Namra Joshi

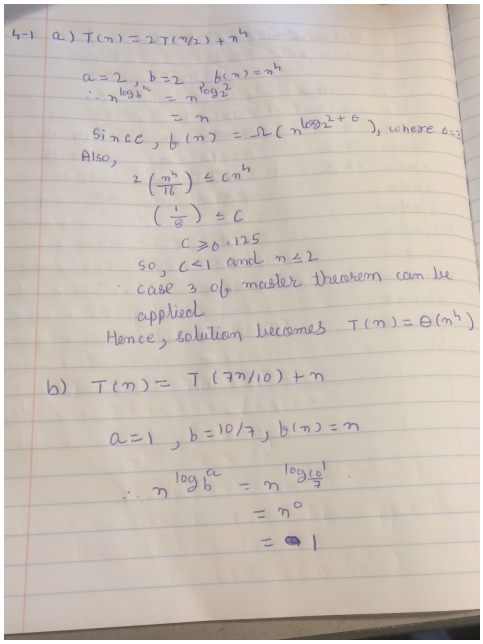
Professor Mortezaie

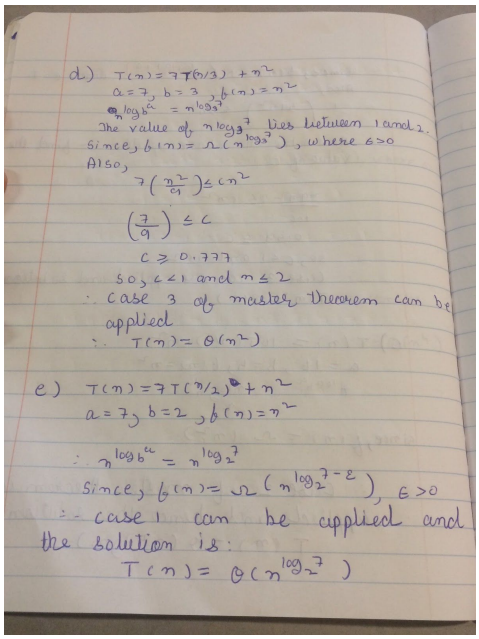
CS 146-04

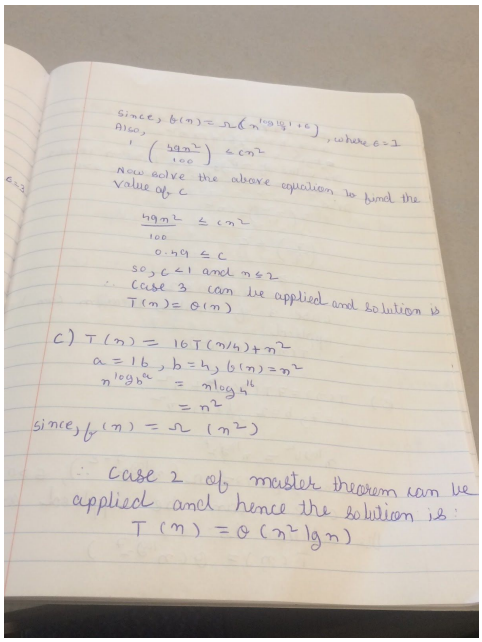
October 2, 2019

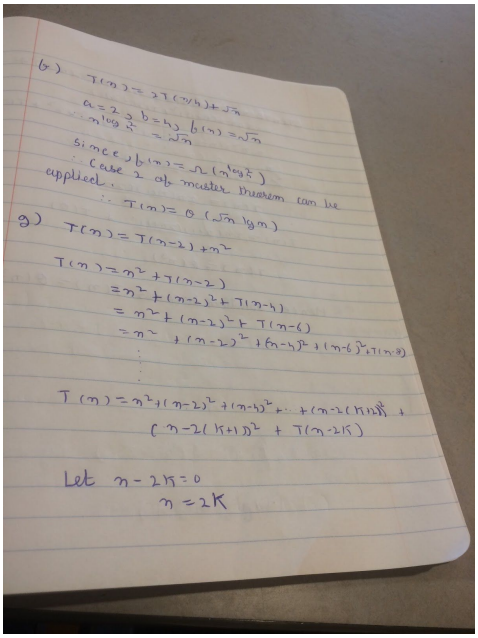
Homework 2

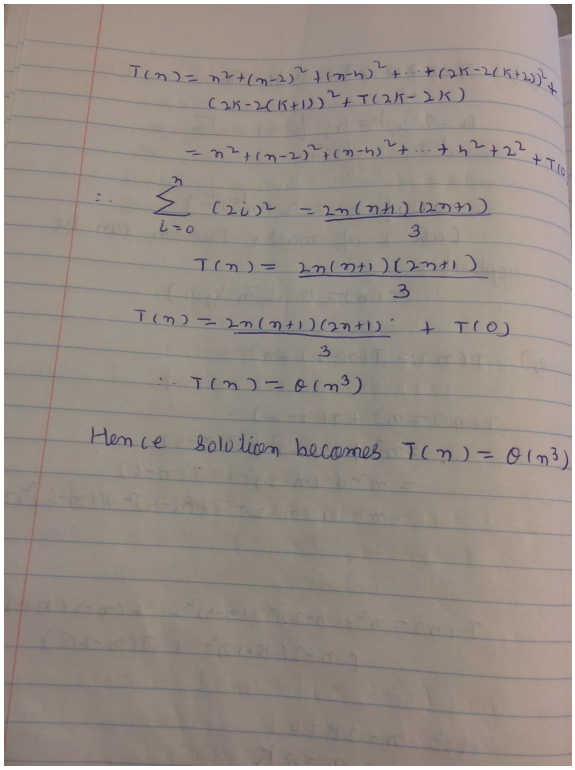
Problem 1:



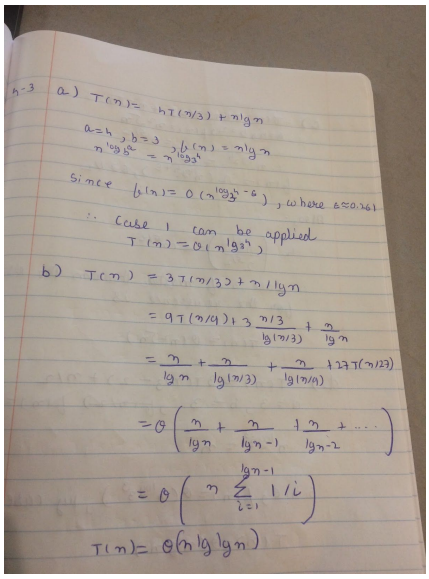


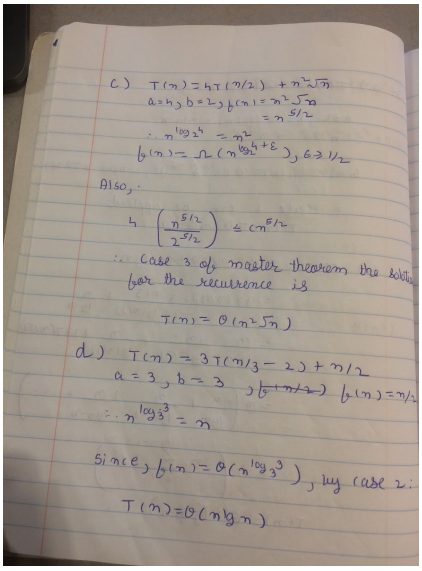

Since, $f(n) = \Omega(n^{\log_b a + \epsilon})$, where $\epsilon = 1$
Also,
$\left(\frac{49n^2}{100}\right) \leq cn^2$
Now solve the above equation to find the value of c
$\frac{49n^2}{100} \leq cn^2$
$0.49 \leq c$
So, $c < 1$ and $n \geq 2$
Case 3 can be applied and solution is
$T(n) = \Theta(n)$
c) $T(n) = 16T(n/4) + n^2$
$a = 16, b = 4, f(n) = n^2$
$n^{\log_b a} = n^{\log_4 16}$
$= n^2$
Since, $f(n) = \Omega(n^2)$
∴ Case 2 of master theorem can be applied and hence the solution is:
$T(n) = \Theta(n^2 \lg n)$


b) $T(n) = 2T(n/4) + \sqrt{n}$
$a = 2, b = 4, f(n) = \sqrt{n}$
$n^{\log_b a} = \sqrt{n}$
Since, $f(n) = \Omega(n^{\log_b a})$
∴ Case 2 of master theorem can be applied.
∴ $T(n) = \Theta(\sqrt{n} \lg n)$
g) $T(n) = T(n-2) + n^2$
$T(n) = n^2 + T(n-2)$
$= n^2 + (n-2)^2 + T(n-4)$
$= n^2 + (n-2)^2 + T(n-6)$
$= n^2 + (n-2)^2 + (n-4)^2 + (n-6)^2 + T(n-8)$
$T(n) = n^2 + (n-2)^2 + (n-4)^2 + \cdots + (n-2(k-1))^2 + (n-2k)^2 + T(n-2k)$
Let $n - 2k = 0$
$n = 2k$


$T(n) = n^2 + (n-2)^2 + (n-4)^2 + \cdots + (2k-2(k+2))^2 + (2k-2(k+1))^2 + T(2k-2k)$
$= n^2 + (n-2)^2 + (n-4)^2 + \cdots + 4^2 + 2^2 + T(0)$
$\therefore \sum_{i=0}^{n} (2i)^2 = \frac{2n(n+1)(2n+1)}{3}$
$T(n) = \frac{2n(n+1)(2n+1)}{3}$
$T(n) = \frac{2n(n+1)(2n+1)}{3} + T(0)$
$\therefore T(n) = \Theta(n^3)$
Hence solution becomes $T(n) = \Theta(n^3)$

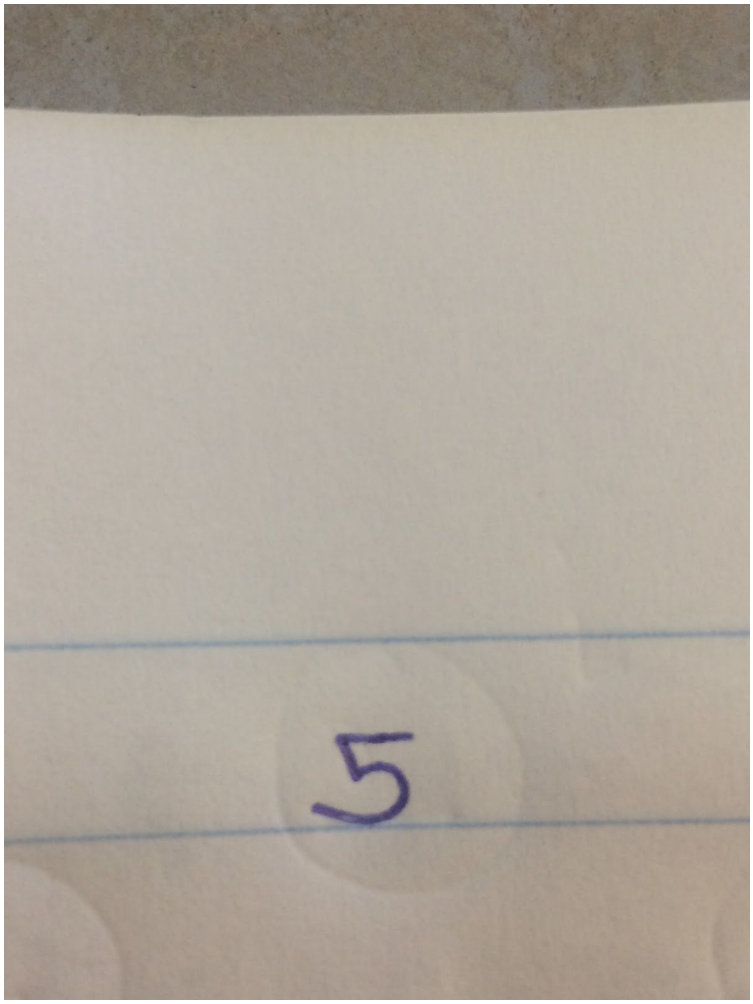
Problem 2



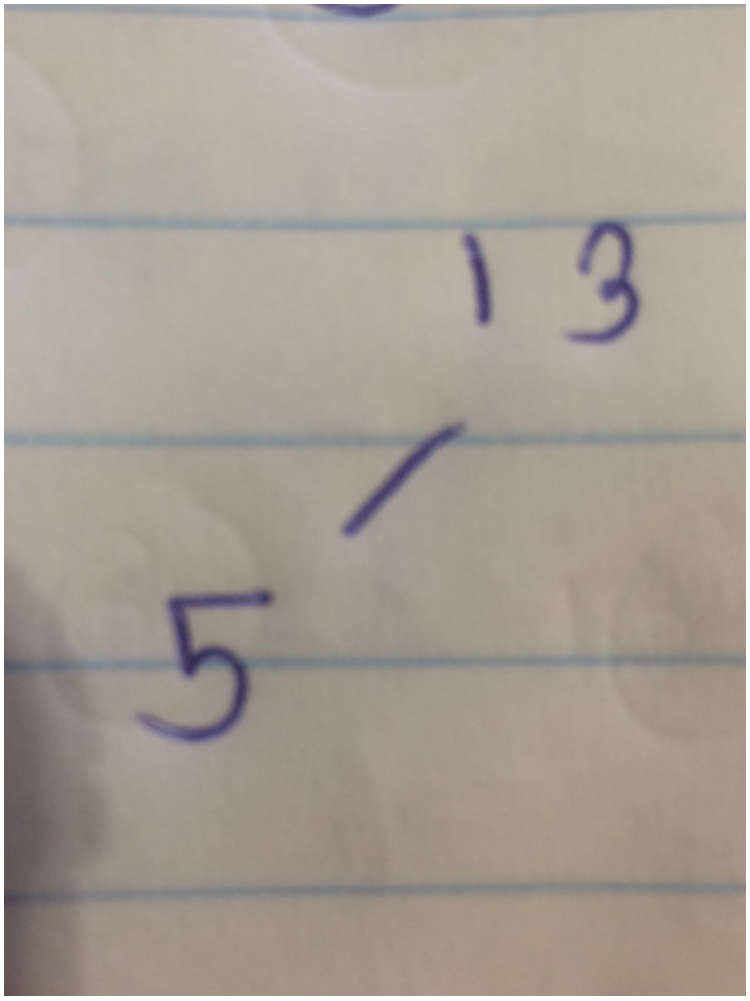


Problem 3

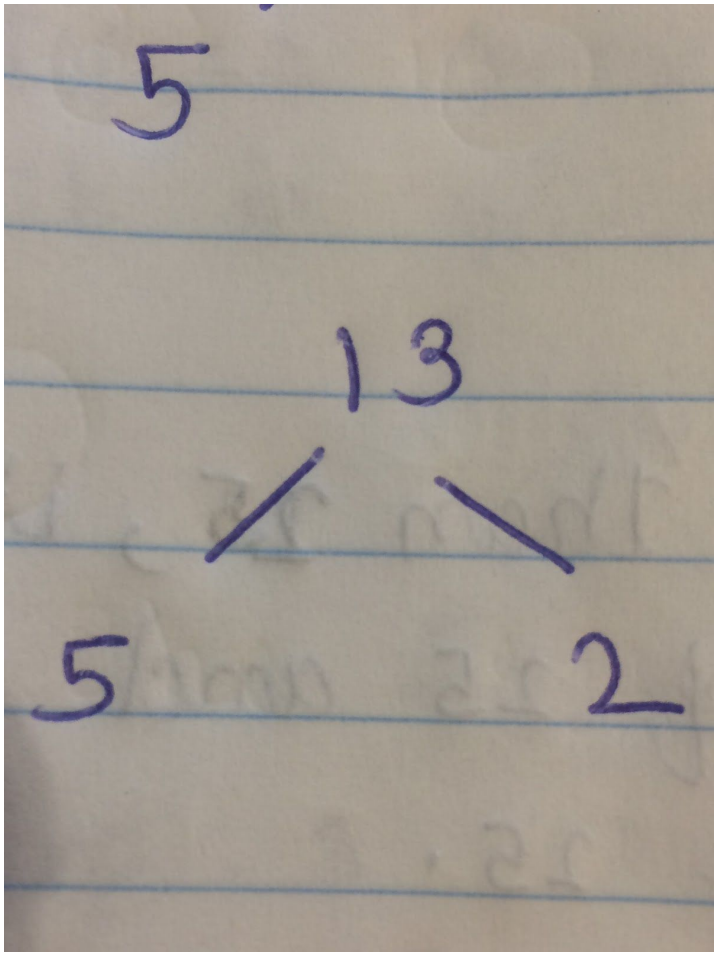
The construction of the heap for the array A=<5, 13, 2, 25, 7, 17, 20, 8, 4 > is as shown below:
Insert 5:



