Page:
Date: / /

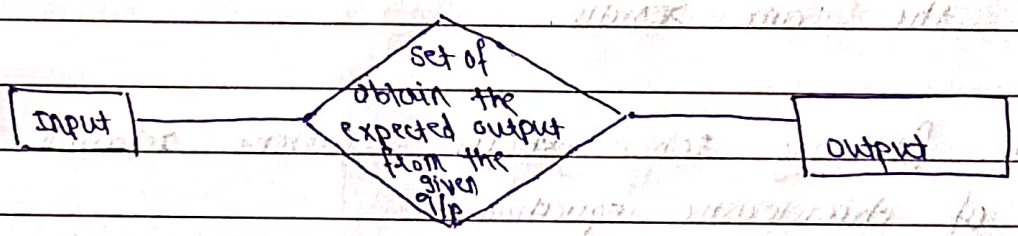
Assignment No. 1

Ques. 1) What is an algorithms ? Explain in detail about various characteristics of an algorithm.

→ ① An algorithm is a set of commands that must be followed for a computer to perform calculation or other problem-solving operations.

② An algo. is a finite set of instruction carried out in a specific order to perform a particular task.

③ It is not the entire program or code ; it is simple logic to a problem represented as an informal description in the form of a flowchart or pseudocode.



Algorithm

* Characteristics of algorithm :—→

① Input :—→

An algo. requires some input values. An algo. can be given a value other than 0 as input.

② Output :—→

At the end of the algo., you will have one or more outcomes.

③ Unambiguity :—→

A perfect algo. is defined as unambiguity, which means that its instructions should be clear and straightforward.

④ Finiteness :—→

An algorithms must be finite. Finiteness in this context means

that the algo. should have a limited number of instructions.

(5) Effectiveness :→

Because each instruction in an algo. affects the overall process, it should be adequate.

(6) Language independence :→

An algo. must be language-independent, which means that its instructions can be implemented in any language and produce the same results.

Ques. 2 a) Solve the following nonhomogenous recurrence relation by the method of characteristic equation.

$$t_n = \begin{cases} 1 & \text{if } n=0 \\ 4t_{n-1} - 2^n & \text{otherwise} \end{cases}$$

→ $t_n = 4t_{n-1} + 2^n$

$t_n - 4t_{n-1} = 2^n$

$(x-4)(x-2) = 0$

$x_1 = 4 \quad x_2 = 2$

$t_n = c_1 4^n + c_2 2^n$ ---- (1)

For $n=0$,

$c_1 + c_2 = 1$ ---- (2)

For $n=1$,

$4c_1 + 2c_2 = t_1$

$4c_1 + 2c_2 = 6$ ---- (3)

$t_n = 4t_{n-1} + 2^n$

$\therefore t_1 = 4t_0 + 2$ ... $n=1$

$= 4 \times 1 + 2$

$= 6$

from eqn ① and ③ & eqn ② × 2

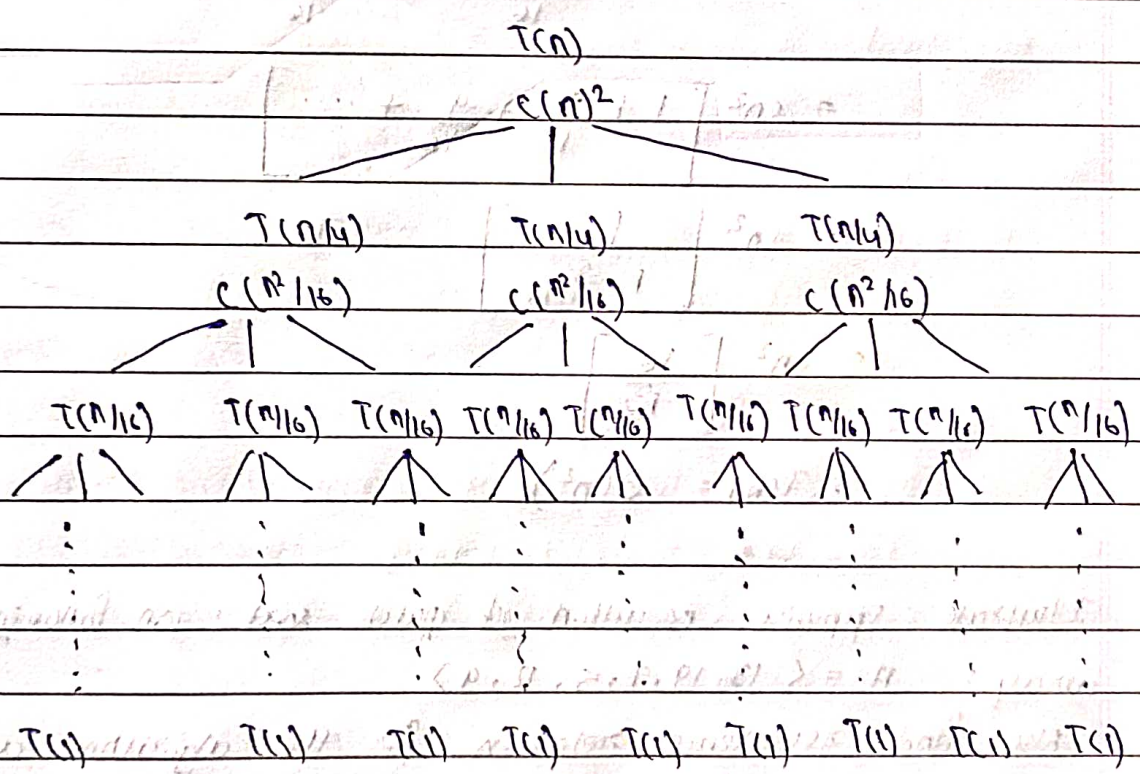
$$2c_1 + 2c_2 = 2$$
$$- \quad 4c_1 + 2c_2 = 6$$
$$-2c_1 = -4$$
$$\therefore c_1 = 2$$
$$\therefore c_2 = -1$$

Therefore, $t_n = 2 \cdot 4^n - 2^n$

$t_n = \Theta(4^n)$

Ques. 2b) Solve the following recurrence relation $T(n) = 3T(n/4) + \Theta(n^2)$ using Recursion Tree method.

→



Subproblem of depth i:

$$1 = \frac{n}{4^i}$$

$$\Rightarrow 4^i = n$$

$$i \lg 4 = \lg n$$

$$i = \lg_4 n$$

No. of nodes at particular depth = 3^i

∴ cost of node at particular depth = $\frac{cn^2}{3^i}$

∴ Total cost of particular depth = $3^i \times \frac{cn^2}{3^i}$

$= cn^2$

cost of Recussion Tree (T(n)) :→

$= (cn^2) + c\frac{3}{16}n^2 + c\frac{9}{16}n^2 + \ldots.$

$= cn^2\left[1 + \frac{3}{16} + \frac{9}{16} + \ldots\right]$

$= cn^2\left[\frac{1}{1 - 3/16}\right]$

$= cn^2\left[\frac{16}{13}\right]$

$\therefore T(n) = \theta(n^2)$

Q.3 Illustrate stepwise execution of quick sort on following input array : $A = \langle 13, 19, 7, 5, 12, 9 \rangle$

Also find recurrence relation for the algorithm and analyze it's time complexity.

→ Algo :

```
Quicksort (A,P,r)
{ if (p<r) then
    q ← partition (A,P,r)
    Quicksort (A, P, r-1)
    Quicksort (A, q+1, r)
}
```

Page:
Date: / /

partition:

```
partition (A,p,r)
    x ← A(r)
    i ← p-1
    for j ← p to r-1
        if A(j) <= x
            i ← i+1
                exchange (A(i), A(j))
    exchange (A(i+1), A(r))
    return i+1
```

A = <13, 19, 7, 5, 12, 9>

level 1: Divide : phase 1

A	13	19	7	5	12	9
index	1	2	3	4	5	6
	p					r

Quicksort (A, 1, 6)

if (1 < 6) ... True

q ← partition (A, 1, 6)

q = 3

Quick sort (A, 1, 2)

