Spring 2026 COP 3330 Program #8: Use of Java API (TreeSet, PriorityQueue)

Part A: Closest Rest-Stop

We can model a highway as a portion of the x-axis starting at x = 0 and ending at x = m, where m is the length of the highway. For the purposes of this program, we assume that at first, the only places that one can get off the highway to get gas or food is at x = 0 or x = m. No intermediate stops exist in the beginning. But as time goes on, rest stops or exits are added at different mile markers (greater than 0 and less than n). Also, at different points in time, one might be at a particular mile marker and be very interested in getting food or gas, so they might want to know the least they must drive (even if it's in the wrong direction of their overall travel) to get to gas/food.

For this program, we want to process a sequence of commands each of which is one of two types:

Type 1: A rest stop/exit is added at mile marker x.

Type 2: A query – given that a motorist is at mile marker x, how far is the closest rest stop/exit in either direction?

Input Format (from standard input)

The first line of input will contain two positive integers: n ($n \leq 2 \times 10^5$), and m ($m \leq 10^9$), representing the number of commands to process and the length of the road in miles.

The next n lines will contain each command, 1 per line.

Each command will contain two space separated integers, c ($1 \leq c \leq 2$) and x ($0 \leq x \leq m$), where c represents the command type (1 for adding a stop, 2 for a query), and x, represents the mile marker for the added stop or the current position for the query.

Output Format

For each command of type 2, output the distance in miles (on a line by itself) to the nearest exit/rest stop.

Sample Input

```
20 312
2 200
1 144
2 0
2 302
1 65
2 105
1 299
2 306
1 19
2 5
2 42
1 229
1 184
2 202
1 94
2 187
1 263
2 265
2 163
2 89
```

Sample Output

```
112
0
10
39
6
5
23
18
3
2
19
5
```

Implementation requirements (for part A)

1. You can solve the problem all in a single main method in a class called reststop.java.
2. You must use a TreeSet of Integer.

Part B: What Task to Complete?

Work never seems to end! There's always a queue of tasks waiting for you. Unfortunately, some tasks are more important than others, so instead of processing requests in the order you receive them, whenever you get a free moment, you must process **the most important request** of the ones you've already received.

In this program you'll simulate a worker who gets a series of commands in sequence. Each command will be of one of two forms:

1. Add task to the work queue
2. Perform the most important task in the work queue.

We'll assume that while you are in the middle of performing a task, no new tasks appear in the queue. (This is a ridiculous assumption of course...but it simplifies how you can think about the assignment.)

We'll also assume that you are never asked to perform a task when the queue is empty, because, as previously stated, there's always work to do! (Note: since it doesn't change the programming at all, it will be possible for the queue to be empty in the middle of the sequence of commands. If this ever happens, you are guaranteed that the next command will be to add a task to the queue.)

For each task you perform, your program will need to print out **which task** was completed.

Input Format (from standard input)

The first line of input will contain one positive integer: n ($n \leq 2 \times 10^5$), representing the commands to process.

The next n lines will start with a single integer, c ($1 \leq c \leq 2$), representing the type of command.

If $c = 1$, then on the line a string t and positive integer d, separated by a space, will follow, where t represents the task to be added to the queue and d represents the dollar amount your company will receive for you completing the task. The string t will contain no whitespace.

If $c = 2$, then no other information will follow on the line.

Output Format

For each command of type 2, find the task that makes the company the most money and print both the task name and the amount of money it brings the company, separated by a space on a line by itself. If there are two or more tasks in the queue that bring in the most amount of money, break the tie by lexicographical order of the task name (just using regular compareTo).

Sample Input

```
10
1 make_program8 100
1 class_grades_update 101
2
1 make_treemap_notes 99
1 sleep 101
1 make_treeset_notes 99
2
2
2
1 find_missing_quiz 102
```

Sample Output

```
class_grades_update 101
sleep 101
make_program8 100
make_treemap_notes 99
```

Implementation requirements (for part B)

1. Please put your program in a single file called reststop.java
2. Please create a class called Task that implements Comparable<Task>. A task object should be comprised of a String and an integer.
3. In your main method in reststop.java, use a single PriorityQueue<Task>.

Deliverables

Please submit two source files:

1. **reststop.java** which contains your solution to Part A.

2. **whattask.java** which contains your solution to Part B.

Make sure to include a header comment for each file you submit and appropriate internal comments.