Card Game

Mr. Panda is playing a card game where he controls some monsters, each having some amount of health points (HP). Every turn, one of three things can happen:

1. He summons a monster with some amount of HP.
2. The opponent kills one of Mr. Panda's monster with the **lowest** amount of HP. In doing so, the opponent will also receive damage **equal to that amount of HP**.
3. Mr. Panda casts a healing spell that increases the HP of every of Mr. Panda's monsters which are **currently alive** by some amount.

However, it is very difficult to keep track of all the monsters. Thus, Mr. Panda wants to use a computer program to keep track of all the monsters. Having good knowledge of data structures, Mr. Panda knows that he can simulate the monsters using a **Priority Queue**. Thus, he wants to simulate a **Priority Queue** that can support the following operations:

Operation	**Description**
`SUMMON` **`[health]`**	Summons a monster with HP **[health]**.
`KILL`	Kills the monster with the **lowest** amount of HP and output the amount of damage received by the opponent.
`HEAL` **`[health]`**	Heals all monsters that are currently alive, increasing their HP by **[health]**.

Lastly, Mr. Panda wants to know the **status of all his monsters at the end of all the operations**. Thus, he wants you to list the HP of all the **monsters that are alive** in **non-decreasing** order.

Hint: A Java API PriorityQueue does not directly support increasing all the numbers inside by a certain value. **Furthermore, looping through all the values and changing them one by one is too slow.** Instead, a PriorityQueue can be used in conjunction with a numeric variable to *mimic* this behaviour.

To do so, create a separate variable to store the *total* amount of health points (HP) increased by **all HEAL operations** thus far. Hence, for every HEAL operation, you should only need to update this variable. The actual HP of a monster can then be calculated by:

Actual HP = *('base' amount of HP) + (total amount of HP 'healed' so far)*

However, take note that when you SUMMON a new monster, you will need to offset this heal value before inserting into the Priority Queue. This can be done by calculating the 'base' amount of HP where substituting Actual HP in the equation above with the health point of the monster summoned.

'base' amount of HP = (HP of summoned monster) – (total amount of HP 'healed' so far)

Similarly, when you print the damage for the KILL operation, please remember to print the actual HP of the monster killed. In addition, the same applies when printing the HP of the monsters that are alive at the end of all the operations.

Input

The first line of input contains an integer **Q**. **Q** lines will follow, representing an operation each. The operations should be executed in order and the format would be as described in the table above. (See sample)

Output
For every **KILL** operation, output the amount of damage received by the opponent.
At the end of all **Q** operations, output the health point (HP) of all the monsters that are **alive** in **non-decreasing** order. Add a single space between two consecutive health values. **Do not print a space after the last health point value.** Instead, remember to print an end-line character at the end of the output.

Limits
- $1 \leq \mathbf{Q} \leq 100{,}000$
- All the health point (HP) values will range from 1 to 10^9 inclusive.
- As the HP values are healing can be very large, please use the **long** data type to store the HP values
- It is guaranteed there is at least one alive monster when a **KILL** operation happens
- It is guaranteed there is at least one alive monster at the end of all the operations

Sample Input (**cardgame1.in**)	Sample Output (**cardgame1.out**)
11 SUMMON 20 SUMMON 10 SUMMON 10 SUMMON 30 KILL HEAL 30 SUMMON 50 SUMMON 25 HEAL 10 KILL HEAL 5	10 35 55 65 65 75

Explanation
After the first 4 monsters are summoned, the monsters in the priority queue are have health [10, 10, 20, 30] so a monster with health 10 is killed first, leaving [10, 20, 30]. Then, the 3 remaining monsters are healed by 30 so they have health [40, 50, 60]. After that, two more monsters are summoned giving [25, 40, 50, 50, 60] and then all are healed by 10 which gives [35, 50, 60, 60, 70]. Thus, the next kill will kill the monster with health 35, leaving [50, 60, 60, 70]. The last heal will leave the monsters with health [55, 65, 65, 75] and that is the end of all the operations.

Notes:
1. You should develop your program in the subdirectory **ex3** and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. You are free to define your own helper methods and classes (or remove existing ones).
3. Please be reminded that the marking scheme is:
   a. Public Test Cases (1%) - 1% for passing **all** test cases, 0% otherwise
   b. Hidden Test Cases (1%) - Partial scoring depending on test cases passed
   c. Manual Grading (1%)
      i. Overall Correctness (correctness of algorithm, severity of bugs)
      ii. Coding Style (meaningful comments, modularity, proper indentation, meaningful method and variable names)
4. Your program will be tested with a time limit of not less than **2 sec** on Codecrunch.

Skeleton File – Cardgame.java

You are given the below skeleton file `Cardgame.java`. You should see a non-empty file when you open the skeleton file. Otherwise, you might be in the wrong working directory.

```
/**
 * Name          :
 * Matric. No    :
 * PLab Acct.    :
 */

import java.util.*;

public class Cardgame {
    private void run() {
        //implement your "main" method here
    }

    public static void main(String[] args) {
        Cardgame newCardgame = new Cardgame();
        newCardgame.run();
    }
}
```