Homework #1

Blue Correction date: 2023/10/15 23:00
Orange Correction Date: 2023/09/24 23:00
Red Correction Date: 2023/09/22 22:00
Programming Part Due Time: 2023/10/15 23:59
Hand-Written Part Due Time: 2023/10/12 14:20
Contact TAs: `ada-ta@csie.ntu.edu.tw`

Instructions and Announcements

- There are **four programming problems** and **two hand-written problems**
- **Programming.** The judging system is located at `https://ada-judge.csie.ntu.edu.tw`. Please log in and submit your code for the programming problems (i.e., those containing "Programming" in the problem title) by the deadline. NO LATE SUBMISSION IS ALLOWED.
- **Hand-written.** For other problems (also known as the "hand-written problems"), you should upload your answer to **Gradescope**. For each sub-problem, please label (on Gradescope) the corresponding pages where your work shows up, **or you may get some penalty for each problem you did not label** . NO LATE SUBMISSION IS ALLOWED.
- **Collaboration policy.** Discussions with others are strongly encouraged. However, you should write down your solutions **in your own words**. In addition, for **each and every** problem you have to specify the references (e.g., the Internet URL you consulted with or the people you discussed with, or **screenshot of the full conversation** if you are using GPT) on the **first** page of your solution to that problem. You may get zero points due to the lack of references.
- **Tips for programming problems.** Here are some C++ only tips. For this homework, you might want to check out sections about cin/cout, vectors, pairs, references, and range for. Reference is IMHO the most important since it rids you of C pointers. (Yeah!!!)

Problem 1 - Skipping Classes (Programming) (10 points)

Problem Description

As a student, Grouper lives a simple campus life. Each day, it arrives at school, participates in an arbitrary number of consecutive classes (possibly zero), and goes home. That is to say, Grouper may skip the first several classes and/or the last several classes of the day, but it never skips any classes between attended ones.

In the new semester, there are T school days, and Grouper has N_i classes on the i-th day. For each class, Grouper estimates its *rewardingness*. The j-th class on the i-th day has (estimated) *rewardingness* of $r_{i,j}$. Classes that are fun and interesting, have positive *rewardingness*. Dull and exhausting classes have negative *rewardingness*. Note that it is possible for a class to have an estimated *rewardingness* of 0.

A *plan of attendance* can be described as a pair of integers (a, b), meaning Grouper will attend the a-th class through the b-th class that day. For the i-th day, Grouper decides a threshold K_i, and wants to find an optimal plan (a_i, b_i) such that:

- The sum of *rewardingness* of the classes attended that day is not less then K_i; that is, $\sum_{a_i \le j \le b_i} r_{i,j} \ge K_i$.
- The length of a plan is defined as the number of classes attended. We want to minimize the length of the optimal plan while abiding by the previous constraint; that is, the optimal plan should have a minimum $b_i - a_i + 1$ among all possible plans.

Can you find the length of the optimal plan(s) for Grouper?

Note: Grouper is a type of fish. The word originates from the Portuguese "garoupa" and has nothing to do with the English word "group". Source: Wikipedia.

Input

The first line of the input file contains a single integer T. Then, T group of input follows. The i-th group consists of two lines:

- The first line contains two integer N_i, K_i, separated by a single whitespace.
- The second line contains N_i integers $r_{i,j}$, separated by single whitespaces.

Constraints:

- $1 \le T \le 10^6$
- $\forall i \in [1, T], 1 \le N_i \le 10^6$
- $1 \le \sum_{i=1}^{T} N_i \le 10^6$
- $\forall i \in [1, T], 1 \le K_i \le 10^9$
- $\forall i \in [1, T], \forall j \in [1, N_i], |r_{i,j}| \le 10^9$
- All numbers in the input are integers.

Output

For each group of input, output one line containing a single integer, representing the length of the optimal plan of the day. If there are no possible plans, please output -1.

Test Group 0 (0 %)

- Sample input.

Test Group 1 (10 %)

- $\sum_{i=1}^{T} N_i \leq 5000$

Test Group 2 (10 %)

- $\forall 1 \leq i \leq T, \forall 1 \leq j \leq N_i, r_{i,j} \geq 0$

Test Group 3 (10 %)

- $T = 1$, Answer $\geq N_1 - 5000$ or $= -1$.

