Exercise 2. Answer Sheet

Student's Name: Yuta Nemoto Student's ID: s1240234

Problem 1. *(10 points)* Consider a priority queue S implemented as a heap. Write a pseudo-code for the **Maximum(S)** operation on this priority queue.

```
def Maximum (A)
    // Input: heap A[1...n]
    // Output: root element
    // Difference between this and Extract-Max(S) is just it removes the root factor or not
    max = A[1]
    MaxHeapify (A, 1)
    return max
```

Problem 2. *(20 points)* Consider top-down heap construction approach.

a). Write a pseudo-code for a **HeapTopDown(A)** algorithm using **Max-Heap-Insert (A, key)** operation

```
def HeapTopDown(A)
    // Constructs a heap from the elements of a given array by the top-down algorithm
    // Input: An array A[1…n] of orderable items
    // Output: A heap A[1…n]
    // Difference between this and HeapBottomUp(A) is just the time to do heapify
    // In top-down algorithm, all of the elements are heapified when it's inserted
    A.heap_size = A.length
    for i = 2 to n
            Max-Heap-Insert(A, i)
    end for
    return A
```

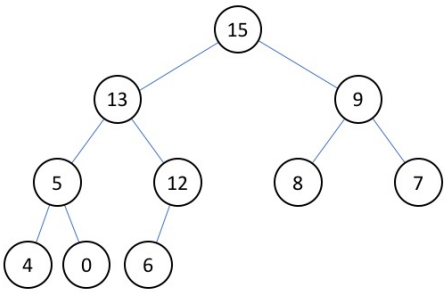
b) What is the time complexity of **HeapTopDown(A)** algorithm? Why?

First, the time complexity of Max-Heap-Insert(A, key) is $O(\log n)$.
For HeapTopDown(A), let t(n) be the time complexity.
Then,

$t(n) = 1 + (n - 2)O(\log n) = O(n \log n)$

Problem 3. *(20 points)* Illustrate the operation **Heap-Extract-Max** on a heap A=[15,13,9,5,12,8,7,4,0,6]

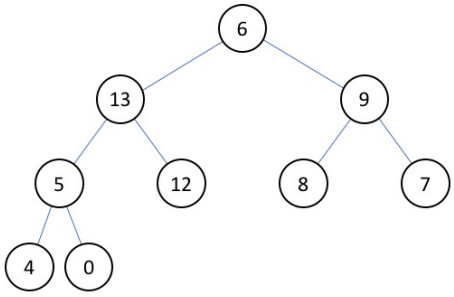
First, the heap is presented in the figure below.



This heap is given as an argument A of Extract-Max(A).
For the first session, processes are below.

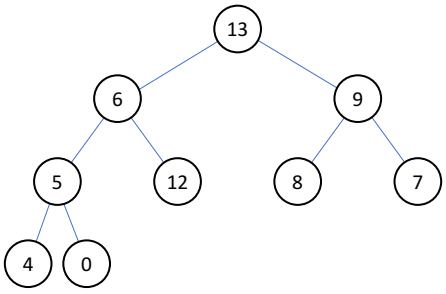
// Put the value of root into the variable “max”.
max = A[1] = **15**
// Put the value of last factor of this heap into root(Final element 6 is copied to the root element.).
A[1] = A[A.heap_size] = A[10] = **6**
// Decrease the heap size to delete the final element 6
A.heap_size = A.heap_size – 1 = 10 – 1 = **9**

And go to MaxHeapify(A, 1).
Now, the heap is presented in the figure below.



In MaxHeapify(A, 1),
l = LEFT(1) = 2 * 1 = 2
r = RIGHT(1) = 2 * 1 + 1 = 3
Now, the relationship between the factor of 1, 2, 3 is
A[l = 2] = 13 > A[r = 3] = 9 > A[1] = 6
largest = l = 2, so exchange the 1 factor ‘6’ and 2 factor ‘13’.

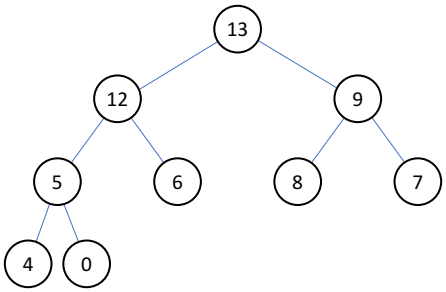
And go to MaxHeapify(A, 2).
Now, the heap is presented in the figure below.



In MaxHeapify(A, 2),
l = LEFT(2) = 2 * 2 = 4
r = RIGHT(2) = 2 * 2 + 1 = 5
Now, the relationship between the factor of 2, 4, 5 is

A[r = 5] = 12 > A[2] = 9 > A[l = 4] = 5
largest = r = 5, so exchange the 2 factor ‘6’ and 5 factor ‘12’.