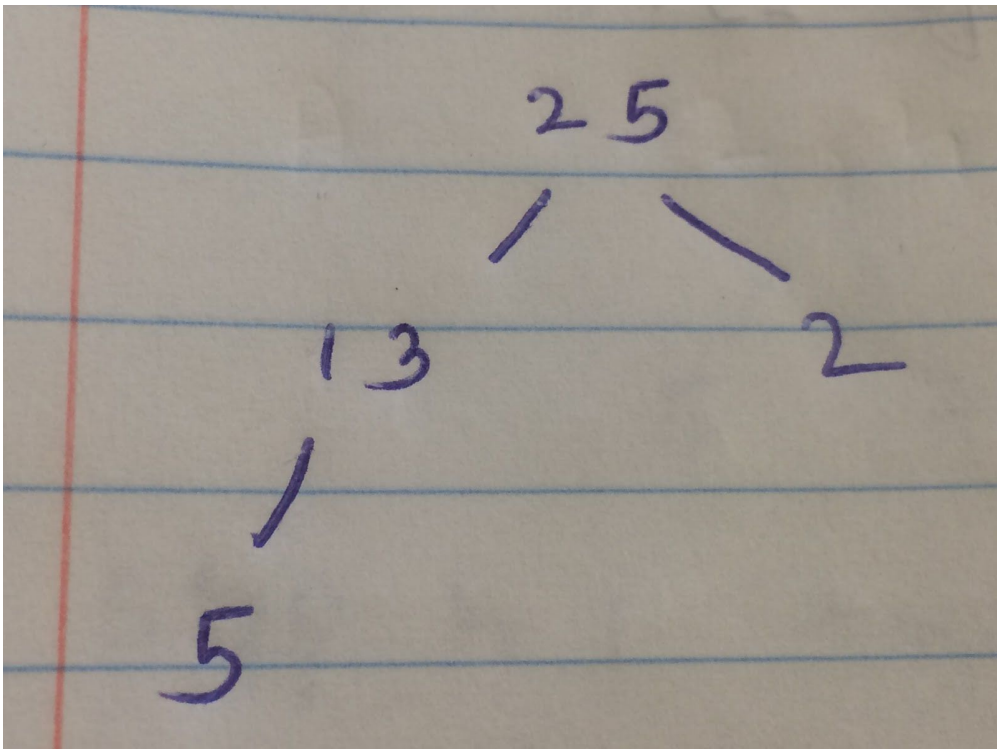
Insert 13: As 13 is greater than 5, it is added as the root node. As 5 is less than 13, it is added as the left node of 13.



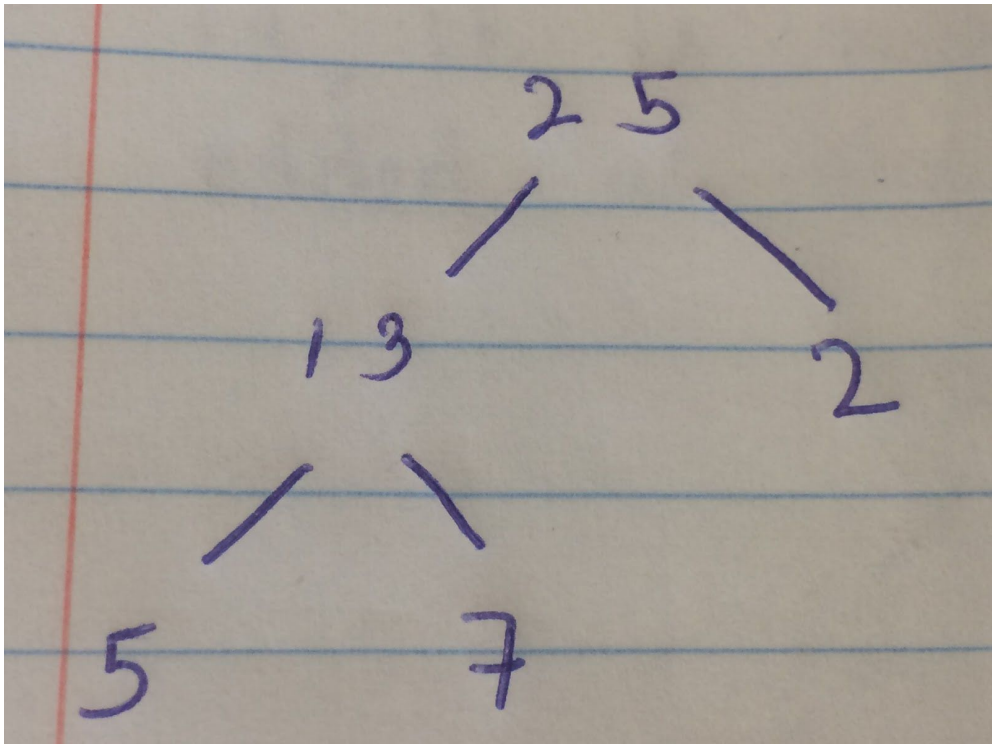
Insert 2: It is added as the right node of 13.



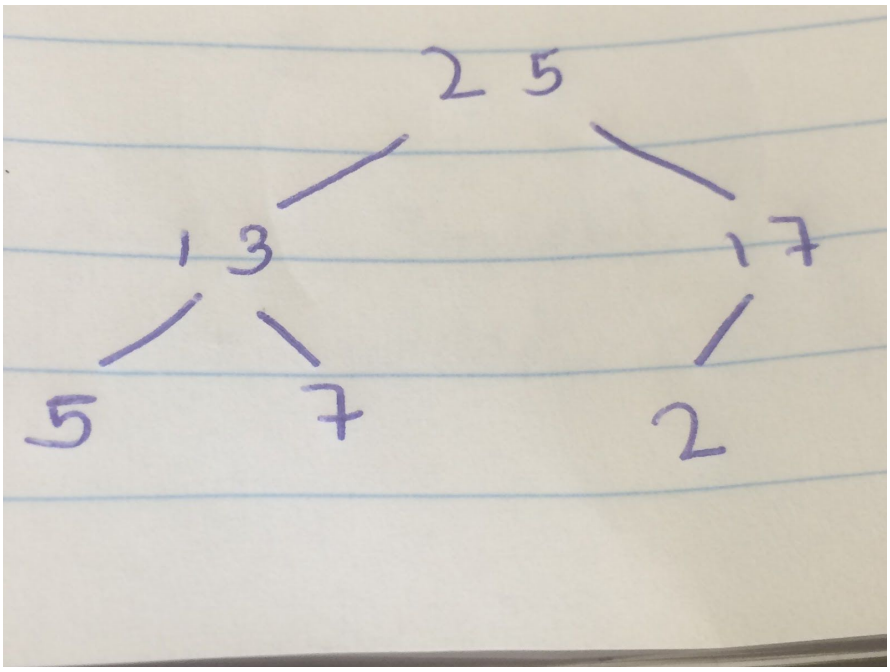
Insert 25: As 25 is greater than 13, it is added as the root node and 13 is added as the left node of 25.



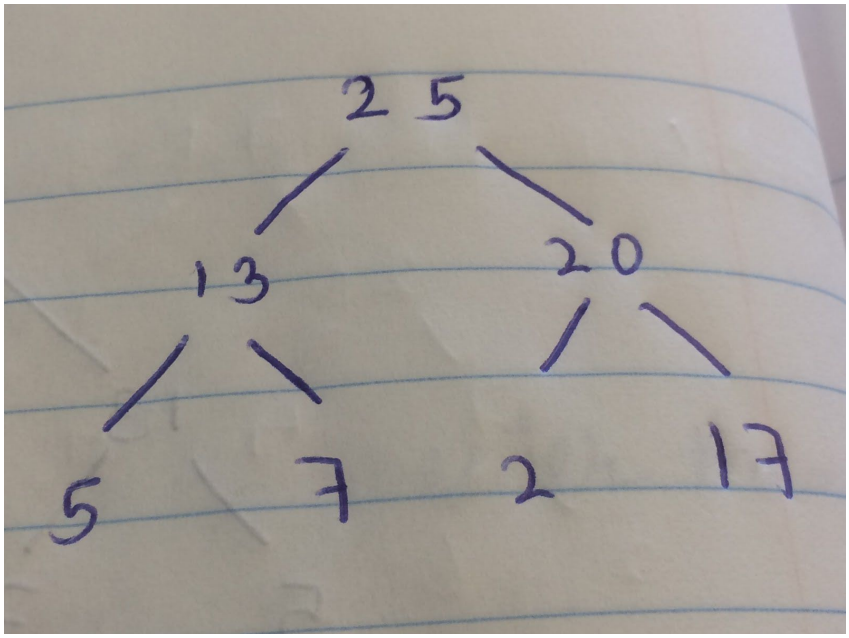
Insert 7: As 7 is less than 13, so it is added as the left node of 13.