Test Group 4 (70 %)

- No additional constraints.

Sample Input 1

```
5
5 6
1 1 1 1 1
5 1
4 -8 7 -6 3
6 10
6 1 1 1 2 5
6 49
7 7 7 7 7 14
7 4
2 -1 3 -2 3 -1 2
```

Sample Output 1

```
-1
1
5
6
3
```

Hint

- In the sample input:
  1. There is no plan satisfying $\sum r_{i,j} \geq 6$, so we should output -1.
  2. $(a, b) \in \{(1, 1), (3, 3), (5, 5)\}$ are all optimal plans.
  3. $(a, b) \in \{(1, 5), (2, 6)\}$ are both optimal plans.
  4. The only optimal plan is $(1, 6)$.
  5. Note that $(a, b) = (1, 7)$ suffice the constraint $\sum r_{i,j} \geq 4$, but we should pick a shorter plan.
- The sum of $r_{i,j}$ may exceed the range of 32-bit `int`. Though `#define int long long` may solve this problem, it is highly recommended to only use `long long` when required. Carefully analyzing the range a variable may need is a better practice and helps debugging sometimes.
- Test group 3: How many possible plans are there? Can you iterate through them all? For some array a, can we calculate $\sum_{i=l}^{r} a_i$ in $O(1)$ time for any given (l, r) after $O(N)$ preprocessing?
- In some test cases, you might have to output up to 10^6 numbers and newline characters. As taught in System Programming, `cout << endl` flushes the buffer of output, which would be time-consuming and may cause TLE even if your algorithm is correct, so please use `cout << \n` instead of `endl`.

Extra questions for those interested

- Test group 2: Can you design an algorithm that only takes $O(N)$ time?
- What about finding a plan with a maximum length satisfying the condition? Is there any solution that is easier to implement? Why can it be easier than solving the original version?
- What about counting suitable plans? Can you reduce this problem to a classic one?

Problem 2 - 8e7 and n by m Grids (Programming) (15 points)

Problem Description

"You have an n by m grid, ……" -8e7

Hehehehehe~ 8e7's favorite thing is n by m grids, so he wants to share his love for n by m grids by setting a problem about them in ADA!

You are a city planner in Okapi City looking to utilize a beautiful coastline nearby. The area around the coastline is given as an n by m grid. Let the cell (i, j) denote the cell on the i-th row from north to south and the j-th column from west to east. Initially, all cells in the grid are "water" cells, and your job is to turn some water cells into "land" cells so that tourist attractions (like hotels and restaurants) can be built on them.

More specifically, the cost to turn cell (i, j) into land is $a_{i,j}$ dollars, due to the safety regulations of the shore, the land cells must be filled with the following rules:

- If cell (i, j) is land $(i \leq n - 1)$, then cell $(i + 1, j)$ must be land.
- If cell (i, j) is land $(j \leq m - 1)$, then cell $(i, j + 1)$ must be land.

Note that we consider the region south of the grid and east of the grid as existing land.

There are K buildings that are planned to be built. The i-th building is located in cell (x_i, y_i) and it has two special numbers: a safety coefficient d_i and the expected revenue r_i. If there is land at all cells (x, y) with $x \in [x_i - d_i, x_i + d_i], y \in [y_i - d_i, y_i + d_i]$, then the building can be constructed and the city would earn r_i dollars in taxes. Note that there may be more than one building located in the same cell.

Given the information above, write a program that can find the maximum amount of money the city could earn.

Input

The first line contains two integers n, m. The next n lines contain m integers each, the j-th integer on the i-th line is $a_{i,j}$. The next line contains one integer K. The next K lines contain four integers each, the i-th line has integers x_i, y_i, d_i, r_i.

- $1 \leq n, m \leq 1500$
- $0 \leq a_{i,j} \leq 10^6, \forall 1 \leq i \leq n, 1 \leq j \leq m$
- $0 \leq K \leq 5 \times 10^5$
- $1 \leq x_i \leq n, 1 \leq y_i \leq m, 0 \leq d_i < \min(x_i, y_i), \forall 1 \leq i \leq K$
- $1 \leq r_i \leq 10^6, \forall 1 \leq i \leq K$

Output

Output one line with an integer, denoting the maximum amount of money (in dollars) the city could earn by filling up some (maybe zero) cells with land.