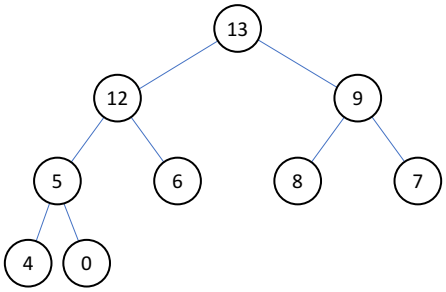
And go to MaxHeapify(A, 5).
Now, the heap is presented in the figure below.



In MaxHeapify(A, 5), there’s no subtree under the factor ‘6’,
so largest = 5, and do nothing in this session.

Here, came back to Extract-max(A),
then it just returns **max** = **15**.

And finally, the heap is operated like this figure.



Problem 4. *(50 points)* Write a program implementing **HeapBottomUp (A)** algorithm. Upload your source code. Show your input array and the output heap in the space below.

<Source Code>
HeapSort.java

```
import java.util.ArrayList;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class HeapSort {

	/* Constructor */
	public HeapSort() { }

	/* HeapBottomUP Method */
	public ArrayList<Integer> heapBottomUp(ArrayList<Integer> array){
		//int heap_size = array.size();
		for(int i = array.size() / 2 - 1; i >= 0; i--) this.maxHeapify(array, i);
		return array;
	}

	/* MaxHeapify Method */
```

```
	public void maxHeapify(ArrayList<Integer> array, int i) {
		int left = 2 * i + 1;
		int right = 2 * i + 2;
		int largest = -1;
		int temp;

		// if L child exists and is > A[i]
		if(left < array.size() && array.get(left) > array.get(i)) largest = left;
		else largest = i;

		if(right < array.size() && array.get(right) > array.get(largest)) largest = right;

		if(largest != i) {
			temp = array.get(i);
			array.set(i, array.get(largest));
			array.set(largest, temp);

			// heapify the subtree
			this.maxHeapify(array, largest);
		}
	}

	/* Require the element of the Array from the standard input and store it in array. */
	public ArrayList<Integer> doInput(ArrayList<Integer> array) {
		// Output the message
		System.out.println("Please enter the elements of the array for heap sort. \n(Elements must
be INTEGER, and must be separated by space. Input any alphabet to finish input operation.):");
		// Prepare Scanner for input
		Scanner scan = new Scanner(System.in);

		// Loop for the input
		while(true) {
			// Read the inputed elements
			String input = scan.next();
			// Set the acceptable pattern by the regular expression. Minus value is also ac-
ceptable.
			Pattern p = Pattern.compile("^-?[0-9]*$");
			// Check whether it matches to the pattern p or not.
			Matcher m = p.matcher(input);
			// If the input is integer value, add it to the array. Else, end the loop.
			if(m.find()) array.add(Integer.valueOf(input));
			else break;
		}

		System.out.println("<Loaded elements in the Array>");
		this.showContent(array);

		return array;
	}

	/* Output the element of the Array */
	public void doOutput(ArrayList<Integer> array) {
		System.out.println("<Result of Heap Sort>");
		this.showContent(array);
	}

	/* Output the content of the Array */
	public void showContent(ArrayList<Integer> array) {
		//System.err.println("showContent is called here ");
		for(int i = 0; i < array.size(); i++) System.out.print(array.get(i) + " ");
		System.out.println();
	}

	/* Main Method */
	public static void main(String[] args) {
		ArrayList<Integer> array = new ArrayList<Integer>();
		HeapSort heapSort = new HeapSort();
		array = heapSort.doInput(array);
		array = heapSort.heapBottomUp(array);
		heapSort.doOutput(array);
```

```
        }
}
```

<How to compile/run>

Command: **javac HeapSort.java**

java HeapSort

Input: **1 2 3 … (The elements for sorting) … 4 5 end**

When you input the data of the array for sorting, please enter the numbers separated by space. And after you finished this operation, please input any alphabet character(In the example above, it's the string "end") to end the input operation.

<Output result>

```
Case 1 Input: 4 1 3 2 16 9 10 14 8 7
       Output: 16 14 10 8 7 9 3 2 4 1
Case 2 Input: 5 6 3 1 2 8 9 4 7 10
       Output: 10 7 9 5 6 8 3 4 1 2
```

<Actual Interface>

```
bash-3.2$ javac HeapSort.java
bash-3.2$ java HeapSort
Please enter the elements of the array for heap sort.
(Elements must be INTEGER, and must be separated by space. Input any alphabet to finish input operation.):
4 1 3 2 16 9 10 14 8 7 end
<Loaded elements in the Array>
4 1 3 2 16 9 10 14 8 7
<Result of Heap Sort>
16 14 10 8 7 9 3 2 4 1
bash-3.2$ java HeapSort
Please enter the elements of the array for heap sort.
(Elements must be INTEGER, and must be separated by space. Input any alphabet to finish input operation.):
5 6 3 1 2 8 9 4 7 10 end
<Loaded elements in the Array>
5 6 3 1 2 8 9 4 7 10
<Result of Heap Sort>
10 7 9 5 6 8 3 4 1 2
```