Problem 3 - P Record (Programming) (11 points)

Problem Description

As a competitive programming athlete, P has participated in N contests, and the i^{th} contest has a unique ID i. However, P knows that he got score s_i in the i^{th} contest and feels slightly dissatisfied with those results. Due to his self-esteem, P wants to let his scores become **increasing** over time by concealing some of the contest records.

b, who is the enemy of P, would like to interfere him. Since b is a great time traveler, he will travel to the past and do one of the following operations many times:

1. Convince P to participate in an extra contest.
2. Convince P not to participate in a contest.

Notice that b may perform the operations at any time. Any operation will not affect the scores of other contests, you don't need to worry about the Butterfly Effect.

P entrusts you, his assistant, to perform the concealment. Unfortunately, P doesn't want to tell you the exact scores of each contest. You can only ask him many times about "whether the score of the a^{th} contest is less than the b^{th} contest", but not too many because P is busy. You also need to keep updating the concealment after b has done any operation.

Oh, P does not care about minimizing the number of concealed contests. **P will accept any approximate solution that is at most twice as many as the minimum.**

Implementation details

Include `"ada-hw4-p3.h"` in the first line of your code, and then you should implement the following procedures:

```
std::vector<int> init(int N)
```

- N: P has participated in N contests initially. The i^{th} contest's position is i initially.
- This procedure is called exactly once, and should return a single array D. The elements of the returned array should form a subset of 0 to $N-1$ in any order, describing the IDs of contests you want to conceal. An empty array is also valid when you don't want to conceal any contest.

```
std::vector<int> insert(int p, int id)
```

- p: position of the inserted contest. After the insertion, the id^{th} contest will have position p, and the original contests whose positions are greater than or equal to p will be increased by one.
- id: the ID of the inserted contest.
- This procedure should return a single array D. The elements of array should form a subset of the current contest **ID** list (after the insertion) in any order, describing the IDs of contests you want to conceal. An empty array is also valid when you don't want to conceal any contest.

```
std::vector<int> remove(int p)
```

- p: position of the removed contest. After the deletion, the contest having position p will be removed, and the original contests whose positions are greater than p will be decreased by one.
- This procedure should return a single array D. The elements of array should form a subset of the current contest **ID** list (after the removal) in any order, describing the IDs of contests you want to conceal. An empty array is also valid when you don't want to conceal any contest.

The above procedures can make calls to the following procedure:

```
bool compare(int a, int b)
```

- a: the first contest's ID you want to compare.
- b: the second contest's ID you want to compare.
- Both the the compared contests should exist (a removed contest is regarded as disappeared). And $a \neq b$.
- The procedure returns `true` if the a^{th} contest's score is less than the b^{th} contest's score. Otherwise, it will return `false`.
- The grading program will record the number of calls of this procedure, which is relate to the condition of getting `Accepted`.
- The time complexity of the procedure is $O(1)$.

Examples

Consider a scenario in which P has participated in 3 contests with scores $[1, 4, 2]$, in order. The procedure `init` is called in the following way:

```
init(3)
```

This procedure may call `compare(0, 1)` which (in this scenario) returns `true`. It may then call `compare(1, 2)`, which returns `false`.

At this point, there is sufficient information to conclude that we should conceal at least one contest. So, the procedure `init` can return $[2]$ and get `Accepted`.

Notice that we can return an answer having twice of the minimum answer size. Hence, if the procedure `init` returns $[1, 2]$, it can also get `Accepted`.

After that, the procedure `insert` may be called in the following way:

```
insert(1, 3)
```

It means that b had convinced P to participate in the 3^{rd} contest at position 1. Assume that the score of it is 3. The contest ID list is $[0, 3, 1, 2]$ and the contest score list is $[1, 3, 4, 2]$ now.

This procedure may call `compare(3, 1)` which returns `true`. At this point, there is sufficient information to conclude that we can conceal only one contest. So, the procedure `insert` can return $[2]$ and get `Accepted`.

Sample grader

Download "ada-hw4-p3.zip"[6], extract and put them in the same directory of your code. Compile your code with the following command, assuming that your code is named `ada-hw4-p3.cpp`:

```
g++ -O2 -std=c++17 grader.cpp ada-hw4-p3.cpp -o ada-hw4-p3
```

We also have provided sample code (`ada-hw4-p3.cpp`), sample compile scripts (`compile_cpp.sh` for Linux and Mac OS, `compile_cpp.bat` for Windows) in the zip file.

The sample grader reads an array s of 32-bits unsigned integers indicating the scores of contests 0 to $N-1$ and Q operations. The sample grader reads input in the following format:

- line 1: N Q
- line 2: $s[0]$ $s[1]$ $\ldots$ $s[n-1]$
- line $3+i(0 \le i < Q)$: operation$[i]$: could be of the following two types:
  - insert p s: Insert a contest with score s at position p. The contest's ID will automatically become $N+j-1$ at the j^{th} call of `insert`.
  - remove p: Remove the contest at position p.

The output of sample grader is in the following format:

- line 1: $choice_{\text{init}}$ (t)
- line $2+i(0 \le i < Q)$: $choice_i$ (t)

, where:

- $choice_{\text{init}}$: Choice returned by `init`.
- $choice_i$: Choice returned by operation$[i]$.
- t: the number of calls of the procedure `compare`.

All of the choices will be printed in the following format:

- k $c[0]$ $c[1]$ $\ldots$ $c[k-1]$

Where k is the size of the choice and array c is the content of the choice.

Note:

- The sample grader will not help you judge the correctness of your answer except the format error.
- The sample grader will fail if it tries to compare two contests with the same score.

[6] http://w.csie.org/~b08902103/ADA/ada-hw4-p3.zip

Sample Input 1

```
3 2
1 4 2
insert 1 3
remove 3
```

Sample Output 1

```
2 1 2 (2)
1 2 (1)
0 (0)
```

Sample Input 2

```
3 3
3 2 1
insert 1 4
insert 2 5
remove 3
```

Sample Output 2

```
3 0 1 2 (2)
3 0 1 2 (2)
3 0 1 2 (2)
1 1 (2)
```

Sample Input 3

```
6 6
1 5 6 7 9 10
insert 1 8
insert 1 3
insert 6 4
remove 4
remove 7
remove 5
```

Sample Output 3

```
0 (5)
1 6 (1)
2 7 6 (0)
2 6 8 (2)
2 6 8 (2)
2 6 8 (2)
1 6 (0)
```

Hint

1. Try to reduce this problem to the **minimum vertex cover** problem, which has a 2-approximation algorithm as taught in class.
2. Do not write your own `main` function, and do not try to interact with stdin and stdout.
3. Feel free to use global variables.
4. Do not try to use complex data structures to maintain your array. Since the data size is small, you can spend linear time inserting or removing any element.
5. All you need to return are the IDs of the contests, not the positions.
6. For those who aren't familiar with `std::vector`, take a look at the document.
7. Do not try to steal any data from the grader. Similar behavior will be regarded as cheating and will receive heavy penalties. If you find weird behavior from the grading, please contact TA.