IT251 Lab Assignment 9 – Binary Heaps

Note: Read in the input using a text file, and NOT by typing it in the console. The input file should be given as an argument while running your code. For e.g. for a file solution.cpp, compile it by 'g++ test.cpp' and run it by './a.out input.txt' , where 'input.txt' contains the input to the problem.

Problem 1: Implementing Max-Heaps.

Implement the Priority Queue ADT operations using **Binary Heaps**. You will have implement the following operations:

`insert, maximum, extract-max, build-heap`.

In order to implement the **`extract-max`** and **`build-heap`** methods you will need to implement the method **`heapify`**. If you are writing code in C++, define a class BinaryHeap that implements these operations. This class will need to have a field *elements* which is the array of the elements in the heap. Assume that we are only storing integers in the heap. The **`build-heap`** operation should build a new heap of the unordered elements given as input, discarding the previous heap (if any).

Input: Starting from an initially empty heap, we will execute a sequence of commands which are the heap operations. The first line of the input text file is the number of commands **n**. The following **n** lines contain each of the **n** commands, with each command specified on a new line. Each command line starts with a code for the command, following by an optional list of arguments to that command. We will code the different operations of the heap as follows:

Command No	Heap Operation
0	maximum
1	insert
2	extract-max
3	build-heap

For e.g, a line with a single entry '0' will require you to return (and print) the maximum element of the heap. Similary for the code '2', remove and return the max element in the heap and print it. The command for insert ('1') will be followed by the list of elements to be inserted into the heap, one by one, by repeated calls to the insert function, e,g, the line '1 13' will mean insert the element 13 in the heap and the line '1 14 15 26' will call the insert function three times: first for 14, then for 15 and finally for 26. The build-heap function will be called with a list of elements to be made into a heap. For the purposes of this assignment, whenever the build-heap operation is called, we will discard the existing heap and make a new heap with the elements given in the command. For e.g. the line '3 8 2 7 12 16 4' will discard the current heap and build a new heap with the elements 8,2,7,12,16 and 4. Any further operations will be done on this new heap.

Output: Each call to maximum and build-heap should print the maximum element of the heap in a new line. The calls to insert and build-heap should only do the corresponding operation without printing anything to the console.

Constraints: The number of commands **n** satisfies $\leq \mathbf{n} \leq 100$. The maximum elements a heap can have is 1000, and each heap entry is an integer between 0 and 10000.

Sample Input/Output:
Input:
1 6 2 8 12 3 7
0
2
2
0
1 11
0
3 5 15 12 7 9 13 35
2
2
2

Output:
12
12
8
7
11
35
15
13

Additional (Optional) Exercises:
Problem 2: Heapsort
Implement the in-place Heapsort using the code written in problem 1. Make sure that your sort function does not use additional space other than the input (unordered) list and maybe a few additional variables.

Problem 3:
Write a function **isBinaryHeap** that takes an arbitrary BinaryTree (not an array/list) as an input and checks if that tree is a Binary Heap or not.

Problem 4: Min-Heap
Build a Binary Min Heap similar to the MaxHeap of Problem 1. You will have to implement the following operations:
`insert, heapify, minimum, extract-min, build-heap`.