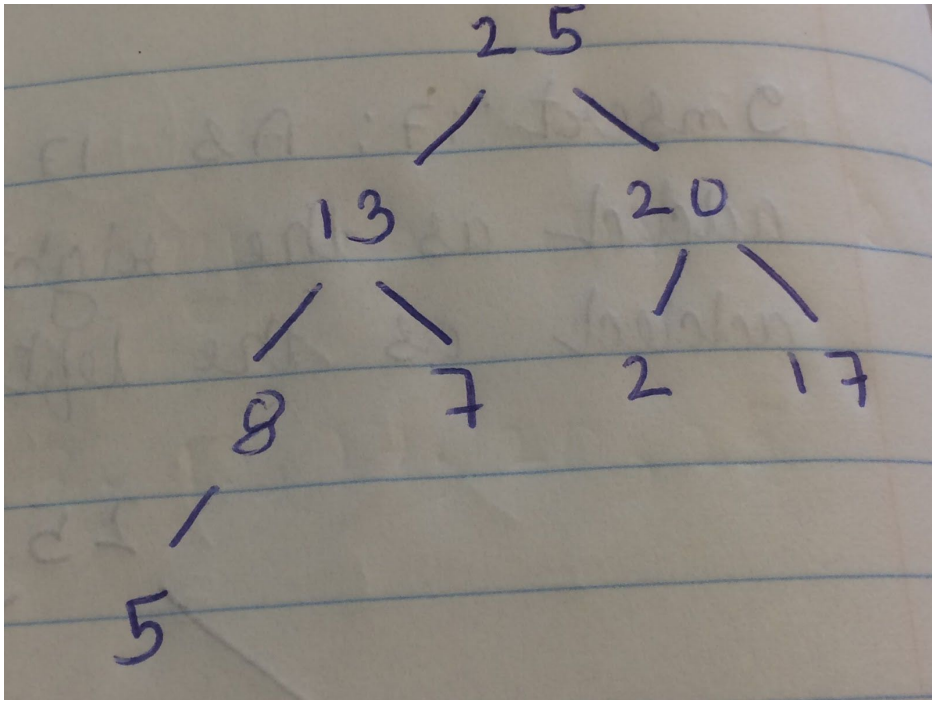
Insert 17: As 17 is less than 25, it is added as the right node of 25 and 2 is added as the left node of 25.



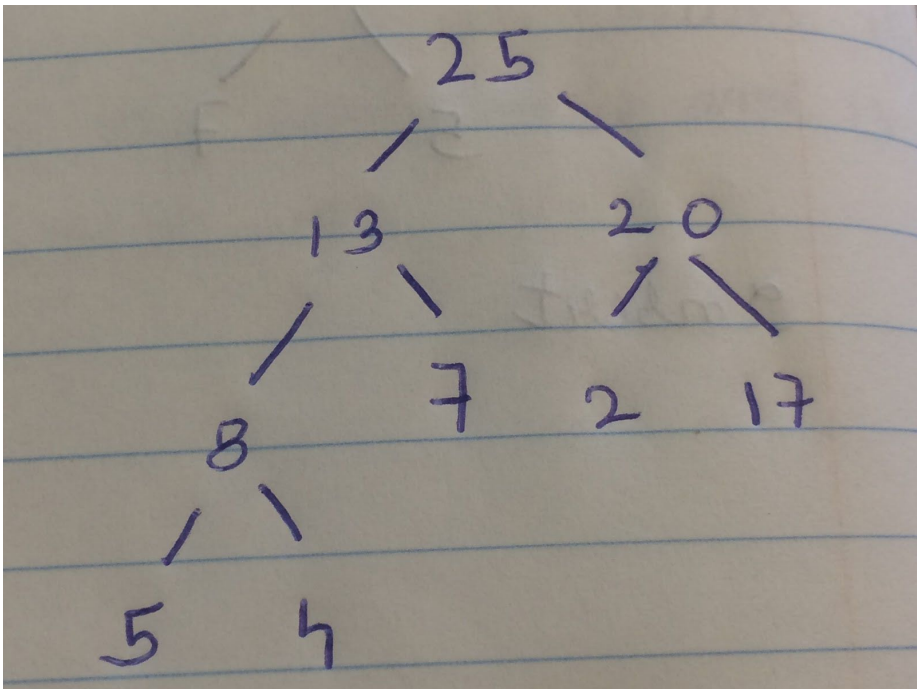
Insert 20: As 20 is greater than 17, it is added as the left node of 25 and 17 is added as the right node of 20.



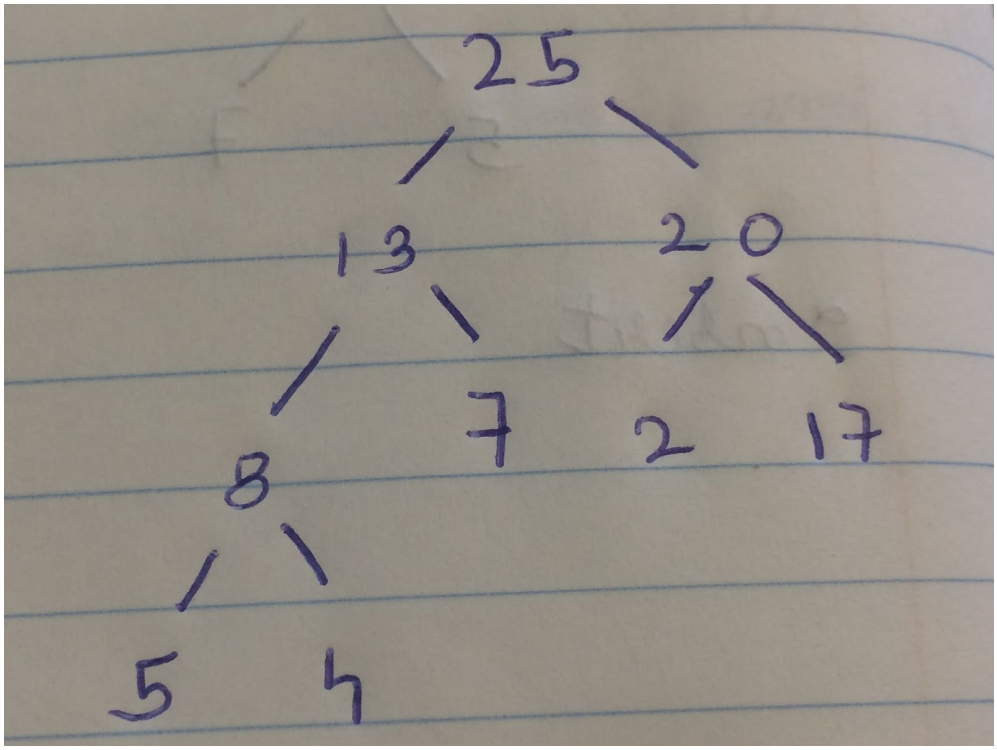
Insert 8: 8 is added as the left node of 13 as 8 is greater than 5. Accordingly, 5 is changed as the left node of 8.



Insert 4: 4 is added as the right node of 8.



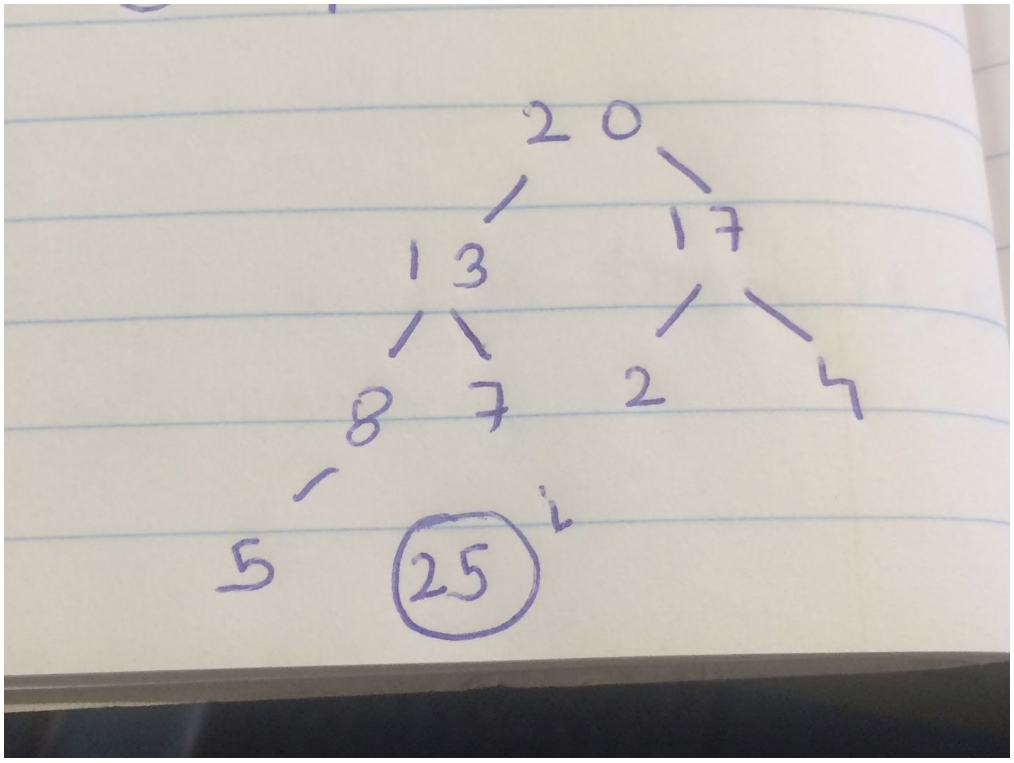
The final heap tree is as shown below:



The max-heap after each call of MAX-HEAPIFY is as shown below:

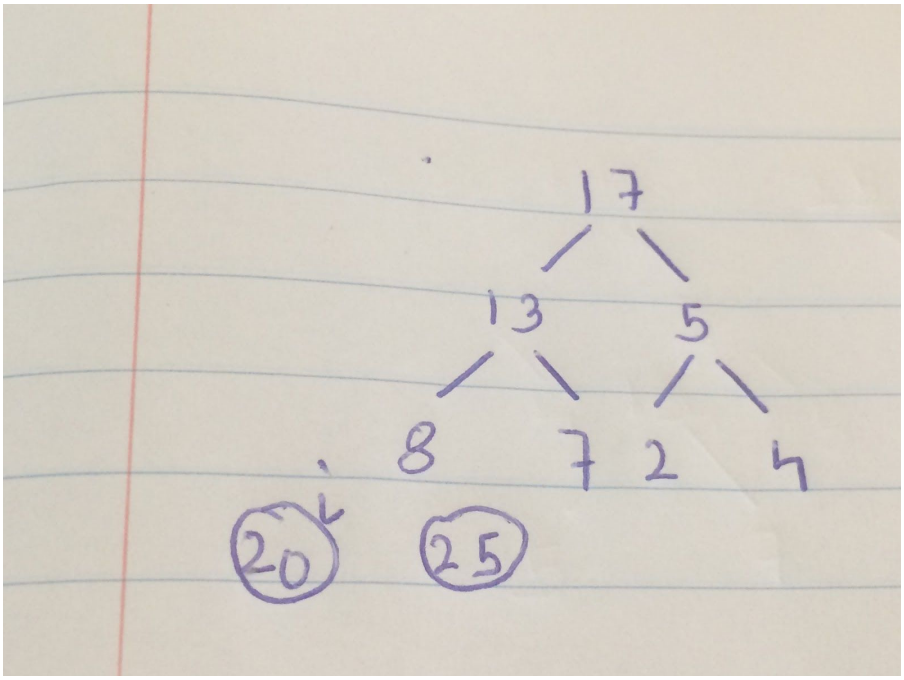
Root node 25:

The node 25 is swapped with the last node 4. As 4 is less than 20, and 17, 20 becomes the root node and 17 becomes the right child of 20 and 4 becomes the right child of 17. The resultant tree is as shown below:



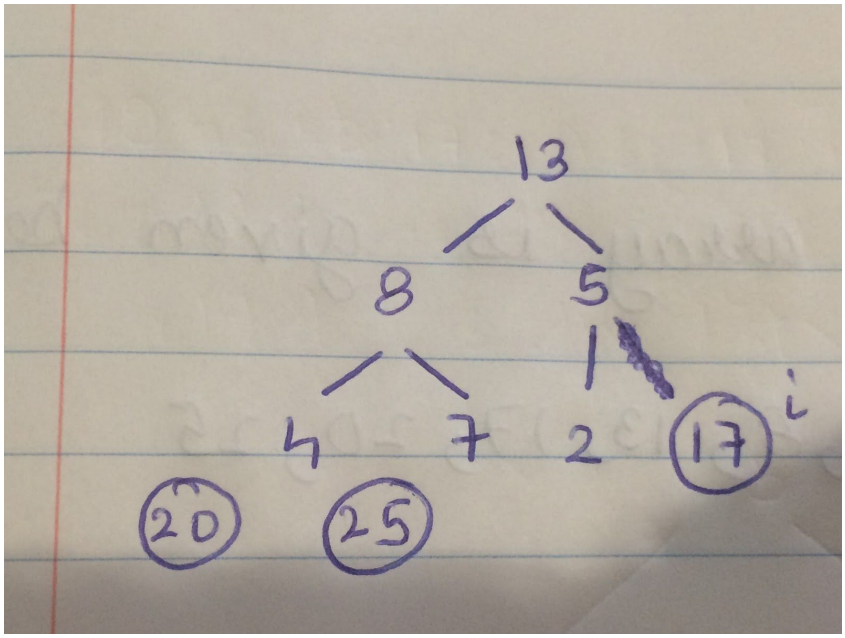
Root node 20:

The node 20 is swapped with the last node 5. As 5 is less than 17, 17 becomes the root node and 5 becomes the right child of 17. The resultant tree is as shown below:



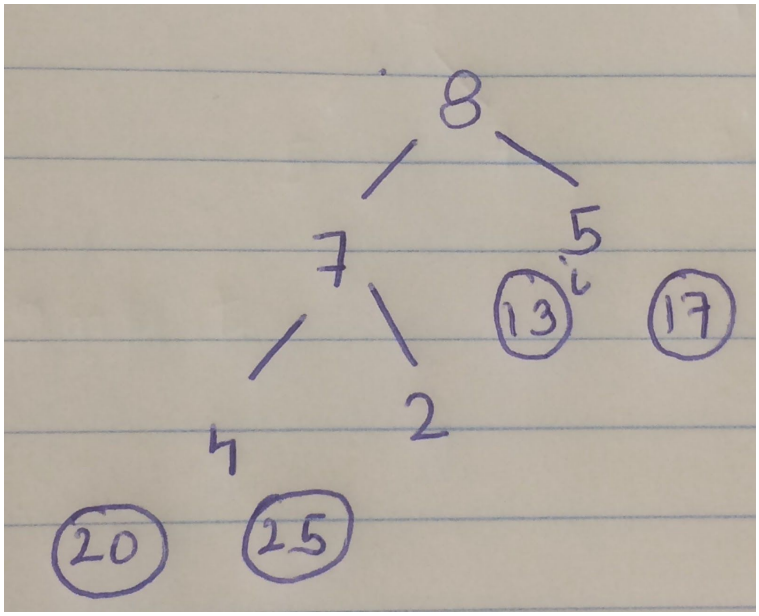
Root node 17:

The node 17 is swapped with the last node 4. As 4 is less than 13 and 8, 13 becomes the root node and 8 becomes the left child of 13. 4 becomes the left child of 8. The resultant tree is as shown below:



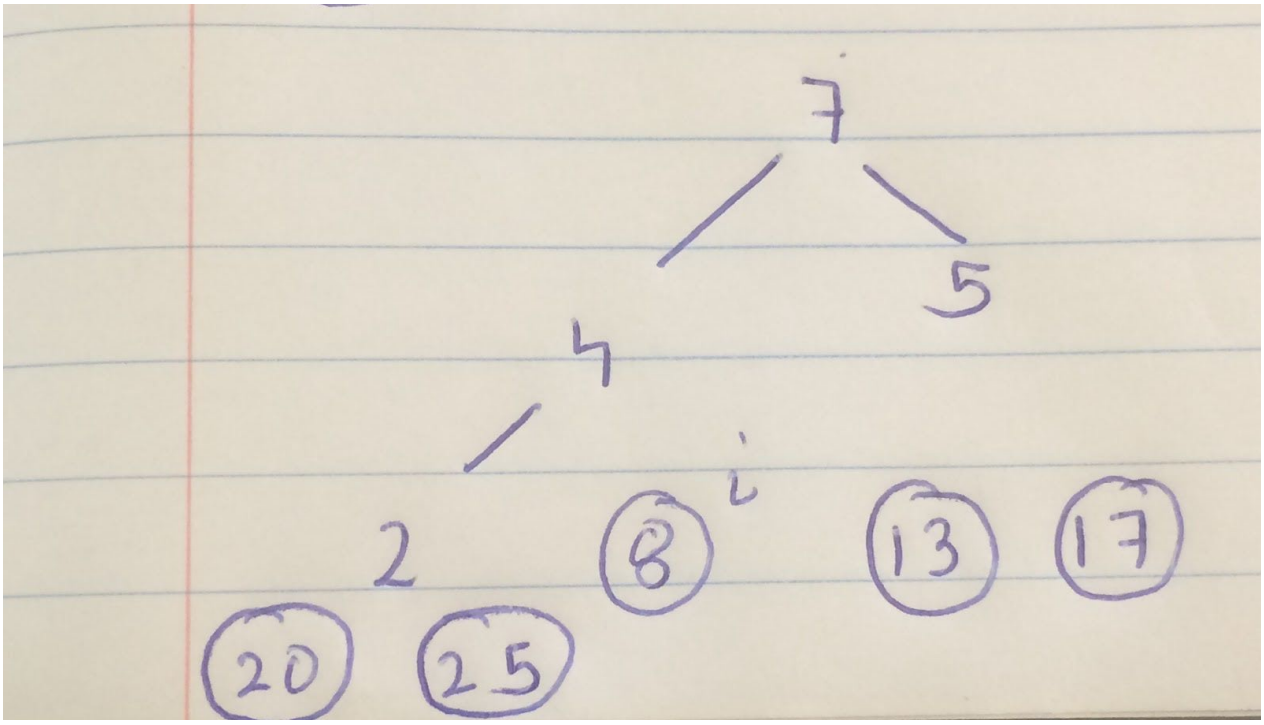
Root node 13:

The node 13 is swapped with the last node 2. As 2 is less than 8 and 7, 8 becomes the root node and 7 becomes the left child of 8. 2 becomes the right child of 7. The resultant tree is as shown below:



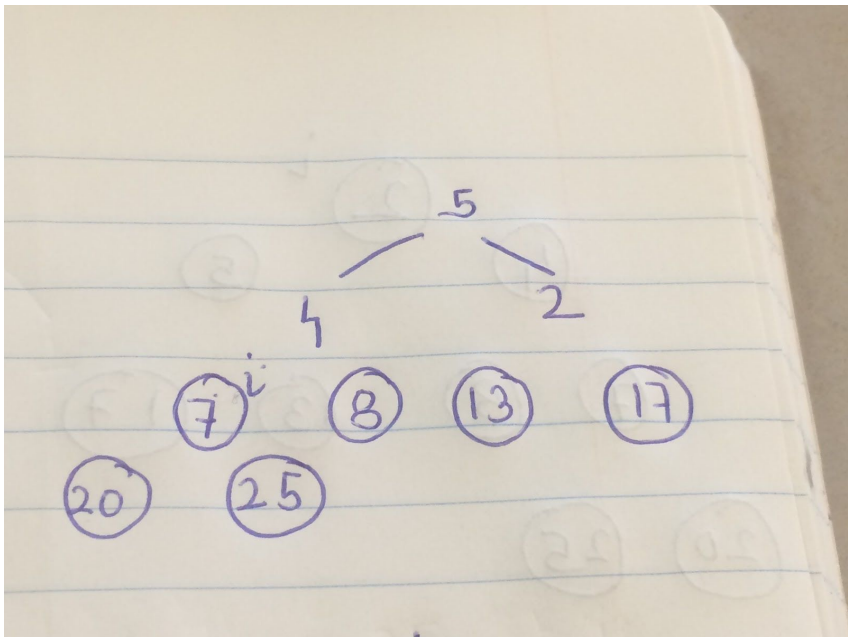
Root node 8:

The node 8 is swapped with the last node 2. As 2 is less than 7 and 4, 7 becomes the root node and 4 becomes the left child of 8. 2 becomes the left child of 4. The resultant tree is as shown below:



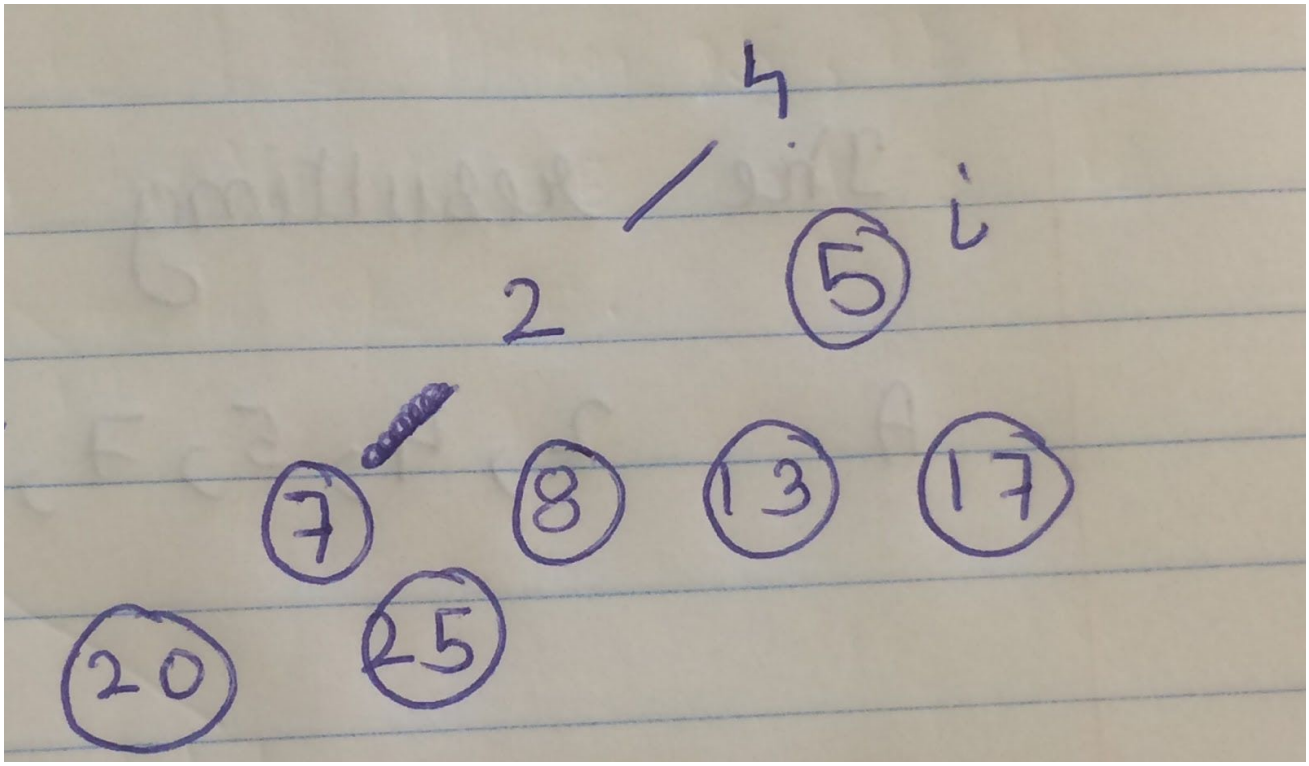
Root node 7:

The node 7 is swapped with the last node 2. As 2 is less than 5, 5 becomes the root node and 2 becomes the right child of 5. The resultant tree is as shown below:



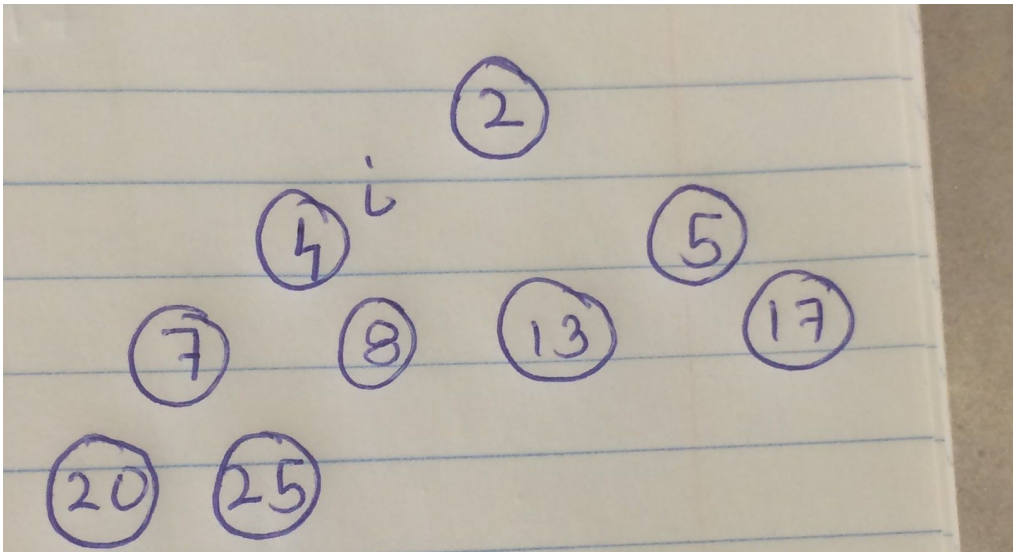
Root node 5:

The node 5 is swapped with the last node 2. As 2 is less than 4, 4 becomes the root node and 2 becomes the left child of 5. The resultant tree is as shown below:



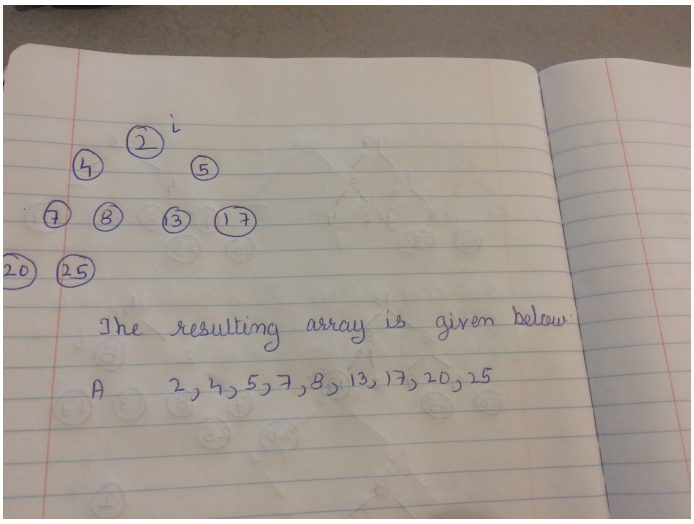
Root node 4:

The node 4 is swapped with the last node 2. 2 becomes the root node and the only node left. The resultant tree is as shown below:



Root node 2:

The resultant tree is as shown below:



Problem 4

Size	Brute force	Recursive
2	1.45	1.62
3	1.19	1.33
4	1.28	1.37
5	1.47	1.50
10	1.37	1.44
14	1.21	1.23
15	1.27	1.17
16	1.25	1.23
18	1.43	1.37
19	1.37	1.22
20	1.39	1.18

The crossover point for my laptop is around 15, however the times were noisy owing to user input collection and garbage collection during the run so it is not entirely reliable. However, it is

this value of n0 that when input size is less than n0 it is advisable to use the brute force method else when input size is greater, recursive method seems a better alternative.

