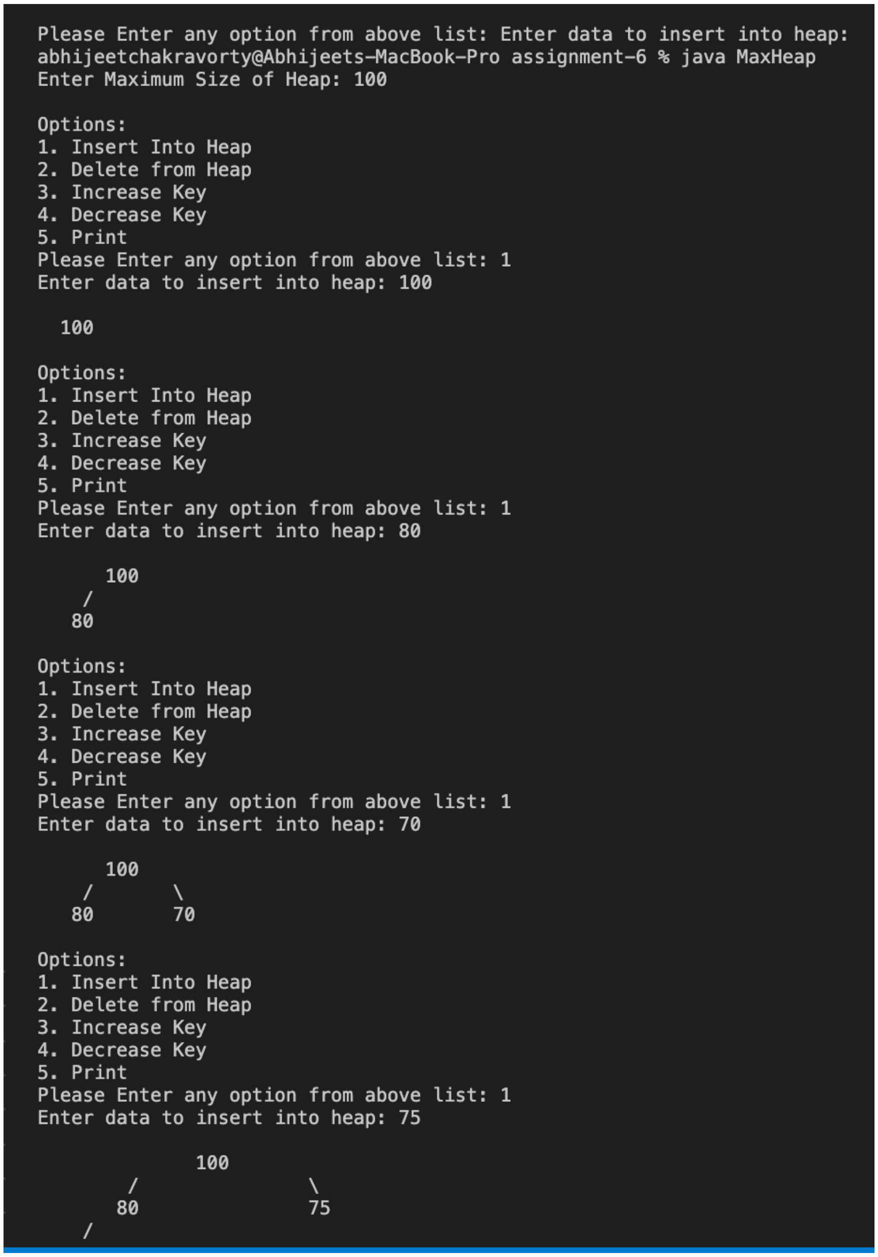
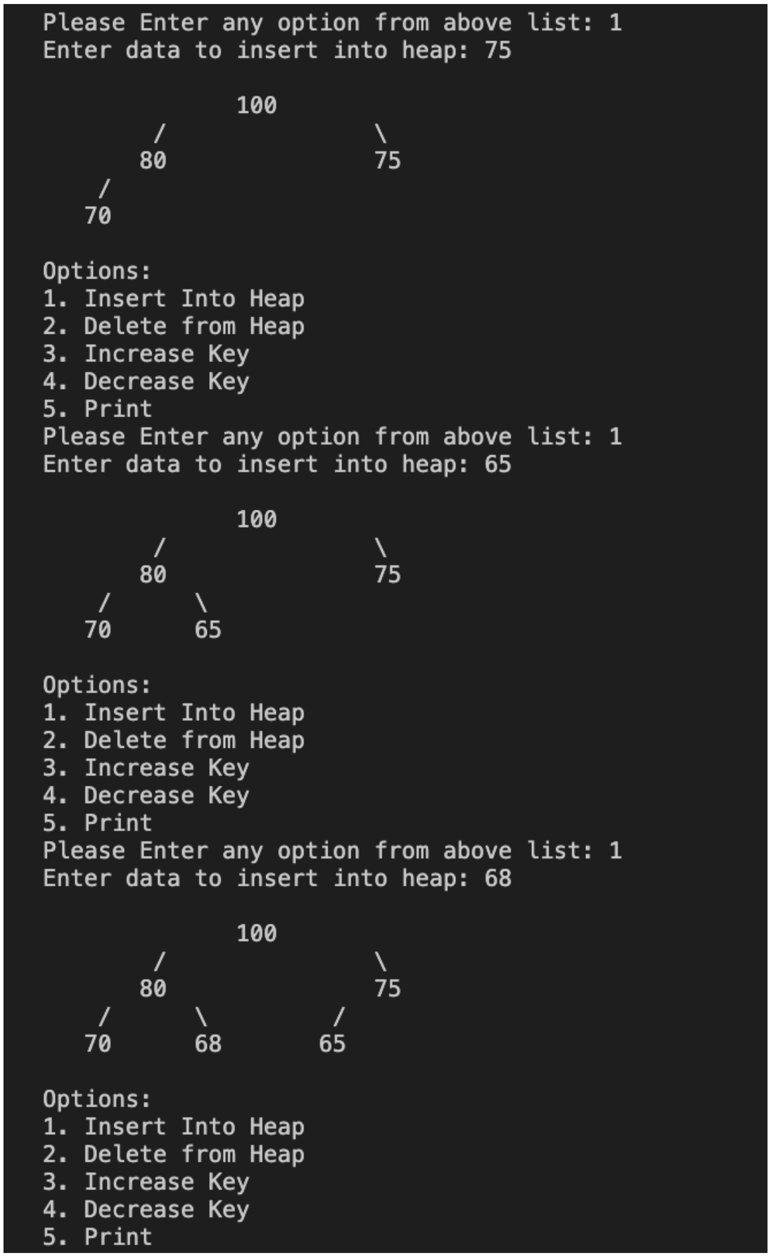
Q 1) Construct a menu driven max heap.

- **a. Insert**
- **b. Delete**
- **c. Increase Key**
- **d. Decrease Key**
- **e. Print**

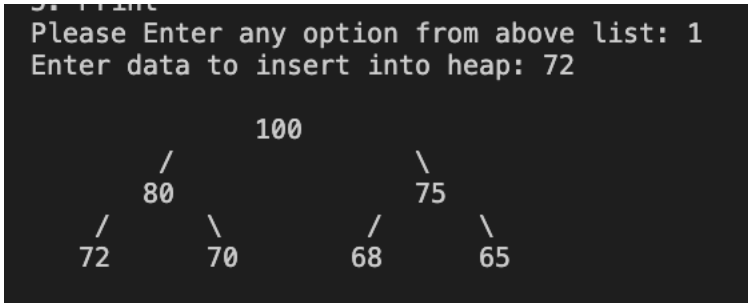
Ans 1)



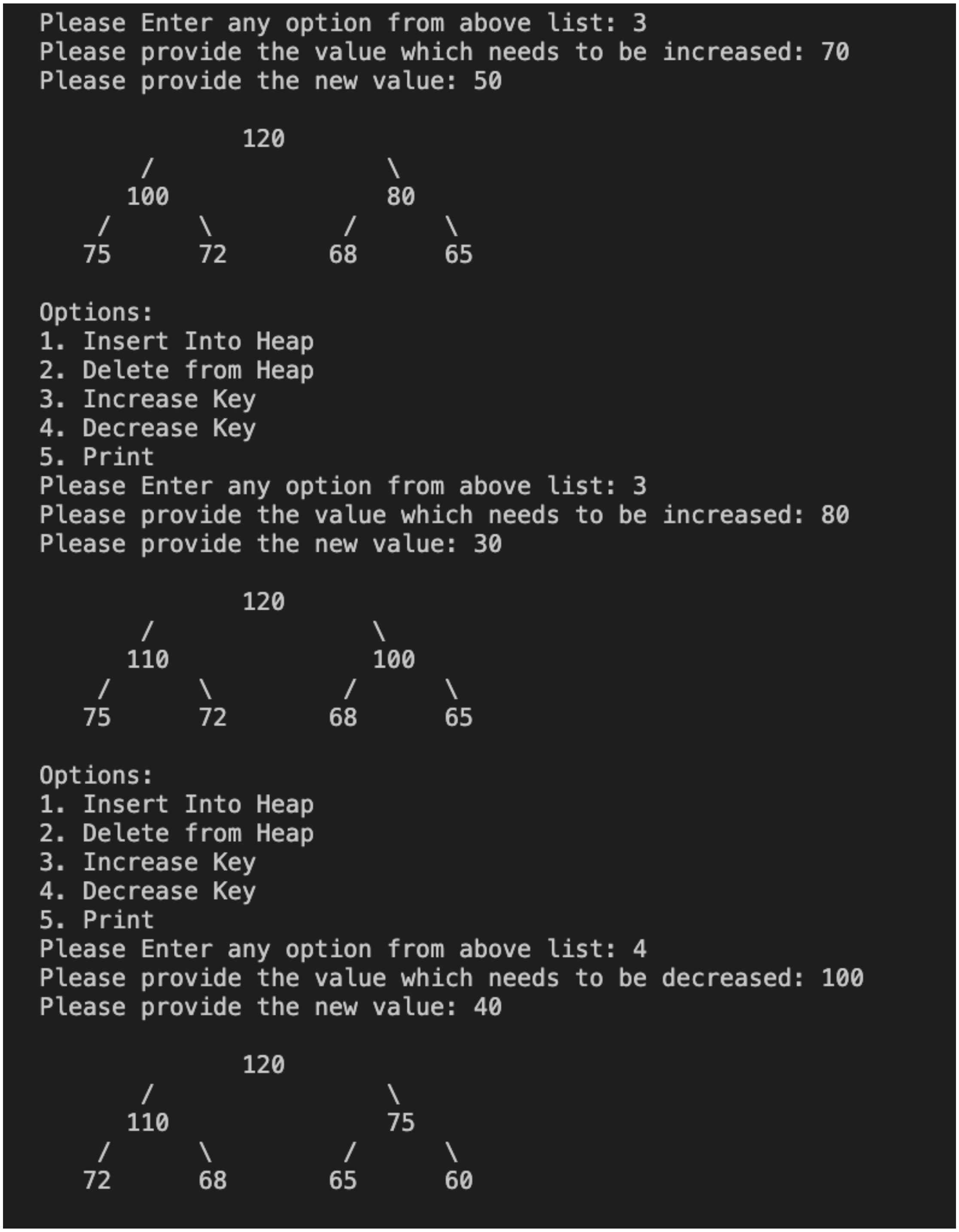
```
Please Enter any option from above list: Enter data to insert into heap:
abhijeetchakravorty@Abhijeets-MacBook-Pro assignment-6 % java MaxHeap
Enter Maximum Size of Heap: 100

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 1
Enter data to insert into heap: 100

  100

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 1
Enter data to insert into heap: 80

      100
    /
   80

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 1
Enter data to insert into heap: 70

      100
    /       \
   80       70

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 1
Enter data to insert into heap: 75

              100
        /               \
       80               75
    /
```



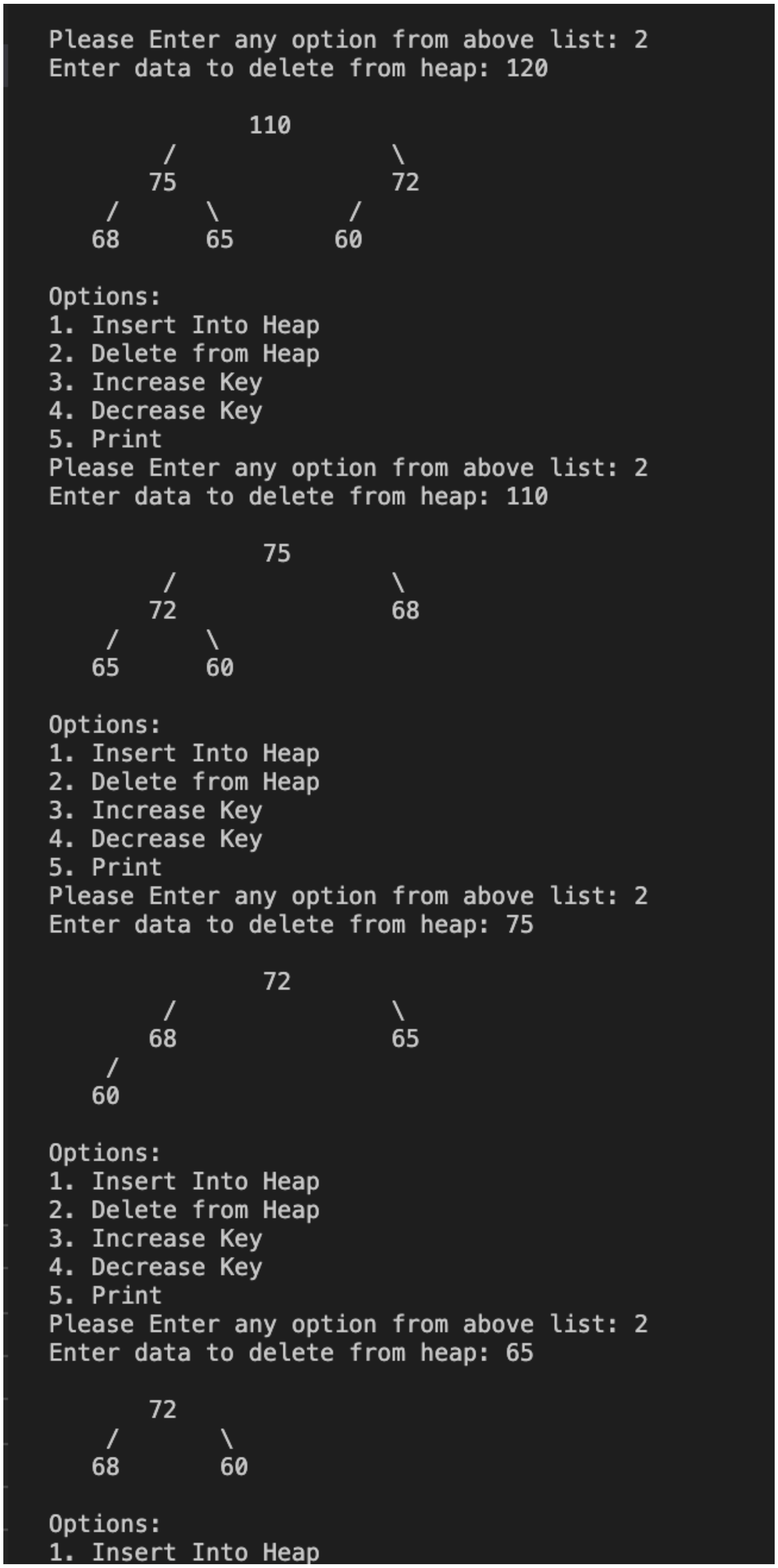
```
Please Enter any option from above list: 1
Enter data to insert into heap: 75

              100
        /               \
       80               75
    /
   70

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 1
Enter data to insert into heap: 65

              100
        /               \
       80               75
    /      \
   70      65

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 1
Enter data to insert into heap: 68

              100
        /               \
       80               75
    /      \        /
   70      68       65

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
```



```
5. Print
Please Enter any option from above list: 1
Enter data to insert into heap: 72

              100
        /               \
       80               75
    /      \        /      \
   72      70       68      65
```



```
Please Enter any option from above list: 3
Please provide the value which needs to be increased: 70
