/ 20 P

Programming Task P1.

Enrollment Key: `AlgoDataExam2018`

Submission: see Section 3 of the Technical Guide

Structure

In this exercise, we implement a max-heap data structure. Your task is to complete the implementation of the following methods:

- **`buildHeap()`**: builds a heap from a given array of keys in arbitrary order.
- **`insert(`**x**`)`**: adds a new key x to the heap.
- **`deleteMax()`**: deletes the maximum key from the heap.

Most of the implementation of the max-heap data structure is already provided by the template (including the code to read the input, allocate space for the heap structure, write the state of the heap, etc).

The heap holds N keys which are integer values in the range $[-2^{31}, 2^{31} - 1]$. For simplicity, we assume that the heap will not exceed more than 100'000 keys, i.e, $N \leq 100000$.

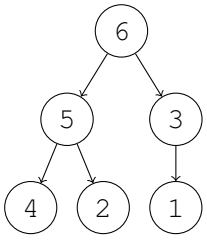
To validate the correctness of your implementation of these methods, the state of the heap is written to the output. Assuming a heap of N keys, the state of the heap is described by N partially-ordered integers that satisfy the heap property. For simplicity and convenience, the template already includes routines to output the state of the heap.

Example

- The **`buildHeap()`** method builds a heap structure by restoring the heap condition for a given array of keys. For example, given an array of $N = 5$ keys:

1	2	3	4	5	6

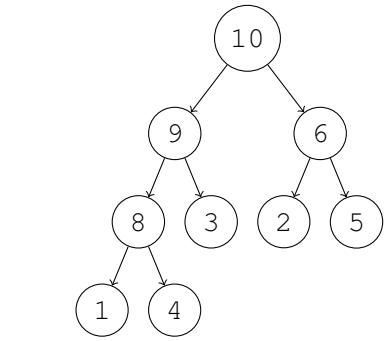
the state of the heap, as well as the partial order on the numbers is given below:



6	5	3	4	2	1

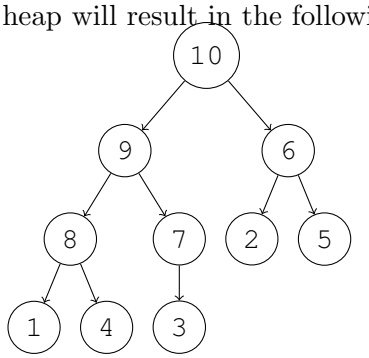
Once the heap is built, the state of the heap is written to the output.

- For the **`insert(`**x**`)`** method, assume you are given the following heap of $N = 9$ keys:



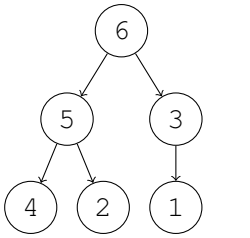
10	9	6	8	3	2	5	1	4

Inserting the key $x = 7$ into the heap will result in the following heap of $N = 10$ keys:



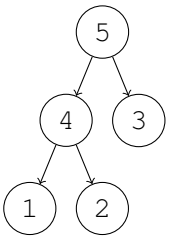
10	9	6	8	7	2	5	1	4	3

- For the **`deleteMax()`** method, assume you are given the following heap of $N = 6$ keys:



6	5	3	4	2	1

Once the maximum key is removed, the resulting heap of $N = 5$ keys looks as follows:



5	4	3	1	2

Grading

Overall, you can obtain a maximum of 20 judge points for this programming task. To get full points your program should require $O(n)$ time for the **`buildHeap()`** method, and $O(\log(n))$ for both **`insert(`**x**`)`** and **`deleteMax()`** methods (with reasonable hidden constants). Incomplete solutions can obtain partial points, namely:

- You can obtain up to 8 points for a correct implementation of **`buildHeap()`**.
- You can obtain up to 8 points for a correct implementation of **`insert(x)`**.
- You can obtain up to 4 points for a correct implementation of **`deleteMax()`**.

Instructions

For this exercise, we provide a program template as an Eclipse project in your workspace that helps you reading the input and writing the output. Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class).

The project also contains data for your local testing and a `JUnit` program that runs your `Main.java` on all the local tests – just open and run `StructureTest.launch` in the project. The local test data are different and generally smaller than the data that are used in the online judge.

Submit only your `Main.java`.

The input and output are handled by the template – you should not need the rest of this text.

Input The input of this problem consists of a number of test-cases. The first line contains T, the number of test-cases. Each of the T cases is independent of the others, contains one line, and starts with a command number that can be either 1, 2 or 3.

1. If the command number is set to 1, then we read an array from the input, build a heap, and print the state of the heap on the output.

   The command number is followed by a number N, that describes the number of keys in the array, followed by N integer values in the range $[-2^{31}, 2^{31}-1]$. All numbers are separated by a blank space.

2. If the command number is set to 2, we read in the heap, insert elements into the heap, and output the state of the heap.

   The command number is followed by a number N that describes the number of keys in the heap, followed by N integer values in the range $[-2^{31}, 2^{31}-1]$, that satisfy the heap property. The N values are then followed by a number M that describes the number of keys that must be inserted in the heap, followed by M integers in the range $[-2^{31}, 2^{31}-1]$. All numbers are separated by a blank space.

3. If the command number is set to 3, then we read in the heap, delete the maximum element from the heap once or several times, and output the state of the heap.

The command number is followed by a number N that describes the number of keys in the heap, followed by N integer values in the range $[-2^{31}, 2^{31}-1]$, that satisfy the heap property. The N values are then followed by a number M that describes the number of times we need to remove the maximum key from the heap. All numbers are separated by a blank space.

Output For every case, the output is the state of the heap.

The output contains one line for each test-case. More precisely, the i-th line of the output contains an array of integer values that correspond to the heap state, separated by a blank space. The output is terminated with an end-line character.

Example input:

```
3
1 6 1 2 3 4 5 6
2 9 10 9 6 8 3 2 5 1 4 1 7
3 6 6 5 3 4 2 1 1
```

Example output:

```
6 5 3 4 2 1
10 9 6 8 7 2 5 1 4 3
5 4 3 1 2
```