Student ID: 202112047

IT623 - Lab Assignment 8

1. **Write a program to implement a min heap. You need to write insert function, top function, pop function.**

   Code:

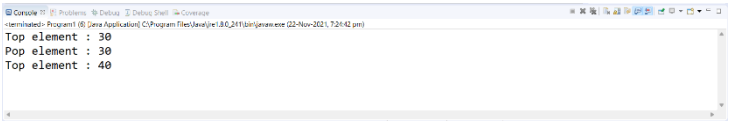
```
public class Program1 {
	int[] Heap;
	int size;
	int maxSize;

	static final int FRONT = 1;

	public Program1(int s) {
		maxSize = s;
		size = 0;
		Heap = new int[maxSize + 1];
		Heap[0] = Integer.MIN_VALUE;
	}

	public int parent(int pos) {
		return pos / 2;
	}

	public int leftChild(int pos) {
		return (pos * 2) + 1;
	}

	public int rightChild(int pos) {
		return pos * 2;
```

```
	}

	public boolean isLeaf(int pos) {
		if (pos >= (size / 2) && pos <= size)
			return true;

		return false;
	}

	public void swap(int x, int y) {
		int temp;

		temp = Heap[x];
		Heap[x] = Heap[y];
		Heap[y] = temp;
	}

	public void minHeap(int pos) {
		if (!isLeaf(pos))
			if (Heap[pos] > Heap[leftChild(pos)] || Heap[pos] >
Heap[rightChild(pos)])
				if (Heap[leftChild(pos)] < Heap[rightChild(pos)]) {
					swap(pos, leftChild(pos));
					minHeap(leftChild(pos));
				} else {
					swap(pos, rightChild(pos));
					minHeap(rightChild(pos));
				}
	}

	public void insert(int element) {
```

```
        if (size >= maxSize)
                return;

        Heap[++size] = element;
        int curr = size;

        while (Heap[curr] < Heap[parent(curr)]) {
                swap(curr, parent(curr));
                curr = parent(curr);
        }
}

public int top() {
        int top = Heap[FRONT];
        minHeap(FRONT);
        return top;
}

public int pop() {
        int popped = Heap[FRONT];
        Heap[FRONT] = Heap[size--];
        minHeap(FRONT);
        return popped;
}

public static void main(String[] args) {
        Program1 p = new Program1(3);

        p.insert(50);
        p.insert(40);
        p.insert(30);
```

```
            System.out.println("Top element : " + p.top());
            System.out.println("Pop element : " + p.pop());
            System.out.println("Top element : " + p.top());
        }

}
```

Output Snapshot:



2. **Write a program to implement a max heap. You need to write insert function, top function, pop function.**

Code:

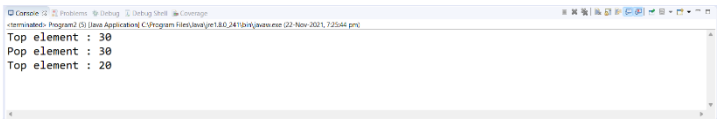
```
public class Program2 {
        int[] Heap;
        int size;
        int maxSize;

        public Program2(int s) {
            maxSize = s;
            size = 0;
            Heap = new int[maxSize];
        }
```

```
public int parent(int pos) {
	return (pos - 1) / 2;
}

public int leftChild(int pos) {
	return pos * 2;
}

public int rightChild(int pos) {
	return (pos * 2) + 1;
}

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

public void swap(int x, int y) {
	int temp;

	temp = Heap[x];
	Heap[x] = Heap[y];
	Heap[y] = temp;
}

public void maxHeap(int pos) {
	if (isLeaf(pos))
		return;

	if (Heap[pos] < Heap[leftChild(pos)] || Heap[pos] <
	Heap[rightChild(pos)])
		if (Heap[leftChild(pos)] > Heap[rightChild(pos)]) {
```

```
                swap(pos, leftChild(pos));
                maxHeap(leftChild(pos));
            } else {
                swap(pos, rightChild(pos));
                maxHeap(rightChild(pos));
            }
}

public void insert(int element) {
    Heap[size] = element;
    int curr = size;

    while (Heap[curr] > Heap[parent(curr)]) {
        swap(curr, parent(curr));
        curr = parent(curr);
    }
    size++;
}

public int pop() {
    int popped = Heap[0];
    Heap[0] = Heap[--size];
    maxHeap(0);
    return popped;
}

public int top() {
    int top = Heap[0];
    maxHeap(0);
    return top;
}

public static void main(String[] args) {
```

```
        Program2 p = new Program2(5);

        p.insert(10);
        p.insert(20);
        p.insert(30);

        System.out.println("Top element : " + p.top());
        System.out.println("Pop element : " + p.pop());
        System.out.println("Top element : " + p.top());
    }

}
```