When the base case of recursive function is changed to check for value of input size greater than n0, in my case to be 15. The crossover point does not change significantly. However, it affects the performance of the algorithm at the crossover point.

Brute force

```
import java.util.*;
public class BruteForce {
public static void main(String[] args) {
int i=0;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the length of array:");
int n = sc.nextInt();
System.out.println("Enter the array of numbers:");
int a[] = new int[n];
for(i=0;i<n;i++) {
a[i] = sc.nextInt();
}
sc.close();
int[] b = new int[n];
b = bruteforce(a,n);
System.out.println("The max subarray is:");
for(i=0;i<b.length;i++) {
System.out.println(b[i]+" ");
}
}
public static int[] bruteforce(int[] a, int n) {
int start=0,end=0,total=0;
int max = a[0];
int i=0,j=0;
int b[] = new int[n];
for(i=0;i<n;i++) {
total = 0;
for(j=i;j<n;j++) {
total = total + a[j];
if(total>max) {
max=total;
```

```
start = i;
end = j;
}
}
}
for(i=start;i<=end;i++) {
b[i]=a[i];
}
return b;
}
}
```

Recursive Maximum array code:

```
import java.util.Scanner;
public class RecursiveArray {
public static void main(String[] args) {
int j=0,i=0;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the length of array:");
int n = sc.nextInt();
System.out.println("Enter the array of numbers:");
int a[] = new int[n];
for(j=0;j<n;j++) {
a[j] = sc.nextInt();
}
sc.close();
int b[] = new int[n];
int m = 0;
b = recursive(a,m,n-1);
System.out.println("The max subarray is:");
for(i=0;i<b.length;i++) {
System.out.println(b[i]+" ");
}
}
public static int[] recursive(int a[],int m, int n) {
if(m==n) {
return a;
}
int mid = (m+n)/2;
```

```
int i=0,sumd=0,sume=0,sumf=0;
int d[]=new int[n];
int e[]=new int[n];
int f[]=new int[n];
d = recursive(a,m,mid);
e = recursive(a,mid+1,n);
f = recursive(a,m,mid,n);
for(i=0;i<d.length;i++) {
sumd = sumd + d[i];
}
for(i=0;i<e.length;i++) {
sume = sume + e[i];
}
for(i=0;i<f.length;i++) {
sumf = sumf + f[i];
}
return ((sumd>sume)?((sumd>sumf)? d: f):((sume>sumf)? e: f));
}
public static int[] recursive(int a[],int m,int mid,int n) {
int sum = a[mid+1];
int b[]=new int[n];
int maxRight = sum;
int R = mid;
for(int i=mid+2;i<=n;i++) {
sum = sum + a[i];
if(sum>maxRight) {
maxRight = sum;
R = i;
}
}
sum = a[mid];
int maxLeft = sum,L=mid-1;
for(int j=mid-1;j>=m;j--) {
sum = sum + a[j];
if(sum>maxLeft) {
maxLeft = sum;
L = j;
}
}
```

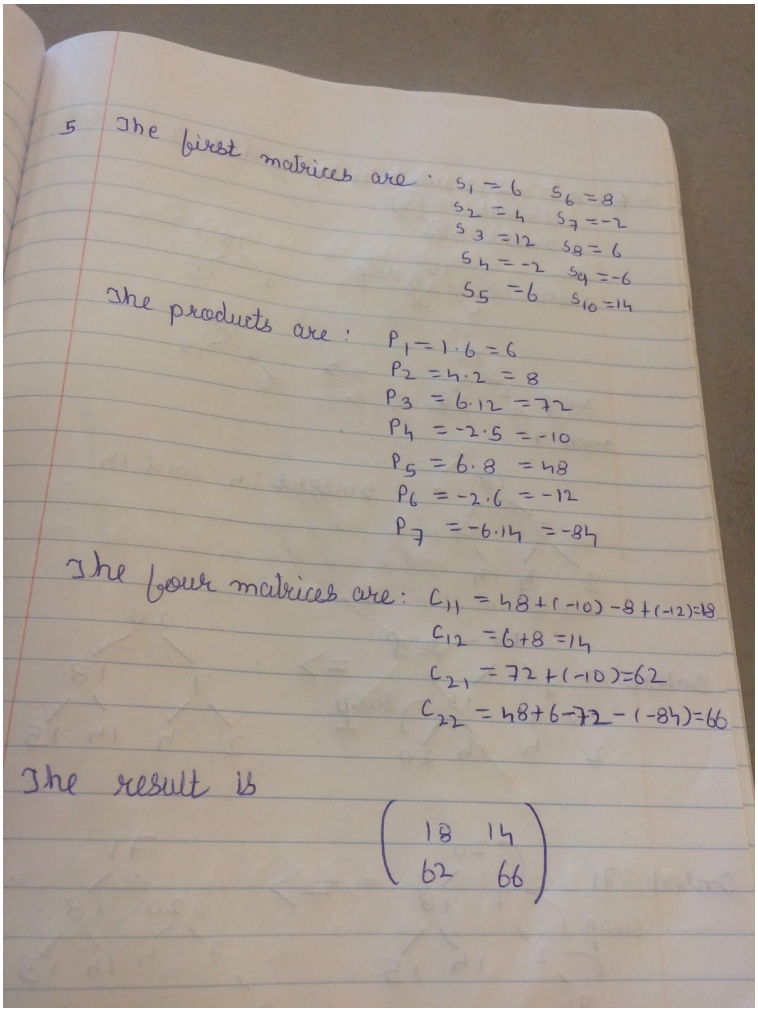
```
System.out.println(L+" "+R);
for(int i=L+1;i<R;i++) {
b[i]=a[i];
}
return b;
}
}
```

Mixed Algorithm: By modifying the recursive function to call the brute force method from the first code when it reaches a crossover point.

```
public static int[] recursive(int a[],int m, int n) {
int crossover_point = 15;
if(m-n < crossover_point) {
return bruteforce(a,n);
}
int mid = (m+n)/2;

int i=0,sumd=0,sume=0,sumf=0;
int d[]=new int[n];
int e[]=new int[n];
int f[]=new int[n];
d = recursive(a,m,mid);
e = recursive(a,mid+1,n);
f = recursive(a,m,mid,n);
for(i=0;i<d.length;i++) {
sumd = sumd + d[i];
}
for(i=0;i<e.length;i++) {
sume = sume + e[i];
}
for(i=0;i<f.length;i++) {
sumf = sumf + f[i];
}
return ((sumd>sume)?((sumd>sumf)? d: f):((sume>sumf)? e: f));
}
```

Problem 5



5 The first matrices are: $S_1 = 6$ $S_6 = 8$
$S_2 = 4$ $S_7 = -2$
$S_3 = 12$ $S_8 = 6$
$S_4 = -2$ $S_9 = -6$
$S_5 = 6$ $S_{10} = 14$

The products are: $P_1 = 1 \cdot 6 = 6$
$P_2 = 4 \cdot 2 = 8$
$P_3 = 6 \cdot 12 = 72$
$P_4 = -2 \cdot 5 = -10$
$P_5 = 6 \cdot 8 = 48$
$P_6 = -2 \cdot 6 = -12$
$P_7 = -6 \cdot 14 = -84$

The four matrices are: $C_{11} = 48 + (-10) - 8 + (-12) = 18$
$C_{12} = 6 + 8 = 14$
$C_{21} = 72 + (-10) = 62$
$C_{22} = 48 + 6 - 72 - (-84) = 66$