Please provide the new value: 50

              120
       /                \
      100               80
    /      \         /      \
   75      72       68      65

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 3
Please provide the value which needs to be increased: 80
Please provide the new value: 30

              120
       /               \
      110              100
    /      \         /      \
   75      72       68      65

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 4
Please provide the value which needs to be decreased: 100
Please provide the new value: 40

              120
       /                \
      110               75
    /      \         /      \
   72      68       65      60
```



```
Please Enter any option from above list: 2
Enter data to delete from heap: 120

              110
        /               \
       75               72
    /      \        /
   68      65       60

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 2
Enter data to delete from heap: 110

               75
        /               \
       72               68
    /      \
   65      60

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 2
Enter data to delete from heap: 75

               72
        /               \
       68               65
    /
   60

Options:
1. Insert Into Heap
2. Delete from Heap
3. Increase Key
4. Decrease Key
5. Print
Please Enter any option from above list: 2
Enter data to delete from heap: 65

       72
    /       \
   68       60

Options:
1. Insert Into Heap
```

```
import java.util.*;
import java.lang.*;
import java.io.*;
public class MaxHeap {
        int heapArray[];
        final int MAX_SIZE;
        int size;

        MaxHeap(int MAX_SIZE) {
                this.MAX_SIZE = MAX_SIZE;
                heapArray = new int[MAX_SIZE];
                size = 0;
        }
        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.printf("Enter Maximum Size of Heap: 
");
                final int MAX_SIZE = input.nextInt();