Test Group 0 (0 %)

- Sample Input.

Test Group 1 (20 %)

- $n = 1$

Test Group 2 (20 %)

- $n \leq 50$

Test Group 3 (20 %)

- $d_i = 0, \forall 1 \leq i \leq K$

Test Group 4 (40 %)

- No additional constraints.

Sample Input 1

```
1 5
4 8 7 6 3
3
1 3 0 15
1 1 0 10
1 5 0 5
```

Sample Output 1

```
4
```

Sample Input 2

```
5 3
1 4 6
0 1 2
0 2 0
4 6 7
4 2 1
3
3 1 0 15
4 2 0 3
5 3 0 2
```

Sample Output 2

```
1
```

Sample Input 3

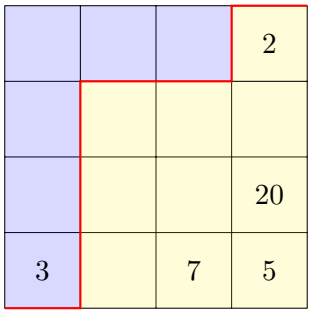
```
4 4
2 4 5 1
9 2 3 7
1 1 2 2
8 2 6 4
5
3 4 1 20
4 3 1 7
4 4 2 5
1 4 0 2
4 1 0 3
```

Sample Output 3

```
4
```

Hint

- The following diagram shows the optimal plan for Sample Input 3. The buildings with revenues 20, 7, 5, and 2 can be built with this plan, so the total earnings is $(20 + 7 + 5 + 2) - (1 + 2 + 3 + 7 + 1 + 2 + 2 + 2 + 6 + 4) = 4$. Note that if the cell $(2, 2)$ wasn't selected, then the building with revenue 5 could not be built.



- Because of the large input size, if you are using `cin` to read input, it is recommended to add the line `ios_base::sync_with_stdio(0);cin.tie(0);` on the first line of the main function to optimize I/O speeds.
- If you are interested in problems with n by m grids, you are more than welcome to contact `8e7` on discord :D

Problem 3 - Kiwi's Fruit Sale (Programming) (15 points)

Problem Description

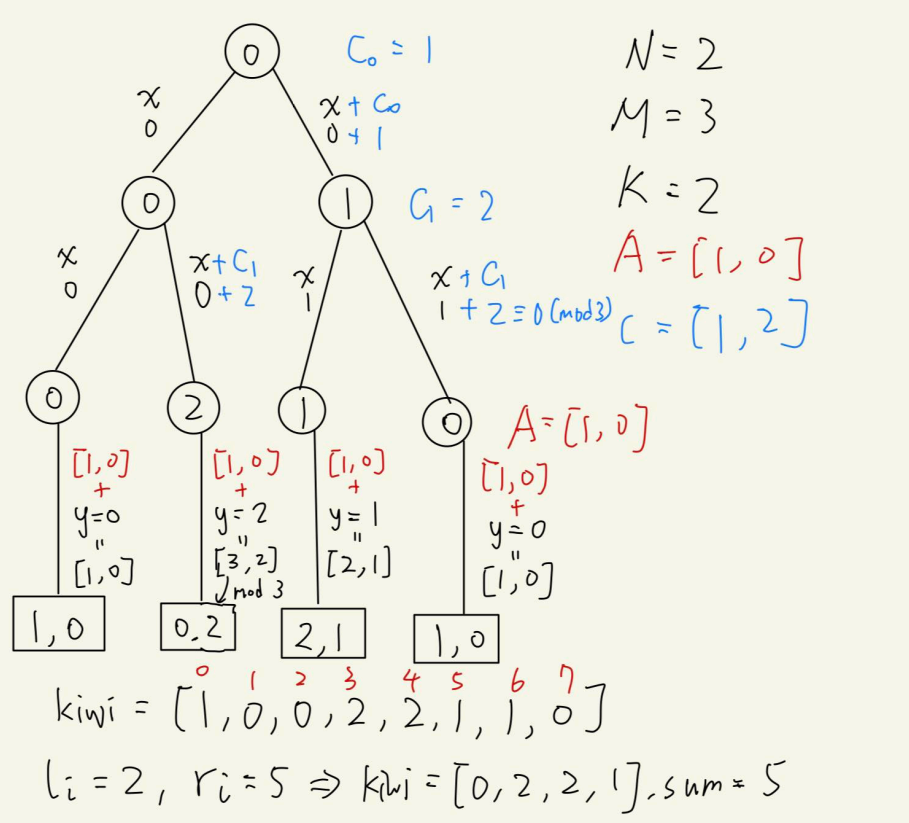
Kiwi is a renowned fruit farmer in the land of the fruit kingdom, and as the name suggests, his favorite fruit is kiwi.