Output Snapshot:



3. **Given an integer array, find k'th largest element in the array where k is a positive integer less than or equal to the length of array.**

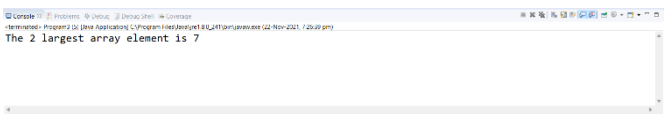
Code:

```
public class Program3 {

    int size = 0;

    public int findKthLargest(int[] nums, int k) {
```

```
        size = nums.length;
        int ans = 0;
        int last = (size - 1) / 2;
        for (int i = last; i >= 0; i--)
                downheapify(nums, i);
        for (int i = 1; i <= k; i++)
                ans = remove(nums);

        return ans;
}

public void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
}

public int remove(int[] nums) {
        swap(nums, 0, size - 1);
        int temp = nums[size - 1];
        size--;
        downheapify(nums, 0);
        return temp;
}

public void downheapify(int[] nums, int pi) {

        int mini = pi;
        int li = 2 * pi + 1;
        int ri = 2 * pi + 2;

        if (li < size && nums[li] > nums[mini])
                mini = li;
```

```
            if (ri < size && nums[ri] > nums[mini])
                mini = ri;

            if (mini != pi) {
                swap(nums, mini, pi);
                downheapify(nums, mini);
            }
        }

        public static void main(String args[]) {
            Program3 p = new Program3();

            int a[] = { 7, 4, 6, 3, 9, 1 };
            int k = 2;
            System.out.println("The " + k + " largest array element is " +
p.findKthLargest(a, k));

        }

}
```

Output Snapshot:



4. **Given an array of pairs, find all symmetric pairs in it. Two pairs (a, b) and (c, d) are said to be symmetric if c is equal to b and a is equal to d. For example, (10, 20) and (20, 10) are symmetric. Given an array of pairs, find all symmetric pairs in it.**
   Assume that the first elements of all pairs are distinct.

   Code:

```
import java.util.HashMap;

public class Program4 {

	static void symmetricPair(int arr[][]) {
		HashMap<Integer, Integer> h = new HashMap<Integer,Integer>();

		for (int i = 0; i < arr.length; i++) {
			int first = arr[i][0];
			int second = arr[i][1];

			Integer value = h.get(second);

			if (value != null && value == first)
				System.out.println("(" + second + " , " + first + ")");
			else
				h.put(first, second);
		}
	}

	public static void main(String[] args) {

		int arr[][] = new int[5][2];

		arr[0][0] = 11;
```

```
            arr[0][1] = 20;
            arr[1][0] = 30;
            arr[1][1] = 40;
            arr[2][0] = 5;
            arr[2][1] = 10;
            arr[3][0] = 40;
            arr[3][1] = 30;
            arr[4][0] = 10;
            arr[4][1] = 5;

            System.out.println("Following pairs have symmetric pairs");
            symmetricPair(arr);
        }

}
```

Output Snapshot:

5. **Given an array A[] of n numbers and another number x, determines whether or not there exist two elements in A[] whose sum is exactly x.**

Code:

```
import java.util.HashMap;

public class Program5 {

	static void findSumPairs(int arr[], int sum) {
		HashMap<Integer, Integer> hm = new HashMap<>();

		for (int i = 0; i < arr.length; i++) {
			for (int j = i + 1; j < arr.length; j++) {
				if ((arr[i] + arr[j]) == sum)
					hm.put(arr[i], arr[j]);
			}
		}

		if (hm.isEmpty())
			System.out.println("\nNo valid pair exists");
		else {
			hm.entrySet().forEach(m -> {
				System.out.println("Pair (" + m.getKey() + ", " +
m.getValue() + ")");
			});
			System.out.println("Valid pair exists");
		}
	}

	public static void main(String[] args) {
		int arr[] = { 0, -1, 2, -3, 1 };
		int sum = -2;

		findSumPairs(arr, sum);

		int arr1[] = { 1, -2, 1, 0, 5 };
		sum = 0;
```

```
            findSumPairs(arr1, sum);
        }
}
```

Output Snapshot: