LAB6

Question 1: Flappy Bird

After surviving several midterm exams, you decided to play flappy bird to have some relax. You transfer the flappy bird game map into a coordinate system. In the game, the bird is at point (0,0) at first, your aim is to let it reach any point with x-coordinate *X*. In the map, there are *n* barriers, which is described by three parameters *x, a, b*. It means that when the bird is at the point with x-coordinate *x*, if the y-coordinate *y* satisfies *y*<=*a* or *y*>=*b*, you will lose the game. Every second, you can choose to click the screen or not. If you don't click the screen, the bird will fly from (*x*, *y*) to (*x*+1, *y*-1). If you click the screen, the bird will fly from (*x*, *y*) to (*x*+1, *y*+1). Since you are so lazy to move your finger, you want to know the minimum times you need to click the screen.

Input:

The first line: *n* *X* (0<=*n*<=50000), (0<=*X*<=10^9)

The 2nd to (n+1)th line: *x* *a* *b* (0<*x*<*X*), (-10^9<=*a*, *b*<=10^9)

Output:

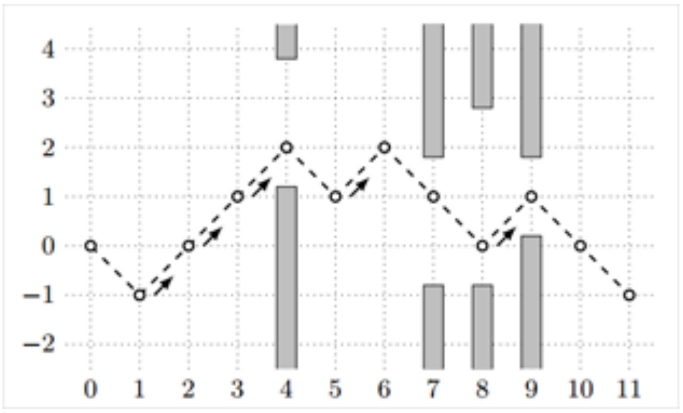
The minimum time you click the screen. If it is impossible to reach the goal, output "NIE" instead.

Sample Input

```
4 11
4 1 4
7 -1 2
8 -1 3
9 0 2
```

Sample Output

```
5
```



Question 2: Dynamic Size Array

Considering you only have 1 day off this weekend and having Chemistry midterm exam soon, TA Group decides to extend the due day of Question 2 to a week later. Next week lab will only grade Question 1, but you may start early on Question 2.

It is known that, in C programming language, the size of array (local variable) must be a constant. However, malloc() accepts a variable as the size, and we will use it to implement a dynamic size array in this problem.

Definition

Dynamic size array: the array will allocate some memory with the size of *s*, and put elements inside the array. When the array is full, but user keep asking putting element inside, the dynamic size array will allocate another pieces of memory with the size of *2s*, and copy the original elements in the array to the new array, and then put the new element into the new array.

Implementation

You need to realize the following functions:

- num_elements(): return the number of elements in the dynamic size array
- available_capacity(): return the number of available positions in the array (the number of elements the user can keep putting in without reallocate memory)
- push_back(int x): add an element x to the end of the dynamic size array. If the array is full, then reallocate another array with double size and copy elements into it, and then put in the new element.
- pop_back(): delete the last element in the dynamic size array. Print out the deleted number, or output error message when the array is empty.

- is_empty(): return 1 if the the dynamic size array has no element; 0 otherwise
- sum(): return the sum of all the elements in the array
- reserve(int s): manually allocate the array with capacity of *s*, and copy elements into the new array. You may assume *s* is always greater than the number of elements in the array.

You may initialize the dynamic array with 0 elements inside and 1 capacity.

You may design the functions as you like, and it is not necessary to follow the function prototypes listed above.

Hint: You may assume the elements in the array are integer, and use an int* to indicate the dynamic size array.

Output

Each time you should output two rows:

The first row: Ask the user to choose function.

The second row: The corresponding message.

Sample

```
Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),
5. is_empty, 6. sum, 7. reserve, 8. quit) 2
The available camapcity is 1.
Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),
5. is_empty, 6. sum, 7. reserve, 8. quit) 3
Please input a number: 24
Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),
5. is_empty, 6. sum, 7. reserve, 8. quit) 1
```

```
The array contains 1 element(s).

Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),

5. is_empty, 6. sum, 7. reserve, 8. quit) 7

How large should the array be? 16

Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),

5. is_empty, 6. sum, 7. reserve, 8. quit) 2

The available camapcity is 15.

Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),

5. is_empty, 6. sum, 7. reserve, 8. quit) 5

The array is not empty.

Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),

5. is_empty, 6. sum, 7. reserve, 8. quit) 6

The sum is 24.

Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),

5. is_empty, 6. sum, 7. reserve, 8. quit) 4

Number 24 deleted.

Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),

5. is_empty, 6. sum, 7. reserve, 8. quit) 4

Error: array already empty.

Please choose function: (1. num_elements, 2. available_capacity, 3. push_back, 4. pop_back(),

5. is_empty, 6. sum, 7. reserve, 8. quit) 8

Goodbye!
```