Every year, he grows a massive kiwi tree that produces lots of kiwis, each with varying levels of sweetness.

The tree grows according to the following rules:

- Initially, there is just the trunk of the tree. We call it a **level 0 branch**. This branch has **growth factor** 0.
- The tree then expands for K days. On the i-th day, all level $i-1$ branches grow two level i branches, one of which is the **left branch** and the other is the **right branch**. If a level $i-1$ branch has a growth factor of x, then its left branch has growth factor x, and its right branch has growth factor $(x + c_{i-1}) \mod M$, where c_{i-1} is a **magic number** given for that day.
- On the $(K+1)$-th day, all level K branches grow N kiwis. For a level K branch with growth factor y, it grows N kiwis with sweetness values $(A_0 + y) \mod M, (A_1 + y) \mod M, \ldots, (A_{n-1} + y) \mod M$. Where $A_0, A_1, \ldots, A_{n-1}$ are given values.

For example, if $N = 2, M = 3, K = 2, A = [1, 0], c = [1, 2]$, this means that the tree grows for $K = 2$ days. The growth factors of the level 1 branches are $[0, 1]$ from left to right, the growth factors of the level two branches are $[0, 2, 1, 0]$ from left to right, and the sweetness values of the kiwis are $[1, 0, 0, 2, 2, 1, 1, 0]$ from left to right. As shown in the graph below:



After the tree has grown, there are Q customers who want to purchase the kiwis. The i-th customer is looking to purchase all the kiwis between the range $[l_i, r_i]$ of fruits on the tree **(note that the kiwis are numbered based on 0-index)**, and the customer wants to know the **sum of sweetness values** of all kiwis they would purchase. For example, if $l_i = 2, r_i = 5$, the kiwis that the customer buys have a sum of sweetness values $0 + 2 + 2 + 1 = 5$. However, Kiwi is very busy and doesn't have time to count all the kiwis, so he is asking for your help!

Given the information on the initial array A, the magic numbers c_j and the customer requests l_i, r_i, find the sum of sweetness values for each customer's purchase. Note that the customers are only looking at the kiwis for now, so the array stays the same for each customer.

Note: The expression $a \mod b$ for integers $a \geq 0, b > 0$ gives the remainder when a is divided by b (which is "%" in programming languages). For example, $5 \mod 2 = 1, 6 \mod 3 = 0$, etc.

Input

On the first line, there are three integers N, M, K. The second line has N integers $A_0, A_1, \ldots, A_{N-1}$. The third line has K integers $c_0, c_1, \ldots, c_{K-1}$. The fourth line has one integer Q. The next Q lines each contain two integers, the i-th of which is l_i, r_i.

- $1 \leq N \leq 10^5$
- $1 \leq M \leq 10$
- $0 \leq K \leq 30$
- $0 \leq A_i < M, \forall 0 \leq i \leq N - 1$
- $0 \leq c_j < M, \forall 0 \leq j \leq K - 1$
- $1 \leq Q \leq 10^5$
- $0 \leq l_i \leq r_i < N \times 2^K, \forall 1 \leq i \leq Q$

Output

Output Q lines, the i-th line contains one integer representing the sum of sweetness values the i-th customer would get.