Quick sort (A, 4, 6)

Step 1: Set pointer and pivot

A	13	19	7	5	12	9	pivot 9
index	1	2	3	4	5	6	r

partition: partition (A, 1, 6)

x ← A(r)

$x \leftarrow A(6) \leftarrow 9$

$x = 9$

$i \leftarrow p-1$ ie. $i = 1-1 = 0$

Step 2: iteration

A	13	19	7	5	12	9	9
index	1	2	3	4	5	6	x
	* p					x	pivot

for $j=1$:

if $A(j) <= x$ ie. $13 <= 9$ ... False

No need for increment and exchange

for $j=2$:

if $A(j) <= x$ ie. $19 <= 9$ ... false

No need for increment and exchange

for $j=3$:

if $A(j) <= x$ ie. $7 <= 9$ ... True

$i \leftarrow i+1$ ie. $i = 0+1 = 1$

exchange $(A(i), A(j))$ ie. exchange $(A(1), A(3))$

$A(1) = 7$

$A(3) = 13$

A	7	19	13	5	12	9	9
index	1	2	3	4 *	5	6	x
	p					x	pivot

for $j=4$:

if $A(j) <= x$ ie. $5 <= 9$ ... True

$i \leftarrow j+1$ ie. $1+1 = 2$

exchange (A(i), A(j)) ie. exchange (A(7), A(4))

A(2) = 5 , A(4) = 13

A	7	5	13	19	12	9	p
index	1	2	3	4	5	6	r
	p					r	pivot

for j=5:

if A(j) <= x ie. 12 <= 9 ... false

No need to increment and exchange.

Step 3: exchange pivot with A(i+1)

exchange (A(i+1), A(r))

ie. exchange (A(2+1), A(6))

exchange (A(3), A(6))

A(3) = 9 ; A(6) = 13

return i+1 = 2

A	7	5	13	19	12	9	p
index	1	2	3	4	5	6	x
	p					r	pivot

Phase 2: Conquer :→

A	7	5	9	19	12	13
index	1	2	3	4	5	6

level 2: phase 1 (Divide) :→

A	7	5	9	19	12	13
index	1	2	3	4	5	6

Quicksort (A,1,2) — True

Quicksort (A,4,6) — True

A	5	7	12	13	19
index	1	2	4	5	6

phase 2: Conquer:

A	5	7	12	13	19
index	1	2	4	5	6

ie.

A	5	7	9	12	13	19
index	1	2	3	4	5	6

level 3 phase 1 (Divide):

A	5	7	12	13	19
index	1	2	4	5	6

Quicksort (A,2,2) — false

Quicksort (A,4,4) — False

Quicksort (A,6,6) — False

phase 2 : Conquer :

A	5	7	12	13	14
index	1	2	4	5	6

ie.

A	5	7	9	12	13	14
index	1	2	3	4	5	6

Therefore, sorted array ⇒ 5, 7, 9, 12, 13, 14

Analysis :-

Best case : Each partitioning state divide the array into exactly two halfs.

Therefore, the recurrence relation is given as

$$T_n = \begin{cases} \theta(1) & \text{if } n=1 \\ 2T\left(\frac{n}{2}\right) + \theta(n) & \text{if } n>1 \end{cases}$$

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

worst case : when array is already sorted.

If the array is already sorted then partition produce split the array of size

$(n-8), (n-4), (n-2)$ ... and so on

∴ The recurrence relation given as

$$T(n) = n + (n+4) + (n+2) + \dots$$

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

$$T(n) = 0.5\ (n^2 + n - 2)$$

$$\therefore T(n) = \theta(n^2)$$

Page:
Date:

Ques.4 Illustrate stepwise execution of heap sort on following input array:

A = <5, 13, 2, 25, 7, 17, 20, 8, 4>

Also find recurrence relation for the algorithm and find its time complexity.

→ Algorithm for heap sort:

```
Heapsort (A)
    Buildheap (A)
    for (i ← n to 1 do)
        swap (A[1], A[i])
        Heapify (A, 1)
```

Algorithm for Buildheap:

```
Buildheap (A)
    n ← |A|
    for (i ← ⌊n/2⌋ to 1 do)
        Heapify (A, i)
```

Algorithm for heapify:

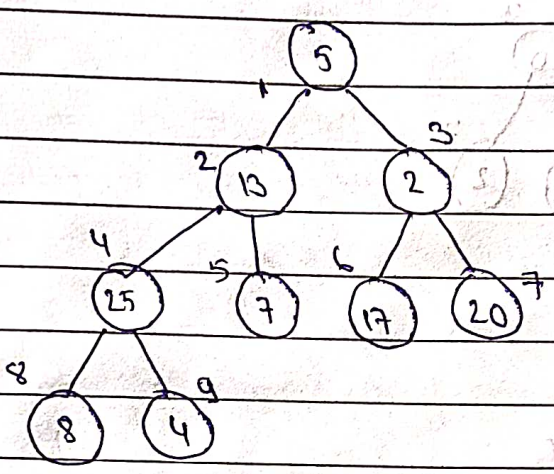
```
Heapify (A, i)
    l ← left (i)
    r ← right (i)
    if (l <= heapsize (A) && A(l) > A(i))
            then largest ← l
            else largest ← i
    if (r <= heapsize (A) && A(i) > largest)
            then largest ← r
    if (largest ≠ i)
            then exchange (A[i], A[largest])
                 Heapify (A, largest)
```

Page: _____
Date: ___/___/___

A =

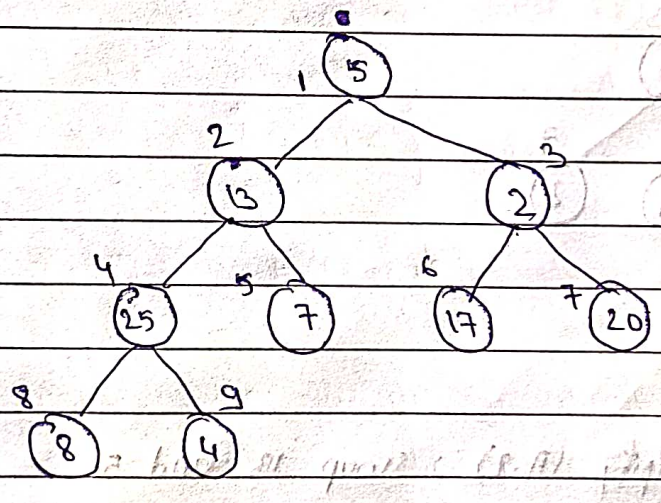
1	2	3	4	5	6	7	8	9
5	13	2	25	7	17	20	8	4

corresponding array is complete binary tree is :