                MaxHeap heap = new MaxHeap(MAX_SIZE);
                int data, key, replace;
                boolean val = false;
                while (true) {
                        System.out.println("\nOptions:");
                        System.out.println("1. Insert Into 
Heap");
                        System.out.println("2. Delete from 
Heap");
                        System.out.println("3. Increase Key");
                        System.out.println("4. Decrease Key");
                        System.out.println("5. Print");
                        System.out.printf("Please Enter any 
option from above list: ");
```

```
            int choice = input.nextInt();
            switch (choice) {
                case 1:
                    if (heap.size ==
heap.MAX_SIZE) {

System.out.println("HEAP is FULL. Insertion isn't possible");
                        break;
                    }
                    System.out.printf("Enter
data to insert into heap: ");
                    data = input.nextInt();
                    heap =
heapOperation.insert(heap, data);

heapOperation.display(heap);
                    break;
                case 2:
                    System.out.printf("Enter
data to delete from heap: ");
                    data = input.nextInt();
                    heap =
heapOperation.delete(heap, data);

heapOperation.display(heap);
                    break;
                case 3:
                    System.out.print("Please
provide the value which needs to be increased: ");
                    key = input.nextInt();
                    System.out.print("Please
provide the new value: ");
                    replace =
input.nextInt();
```

```
if(heapOperation.check(heap.heapArray, key)) {
                                                for (int k=0; k
< heap.heapArray.length; k++) {
                                                        if
(heap.heapArray[k] == key) {

heap.heapArray[k]+=replace;
                                                        }
                                                }
                                                heap.heapArray =
heapOperation.sort(heap.heapArray);

heapOperation.display(heap);
                                        } else {

System.out.println("Please provide a valid value");
                                        };
                                        //
heapOperation.display(heap);
                                        break;
                                case 4:
                                        System.out.print("Please
provide the value which needs to be decreased: ");
                                        key = input.nextInt();
                                        System.out.print("Please
provide the new value: ");
                                        replace =
input.nextInt();

if(heapOperation.check(heap.heapArray, key)) {
                                                for (int k=0; k
< heap.heapArray.length; k++) {
                                                        if
(heap.heapArray[k] == key) {
```

```
heap.heapArray[k]-=replace;
                                                        }
                                                }
                                                heap.heapArray = 
heapOperation.sort(heap.heapArray);

heapOperation.display(heap);
                                        } else {

System.out.println("Please provide a valid value");
                                        };
                                        //
heapOperation.display(heap);
                                        break;
                                case 5:

heapOperation.display(heap);
                                        break;
                                default:

System.out.println("Oops!! Invalid Option. Please select valid 
option");
                                        break;
                        }
                }
        }
}

class heapOperation{

        static MaxHeap insert(MaxHeap heap, int data) {
                heap.heapArray[heap.size] = data;
                heap.size += 1;
                int size = heap.size - 1;
                while (size != 0) {
                        int parent = (size - 1) / 2;
                        if (heap.heapArray[size] > 
heap.heapArray[parent]) {
                                heap.heapArray[size] += 
heap.heapArray[parent];
                                heap.heapArray[parent] = 
heap.heapArray[size] - heap.heapArray[parent];
                                heap.heapArray[size] -= 
heap.heapArray[parent];
                                size = parent;
                        } else
                                break;
                }
                heap.heapArray = sort(heap.heapArray);
                return heap;
        }

        static Boolean check(int[] arr, int toCheckValue) {
                // check if the specified element
                // is present in the array or not
                // using Linear Search method
                boolean test = false;
                for (int element: arr) {
                        if (element == toCheckValue) {
                                test = true;
                                break;
                        }
                }

                if(!test) {
                        return false;
                } else {
                        return true;
                }
```

```
    }

    static int[] sort(int[] num) {
        int temp = 0;
        for (int i = 0; i < num.length; i++) {
            for (int j = i+1; j < num.length; j++) {
                if(num[i] < num[j]) {
                    temp = num[i];
                    num[i] = num[j];
                    num[j] = temp;
                }
            }
        }
        return num;
    }

    static int[] removeTheElement(int[] arr, int index) {
        int[] my_array = arr;
        int removeIndex = index;

        for(int i = removeIndex; i < my_array.length -1;