Test Group 0 (0 %)

- Sample Input

Test Group 1 (20 %)

- $K = 0$

Test Group 2 (20 %)

- $c_j = 0, \forall 0 \leq j \leq K - 1$

Test Group 3 (20 %)

- $N = 1$

Test Group 4 (40 %)

- No additional constraints

Sample Input 1

```
2 3 2
1 0
1 2
5
2 5
4 6
0 7
1 3
7 7
```

Sample Output 1

```
5
4
7
2
0
```

Sample Input 2

```
7 5 0
1 4 2 2 0 3 1

3
0 5
1 3
4 6
```

Sample Output 2

```
12
8
4
```

Sample Input 3

```
3 8 8
3 7 3
0 0 0 0 0 0 0 0
4
13 25
5 7
13 15
3 767
```

Sample Output 3

```
59
13
13
3315
```

Sample Input 4

```
1 10 30
0
4 8 7 6 3 2 1 4 7 4 8 3 6 4 7
3 1 4 1 5 9 2 6 5 3 1 0 3 0 5
3
0 1073741823
13938423 772637487
102938274 998244353
```

Sample Output 4

```
4831838304
3414145864
4028877492
```

Hint

- The values might need to be stored using `long long`.
- It seems like there is some sort of repeating structure in the array. If the array is too large, try

cutting it in half and use divide and conquer to answer queries.

Problem 4 - Alexander Loves One Dollar Coins (Programming) (10 points)

The time limit of this problem is adjusted to 0.2s.

Problem Description

Alexander loves one-dollar coins. He has a row of boxes that can store many coins. Alexander chooses some consecutive boxes to decorate his garden every day. Peach, who observes Alexander every day, finds out that Alexander's love towards consecutive boxes is strongly related to two variables, the number of boxes and the number of coins in the boxes. More specifically, Alexander's love towards consecutive boxes between the i-th box and the j-th box can be calculated as

$$f(i,j) = (j-i+1)^2 + \left(\sum_{k=i}^{j} a_k\right)^2, (0 \le i \le j < n).$$

where a_k is the number of coins in the k-th box. Peach wants to find out the minimum of Alexander's love towards consecutive boxes.

Input

For each testcase, there will be two lines: The first line contains an integer n representing the length of the given array a. The second line contains n integers $a_0, a_1, a_2, ..., a_{n-1}$ representing the content of the given array. Since Alexander is not as rich as you think, there might be some boxes with less than zero coins inside ;(

- $1 \le n \le 10^5$
- $-10^4 \le a_i \le 10^4$

Output

For each testcase, print a single integer representing the minimum possible value of Alexander's love.

Test Group 0 (0 %)

- Sample Input.

Test Group 1 (30 %)

- $1 \le n \le 1000$

Test Group 2 (70 %)

- No additional constraints.

Sample Input 1

```
4
-1 0 0 1
```

Sample Output 1

```
1
```

Sample Input 2

```
2
1 -1
```

Sample Output 2

```
2
```

Sample Input 3

```
5
-4 -6 -1 -5 -7
```

Sample Output 3

```
2
```

Problem 5 Asymptotic Notations (Hand-Written) (25 + 5 points)

Correctness proof is needed for all the problems unless otherwise specified!
Correctness proof is needed for all the problems unless otherwise specified!
Correctness proof is needed for all the problems unless otherwise specified!

(a) Finding prime numbers (11 points)

Alexander and Bambooboo want to find all prime numbers between 1 and n. They write down the following algorithms: (Assume the push function of stacks is $O(1)$ operation)

Alexander's algorithm: [edit: fix typo]

```
ans = Empty Stack
for i = 2..n:
    flag = true
    for j = 2..i:
        if j * j > i:
            if flag is true:
                ans.push(i)
            break
        if i % j == 0:
            flag = false
return ans
```

Bambooboo's algorithm:

```
ans = Empty Stack
is_prime = bool array with size n

for i = 2..n:
    is_prime[i] = true

for i = 2..n:
    j = i * 2
    while j <= n:
        is_prime[j] = false
        j += i

for i = 2..n:
    if is_prime[i]:
        ans.push(i)

return ans
```

1. (4pts) What is the time complexity for Alexander's algorithm? Express it in Θ notation.
2. (4pts) What is the time complexity for Bambooboo's algorithm? Express it in Θ notation.
3. (3pts) Who's algorithm run faster when n is very large?

