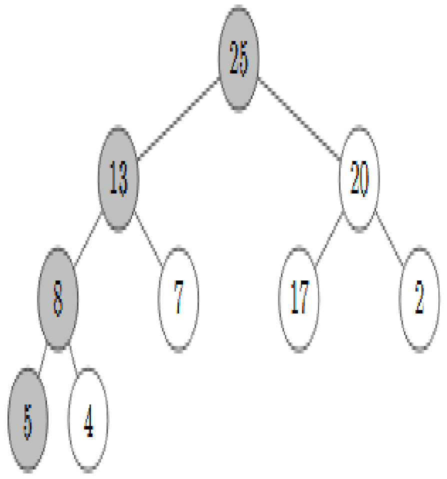
Max-Heapify$(A, 3)$:

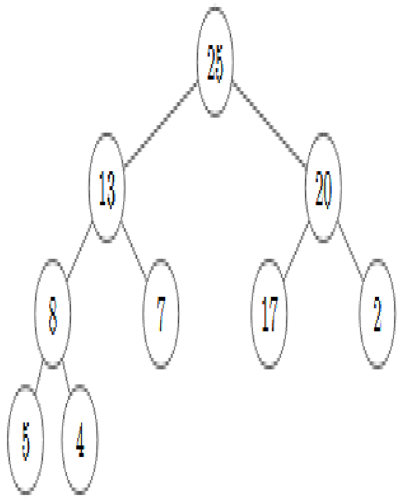


Max-Heapify$(A, 2)$:

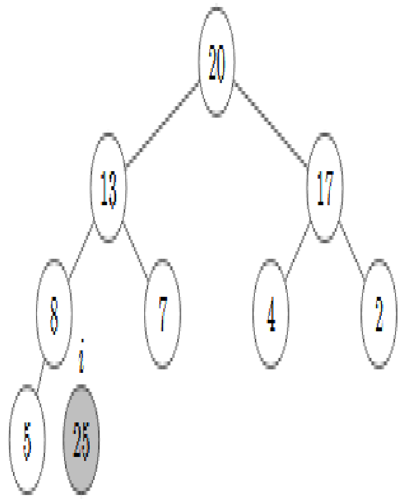


Max-Heapify$(A, 1)$:

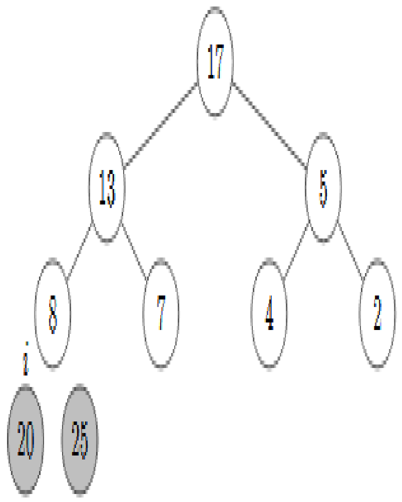




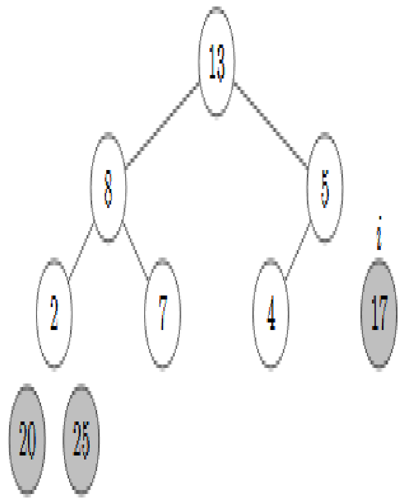
Step 0



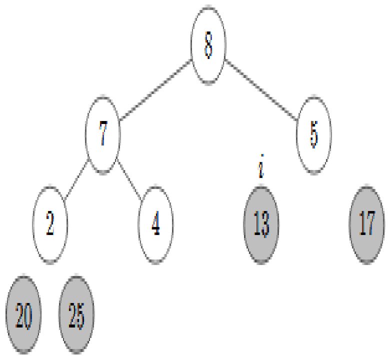
Step 1



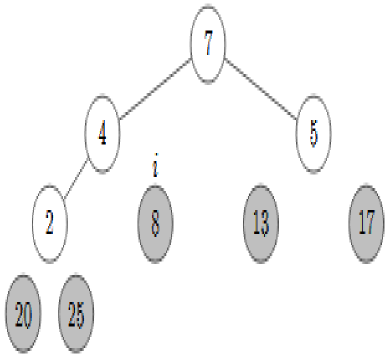
Step 2



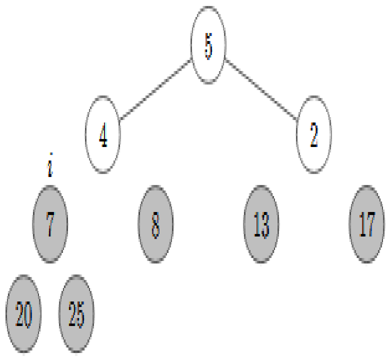
Step 3



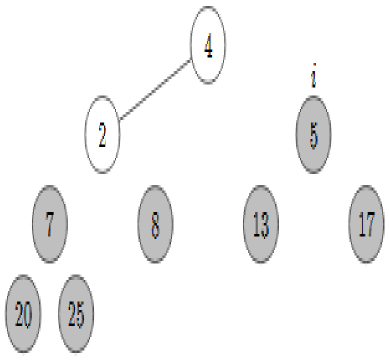
Step 4



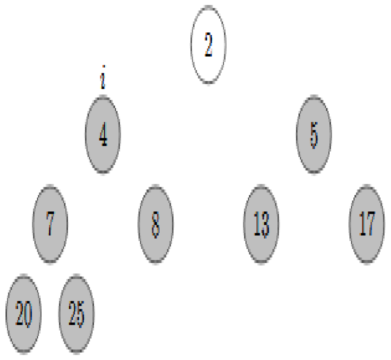
Step 5



Step 6



Step 7



Step 8

This gives a final sorted arrayA= 〈2,4,5,7,8,13,17,20,25〉