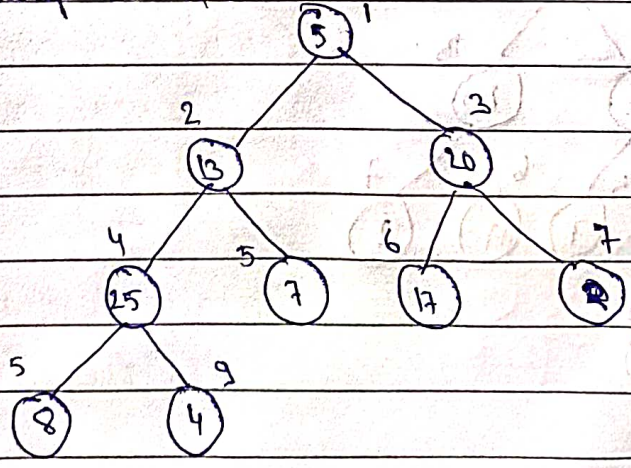


Now build max-heap from above array

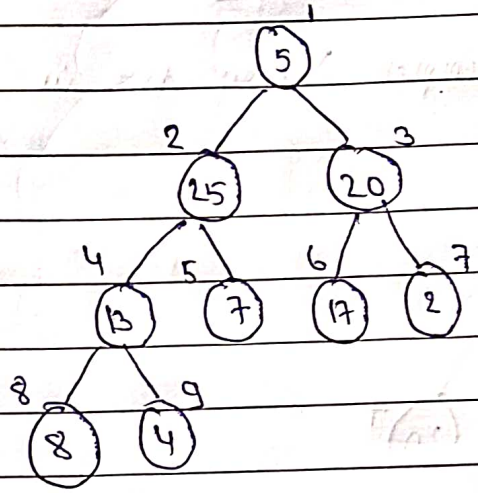
heapify 4: Now swap



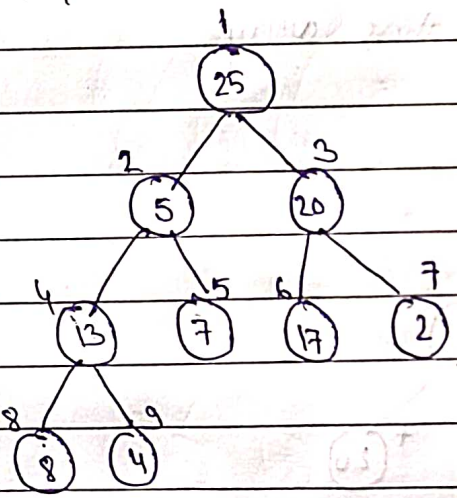
Heapify 3: swap 20 & 2



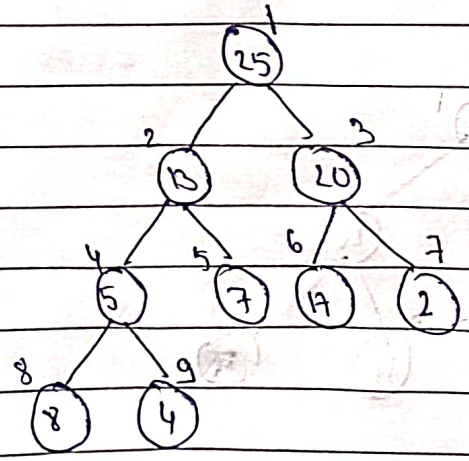
Heapify 2 : swap 25 and 13



Heapify 1 : swap 25 and 5

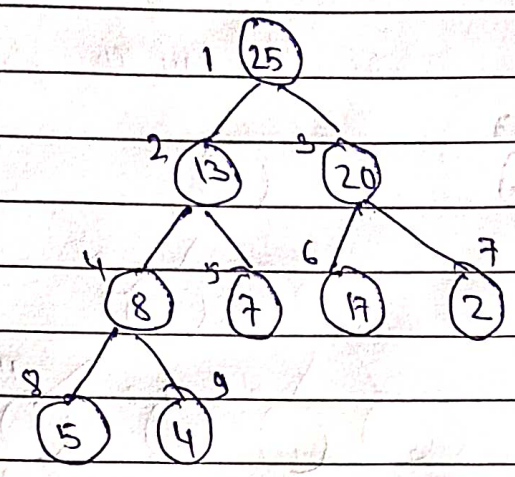


Again calling heapify (A,2) : swap 13 and 5