(b) Sorting (3 points)

Please sort the functions in descending order of the complexity, and write down the indices after sorting. That is, sorting the functions for n is very large.

For example, if the problem is (1) n^3 (2) n (3) n^2, then you should answer (1) > (3) > (2) because $n^3 > n^2 > n$ when n is very large.

Your score in this problem will be

$$\max\left(0,\ 3 - 0.6 \times \{(i,j) \mid \text{correct\_answer}[i] < \text{correct\_answer}[j],\ \text{your\_answer}[i] > \text{your\_answer}[j]\}\right)$$

No proof is needed in this problem.

(1) n^{n+1}

(2) $(2n)^{\log n}$

(3) $n! \cdot e^n$

(4) $\sum_{i=1}^{n+1} i^n$

(5) $(\log n)^{2n}$

where n is a positive integer.

(c) Strange recursions (11 + 5 points)

1. (2pts) Assuming $f(n) = 1$ for all $n \leq 1$, and $f(n)$ satisfies the recurrence relation for all real number n greater than 1:
$$f(n) = f\left(\frac{4n}{9}\right) + f\left(\frac{n}{2}\right) + 49n$$
Prove that $f(n) = \Theta(n)$.

2. (4pts) Assuming $f(n) = 1$ for all $n \leq 1$, and $f(n)$ satisfies the recurrence relation for all real number n greater than 1:
$$f(n) = f(an) + f(bn) + n$$
Where a, b are positive real numbers and $a, b < 1$, prove that $a+b < 1$ if and only if $f(n) = \Theta(n)$.

3. (5pts) Assuming $f(n) = 1$ and $g(n) = 1$ for all $n \leq 1$, and $f(n), g(n)$ satisfies the recurrence relation for all real number n greater than 1:
$$\begin{cases} f(n) = a_1 \times f(b_1 n) + a_2 \times g(b_2 n) \\ g(n) = a_3 \times f(b_3 n) + a_4 \times g(b_4 n) \end{cases}$$
Where $a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4$ are positive real numbers, find the constraints on these real numbers such that $f(n)$ and $g(n)$ can be defined for all n. That is, the variables in the functions f and g are no more than 1 after applying several recursions, ensuring that the recursion will terminate.

4. (bonus, 5pts) Assuming $f(n) = 1$ and $g(n) = 1$ for all $n \leq 1$, and $f(n), g(n)$ satisfies the recurrence relation for all real number n greater than 1:
$$\begin{cases} f(n) = 4f\left(\frac{n}{2}\right) + g\left(\frac{2n}{3}\right) + n \\ g(n) = 3f\left(\frac{3n}{4}\right) + 2g\left(\frac{n}{2}\right) + 2n\log n \end{cases}$$
Find the real numbers k, l such that $f(n) = \Theta(n^k)$ and $g(n) = \Theta(n^l)$.
(Hint: you can prove that $k = l$ first)

Problem 6 - Arctan and Inversions (Hand-Written) (25 points)

Note: In all of the following problems, you need not only to construct the algorithm but also to prove the correctness. Proving the correctness includes proving the complexity.

Arctan Loves Inversions

$\{a_i\}_{i=1}^n$ is a sequence of n real numbers. An ordered pair (i, j) where $i < j$ is called an **inversion** if and only if $a_i > a_j$. The number of inversions in the sequence is defined as the number of (i, j) that are inversions, where $1 \leq i < j \leq n$.

For example, consider a sequence $\{a_i\}_{i=1}^5 = 3, 1, 4, 1, 5$. The number of inversions is 3, where the inversions in this sequence are $(1, 2), (1, 4), (3, 4)$ since $a_1 > a_2, a_1 > a_4, a_3 > a_4$.

(a) (4 points)

Find the number of inversions in a sequence $\{a_i\}_{i=1}^n$ in $O(n \log n)$ time complexity.

(b) (5 points)

Arctan is interested in inversions. From each inversion (i, j) in the sequence, Arctan gains $a_i + a_j$ amount of happiness.

Find the total amount of happiness Arctan will gain from a sequence $a_1, a_2, \ldots, a_n$ in $O(n \log n)$ time complexity.