The result is

$$\begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}$$

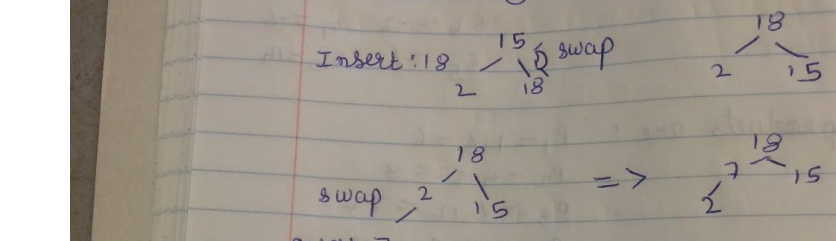
Problem 6

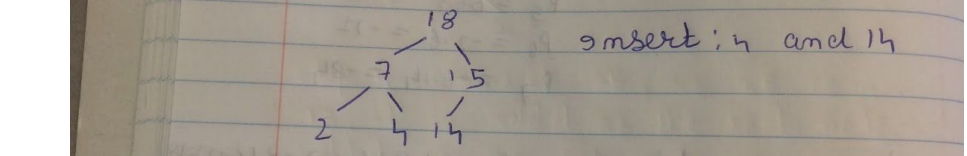
Part a:

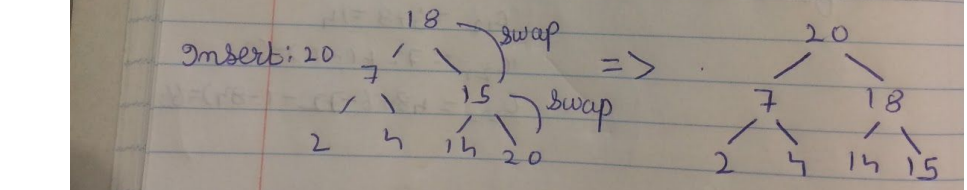
6 a) 15, 2, 18, 7, 4, 14, 20, 71, 6, 3

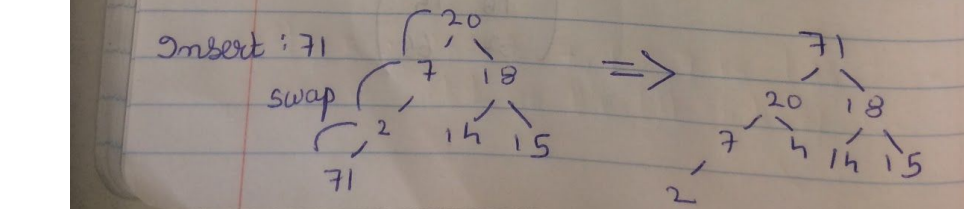
Insert: 15 (15)

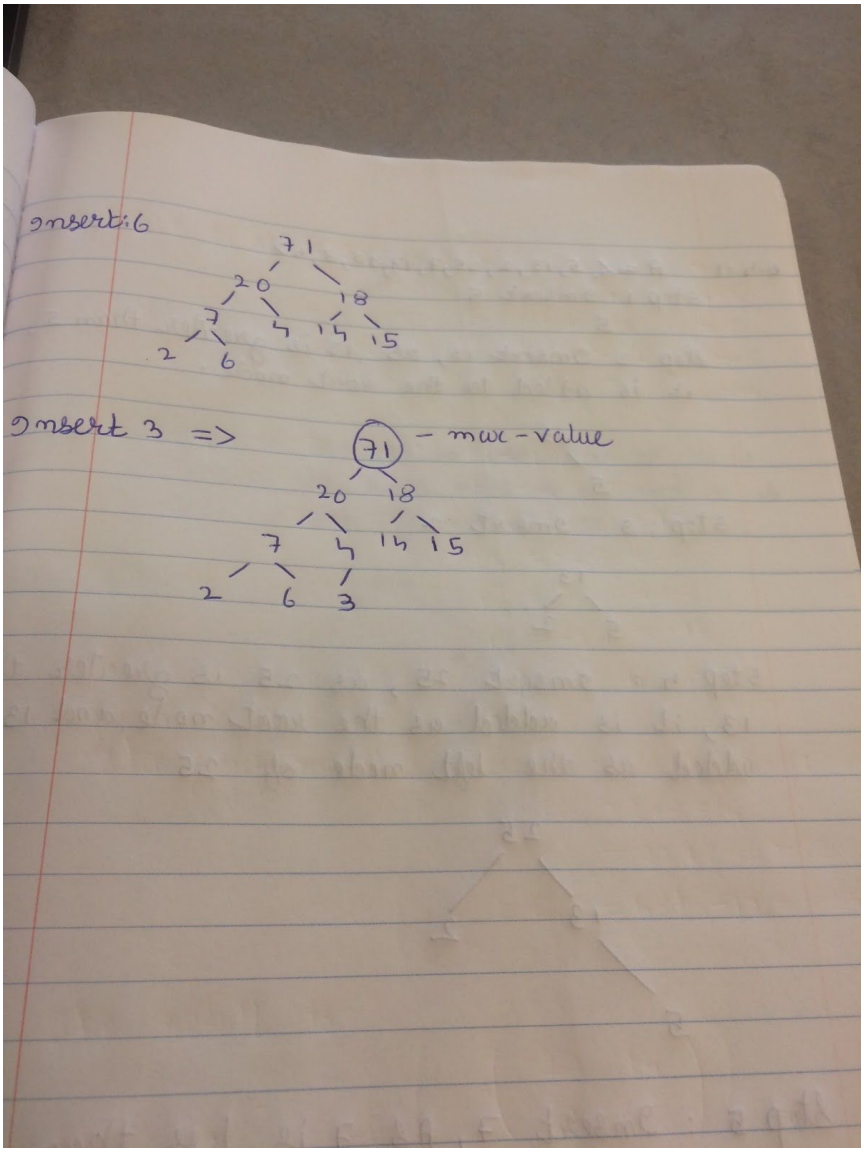
Insert: 2 (2)











Part b Code:

```
public class Main
{

   private int[] heapArray;
   private int size;
   private int maximumSize;

      public Main(int maximumSize)
         {
      this.maximumSize = maximumSize;
```

```
    this.size = 0;
    heapArray = new int[this.maximumSize + 1];
    heapArray[0] = Integer.MAX_VALUE;
        }


private int parent(int position)
        {
    return position / 2;
        }


private int leftChild(int position)
        {
    return (2 * position);
        }
private int rightChild(int position)
        {
    return (2 * position) + 1;
        }

   private boolean isLeaf(int position)
        {
    if (position >= (size / 2) && position <= size) {
        return true;
    }
    return false;
        }

private void swap(int position1, int position2)
        {
    int temporary;
    temporary = heapArray[position1];
    heapArray[position1] = heapArray[position2];
    heapArray[position2] = temporary;
        }

   private void maxHeapify(int position)
        {
```

```
    if (isLeaf(position))
       return;

    if (heapArray[position] < heapArray[leftChild(position)] ||
                heapArray[position] < heapArray[rightChild(position)]) {

       if (heapArray[leftChild(position)] > heapArray[rightChild(position)]) {
                swap(pos, leftChild(position));
                maxHeapify(leftChild(position));
       }
       else {
                swap(pos, rightChild(position));

                maxHeapify(rightChild(position));
       }
    }
       }

   public void insertElement(int element)
       {
    heapArray[++size] = element;

         int current = size;
    while (heapArray[current] > heapArray[parent(current)]) {
       swap(current, parent(current));
       current = parent(current);
    }
       }

public void printHeap()
       {
    for (int i = 1; i <= size / 2; i++) {
       System.out.print(" PARENT : " + heapArray[i] + " LEFT CHILD : " +
                heapArray[2 * i] + " RIGHT CHILD :" + heapArray[2 * i + 1]);
       System.out.println();
    }
       }
```

```
    public int extractMax()
        {
     int popped = heapArray[1];
     heapArray[1] = heapArray[size--];
     maxHeapify(1);
     return popped;
        }

   public static void main(String[] arg)
        {

     System.out.println("The Max Heap is ");
     int[] array={5,2,18,7,4,14,20,71,6,3};
     Main maxHeap = new Main(15);
          for(int element:array)
       maxHeap.insertElement(element);

     maxHeap.printHeap();

     System.out.println("The max val is " + maxHeap.extractMax());
        }
}
```

Part c Code:

```
static void printThreeLargest(int array[], int arraySize)
{
int i, first, second, third;
if (arraySize < 3)

{
System.out.print(" Invalid Input ");
return;
}
third = first = second = Integer.MIN_VALUE;
for (i = 0; i < arraySize ; i ++)
{

if (array[i] > first)
{
```

```
third = second;
second = first;
first = array[i];
}

else if (array[i] > second)
{
third = second;
second = array[i];
}
else if (array[i] > third)
third = array[i];
}
System.out.println("Three largest elements are " +
first + " " + second + " " + third);
}
```