Page:
Date: / /

Finally calling heapify (A, 4) : swap 8 and 5

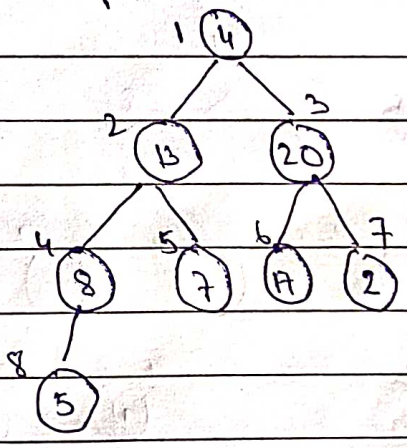


Now, we applying heap sort

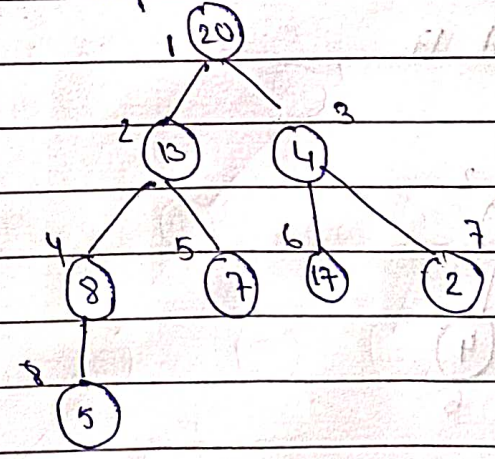
Sorted array :

25	20	17	13	8	7	5	4	2
1	2	3	4	5	6	7	8	9

i = 9 : swap 4 and 25

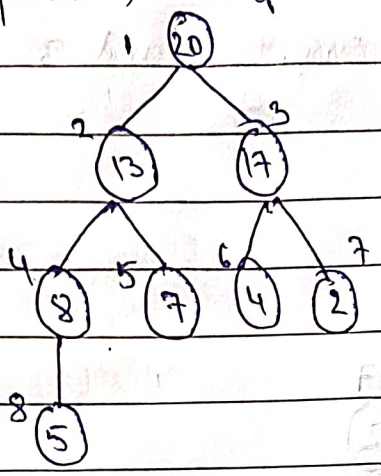


heapify (A, 1) : swap 20 and 4

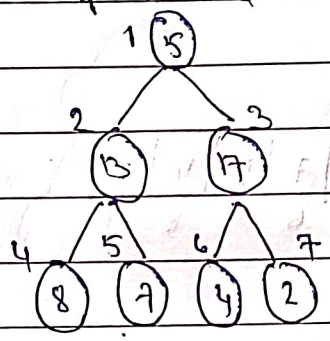


Page:
Date:

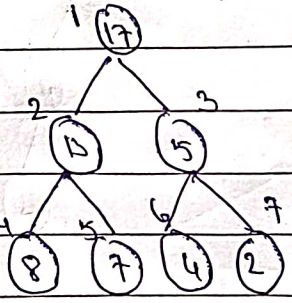
Heapify (A, 3) : swap 17 and 4



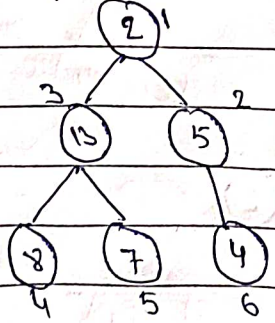
i = 8 : swap 5 and 20



Heapify (A, 1) : swap 17 and 5



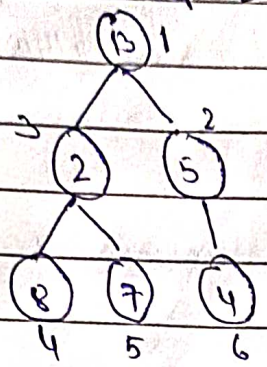
i = 7 : swap 2 and 17



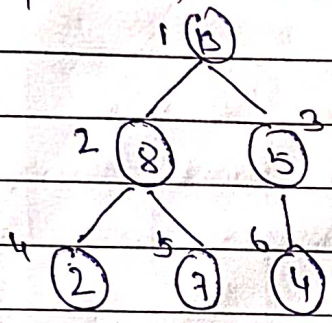
Page: _____

Date: ___/___/_____

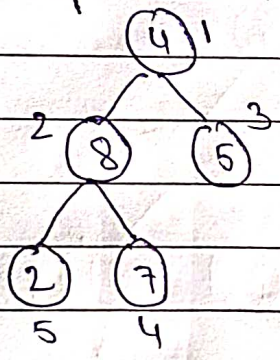
Heapify (A,1) : swap 2 and 13



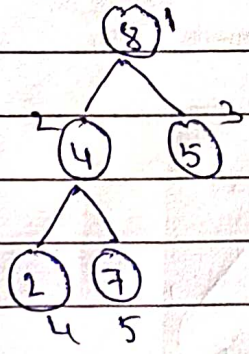
Heapify (A,3) : swap 8 & 2



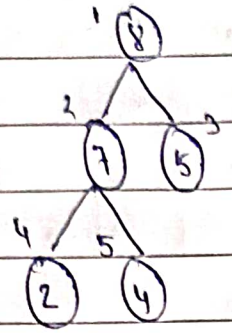
i = 6 : swap 4 and 8



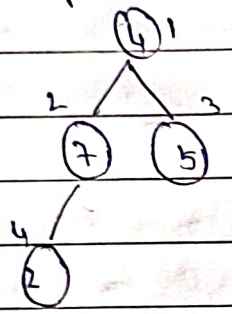
Heapify (A,1) : swap 8 & 4



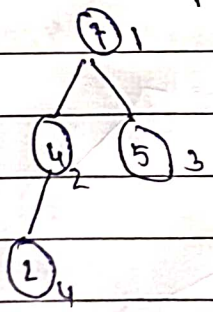
Heapify (A,2) : swap 4 and 7



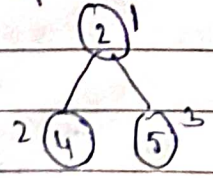
i=5 : swap 4 and 8



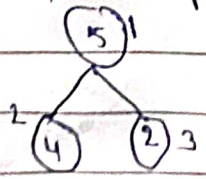
Heapify (A,1) : swap 4 and 7



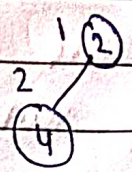
i = 4 : swap 2 and 7



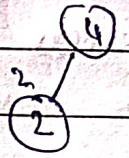
Heapify (A,1) : swap 2 and 5



i=3 : Swap 2 and 5



Heapify (A,2) : Swap 4 and 2



i=2 : Swap 2 and 4

i=1 : Swap 1 and 1

Now final sorted array we get

A=	25	20	17	13	8	7	5	4	2
	1	2	3	4	5	6	7	8	9

Recurrence relations :

$$T(n) = T\left(\frac{2n}{3}\right) + O(1)$$

Time complexity:

Worst case : $O(n \log n)$

Best case : $O(n \log n)$

Average case : $O(n \log n)$

Ques:5 What is minimum spanning tree? write prim's algorithm for finding minimum cost spanning tree. Also give stepwise illustration of this algorithm using suitable example.

→ * Minimum spanning tree: →

It is subset of edges of connected weighted undirected graph that connects all vertices together with minimum possible edges weight.

* prim's algorithm

Step 1: Determine arbitary starting vertex select it.

Step 2: Keep repeating step 3 and 4 until fringe vertices

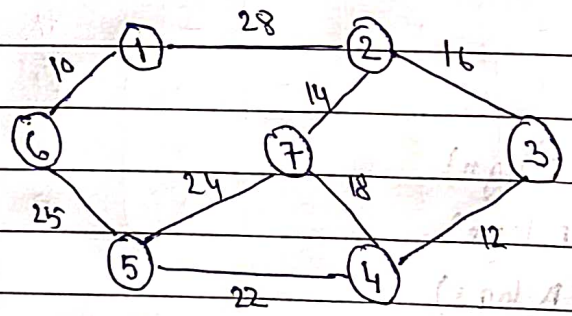
Step 3: Select an edge connecting tree vertex and fringe vertex having minimum weight.

Step 4: Add chosen edge to mst if it doesn't form any closed cycle

Step 5: END of loop.

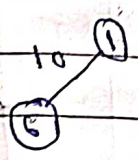
Step 6: Exit.