(c) (5 points)

Arctan becomes crazy in inversions. The amount of happiness Arctan gains from inversions grows exponentially day by day. That is, in the k-th day, Arctan will gain $(a_i + a_j)^k$ amount of happiness from each inversion (i, j).

Find the total amount of happiness Arctan will gain from a sequence $\{a_i\}_{i=1}^n$ in the k-th day in $O(nk \log n)$ time complexity.

Arctan Loves Stacks

Arctan's favorite data structure is the stack because it is like an inversion machine. The order of elements being popped out of a stack is exactly the inverted order of being pushed into. He loves stacks so much that he wants to solve every data structure problem with it. However, his passion for stack gets him into trouble. He can't finish his Data Structures and Algorithms homework only using stacks.

In Arctan's homework, he wants to use a stack to simulate n operations (insertions and deletions) on a set in $O(n \log n)$ time complexity. It is known that this can be easily done by a treelike data structure, but challenging by a stack. More precisely, the problem is as follows:

The set A is initially empty. There are n operations on A with order, each operation is one of the following:

- insert an element x (that is, $A \leftarrow A \cup \{x\}$, and it is guaranteed that $x \notin A$ before the insertion).
- delete an element x (that is, $A \leftarrow A \setminus \{x\}$, and it is guaranteed that $x \in A$ before the deletion).

Arctan wants to use a stack S to simulate A. S is initially empty. After each operation on A, several push and pop operations on S should be performed so that the set of the elements in S is equal to the current state of A.

To avoid confusion, we'll call operations on A (insert or delete) "set operations", and operations on S (push or pop) "stack operations".

Note that in the following problems, we only consider the number of stack operations instead of time complexity for convenience since those which are not stack operations can be easily performed in $O(n \log n)$ time complexity.

(d) (4 points)

The following is Arctan's implementation. He thought he was correct, but he got TLE (time limit exceeded) on the judge. He performed the set operations by their order, and the following is his algorithm when facing a set operation:

```
if the set operation is to insert an element x:
    push the element to the top of S
else if the set operation is to remove an element x:
    pop elements from the top of S until x is popped
    push all popped elements (in this round) instead of x back to the top of S in a
    special order
```

Find a sequence of n set operations so that Arctan's algorithm would use $\Theta(n^2)$ stack operations no matter what the special order he pushed back is. The special order may be related to any set operations since Arctan knows all set operations before performing any of them. Your answer should be n ordered set operations. For the proof of correctness, you need to check that the minimum possible number of stack operations to finish the n set operations you designed is an $\Theta(n^2)$ function.

For example, if your answer is

```
insert 1
insert 2
...
insert n-2
delete 1
delete 2
```

The minimum possible number of stack operations to finish these set operations is $(n-2)+(2n-5)+1 = 3n-6$, where $(n-2)$ represents the $(n-2)$ pushes in the inserts, the $(2n-5)$ represents $(n-2)$ pops and $(n-3)$ pushes while doing the "delete 1" operation (after $n-2$ inserts, 1 should be at the bottom of the stack and you need $(n-2)$ pops to remove the 1, after that you can push $2 \sim n-2$ back to the stack in the order you want). 1 represents that you can finish the "delete 2" operation in one pop since you can put 2 at the top of the stack in the previous operation. However, this is not a valid answer since the number of stack operations is $3n-6 = \Theta(n)$, not $\Theta(n^2)$.

(e) (7 points)

Find a way to complete any sequence of n set operations in $O(n \log n)$ stack operations.

In another word, $\forall$ sequence of sets $A_0, A_1, \ldots, A_n$ where $A_0 = \emptyset$ and $\forall 0 \le i \le n-1$, $|A_i \triangle A_{i+1}| = 1$, find a correspond sequence of stacks $S_0, S_1, \ldots, S_m$ where $m = O(n \log n)$ such that:

1. $\forall 0 \le i \le n-1$, S_{i+1} is formed from S_i through a stack operation.
2. $\exists i_0, i_1, \ldots, i_n$ where $0 = i_0 < i_1 < i_2 < \cdots < i_n \le m$ s.t. $\forall 0 \le j \le n$, $A_j = S_{i_j}$.

Remark: $A \triangle B := (A \cup B) \setminus (A \cap B)$ is called the symmetric difference of A, B.