i++){
            my_array[i] = my_array[i + 1];
        }

        return my_array;
    }

    static MaxHeap delete(MaxHeap heap, int data) {
        int j = 0, index = 0;
        while (j < heap.heapArray.length) {
            if (heap.heapArray[j]==data) {
                index = j;
                break;
            }
```

```
                        j++;
                }
                heap.heapArray = 
removeTheElement(heap.heapArray, index);
                int temp = heap.heapArray[0];
                heap.heapArray[heap.size-1] = temp;
                heap.size -= 1;

                return heap;
        }


        static void display(MaxHeap heap) {
                int height, width;
                height = (int)(Math.log(heap.size) / 
Math.log(2)) + 1;
                width = (int) Math.ceil(Math.pow(2, height + 
2));
                int len = width * height * 2 + 2;
                StringBuilder sb = new StringBuilder(len);
                for (int i = 1; i <= len; i++)
                        sb.append(i < len - 2 && i % width == 
0 ? "\n" : ' ');

                displayR(sb, width / 2, 1, width / 4, width, 
heap.heapArray, 0, " ", heap.size);
                System.out.println(sb);
        }

        static void displayR(StringBuilder sb, int c, int r, int 
d, int w, int[] heap, int n,
                String edge, int size) {
                if (n < size) {
                        displayR(sb, c - d, r + 2, d / 2, w, 
heap, n * 2 + 1, " /", size);
```

```
                        String s = String.valueOf(heap[n]);
                        int idx1 = r * w + c - (s.length() + 
1) / 2;
                        int idx2 = idx1 + s.length();
                        int idx3 = idx1 - w;
                        if (idx2 < sb.length())
                                sb.replace(idx1, idx2, 
s).replace(idx3, idx3 + 2, edge);
                        displayR(sb, c + d, r + 2, d / 2, w, 
heap, n * 2 + 2, "\\ ", size);
                }
        }
}
```