* Example:

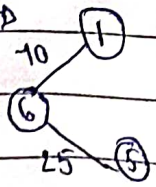


Total weight = 169

Step 1: →

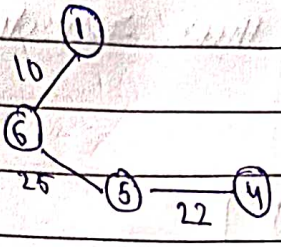


Step 2: →

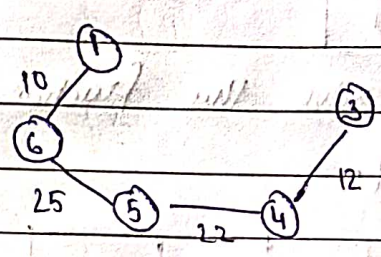


Page: _____
Date: ____/____/____

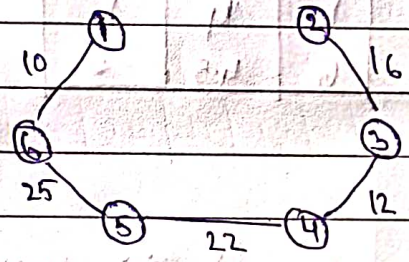
Step 3: →



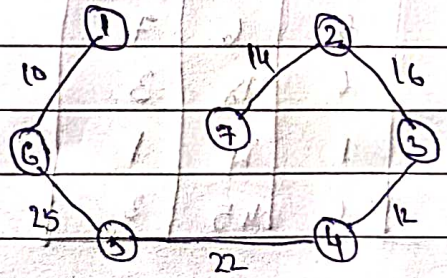
Step 4: →



Step 5: →



Step 6: →



Total weight = 10 + 25 + 22 + 12 + 16 + 14

= 99 units

Page:
Date:

Ques. 6 Consider 5 items along with their respective weight and value as follows:

items	1	2	3	4	5	6	7
value	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1

Capacity of knapsack w = 15. Solve the fractional knapsack problem using Greedy strategy.

→ Step 1:

item	1	2	3	4	5	6	7
value (Pi)	10	5	15	7	6	18	3
weight (wi)	2	3	5	7	1	4	1
Pi/wi = P	5	1.67	3	1	6	4.5	3

w=15

Step 2: Sort the item in descending order of ~~the~~ unit weight (on basis of P)

Before sorting:

items	1	2	3	4	5	6	7
value	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1
P	5	1.6	3	1	6	4.5	3

After sorting:

items	1	2	3	4	5	6	7
value	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1
P	6	5	4.5	3	3	4.5	1

Page:
Date:

Step 3: pack the knapsack using greedy strategy to maximize profit.

Iteration - 1 :→

amount = min (15, 2)

amount ← min (w, w(i))

amount ← min (15, 2) = 2

Solution ← amount

Solution = 2

w ← w ← amount

w ← 15 - 2 = 13

if i+1=2

i	wi	w	amount	soln
-	-	15	-	-
1	2	13	2	2

Iteration 2 :→

amount ← min (13, 3) = 3

Solution ← 3

w ← 13 - 3 = 10

i ← i+1 = 3

i	wi	w	amount	soln
-	-	15	-	-
1	2	13	2	2
2	3	10	3	3

Iteration 3 :→

amount ← min (10, 5) = 5

Soln ← amount = 5

i ← 10 - 5 = 5

i ← i+3 = 4

i	wi	W	Amount	soln
-	-	15	-	-
1	2	13	2	2
2	3	10	3	3
3	5	5	5	5

iteration 4:

amount ← min (5,7) = 5

Solution ← amount = 5

W ← 5-5 = 0

i ← i+1 = 5

i	wi	W	Amount	soln
-	-	15	-	-
1	2	13	2	2
2	3	10	3	3
3	5	5	5	5
4	7	0	5	5

iteration 5 :

i ← i+1

while (w > 0)

amount ← min (W, w(i))

Solution ← amount

W ← W - amount

i ← i+1

return solution

w > 0 - false

Exceeded the capacity of knapsack

i	wi	W	Amount	xi
–	–	15	–	–
1	2	13	2	2
2	3	10	3	3
3	5	5	5	5
4	7	0	5	5
5				

Overall step - 3

Step 3: pack the knapsack using greedy strategy to maximize profit.

profit = (2×5) + (3×1.6) + (5×3) + (5×1)

= 10 + 4.8 + 15 + 5

= 34.8