Q1

1. Question description:

1 02 Representaion (20% of this assignment)

1.1 Problem Description

Each positive integer n corresponds to a unique binary representation (**specifying that binary numbers are written from right to left, in order of most significant digit to least significant digit**).

For example: $10 = 8 + 2 = 2(3) + 2(1)$

For ease of writing, "a(b)" is used in this question to denote "a to the power of b", i.e., a^b.

Specifying the 02 representation of a number can be obtained by the following procedure:

1. substitute a number that is not 0 or 2 with its binary representation, additionally, using 2(0) instead of 1.
2. check whether the result contains only 0 and 2, and if it contains numbers other than 0 and 2, repeat the step 1 for these numbers

It can be proved that the representation is *unique* after the above steps.

Take 137 as an example:

- First round: $137 = 2(7) + 2(3) + 2(0)$, 7 and 3 do not meet the requirements
- Second round:
  - Substitute 7 and 3 with their binary expressions: $7 = 2(2) + 2 + 2(0)$, $3 = 2 + 2(0)$
  - Thus, $137 = 2(2(2) + 2 + 2(0)) + 2(2 + 2(0)) + 2(0)$

So the "02 representation" of 137 is $2(2(2) + 2 + 2(0)) + 2(2 + 2(0)) + 2(0)$, remember that binary numbers are written from right to left, in order of most significant digit to least significant digit

Your task is to write a program that reads a positive integer n and outputs the 02 representation of the given number

1.2 Problem Description

An integer n, the number to be represented in 02 form (ensure that $1 \leq n \leq 10^9$).

Hint: For C/C++ and Java users, an `int` type stores integers range from -2,147,483,648 to 2,147,483,647. So it is possible to store n with an `int` type.

1.3 Output

One line, the 02 representation of n, (**no spaces between characters**).

Trailing spaces and newlines after the last character are ok.

Sample Input I

```
7
```

Sample Output I

```
2(2)+2+2(0)
```

Sample Input II

```
137
```

Sample Output II

```
2(2(2)+2+2(0))+2(2+2(0))+2(0)
```

Problem Scale & Subtasks

For 30% of the testcases $1 \leq n \leq 200$

For 100% of the testcases $1 \leq n \leq 10^9$

Test Case No.	Constraints
1	$n = 7$
2	$n \leq 200$
3	$n = 137$
4-10	No additional constraints

2. Solution:

```cpp
#include<iostream>
#include<cmath>
using namespace std;
string representation02(int num){
    string result = "";
    int logNum;
    if (num == 0){
        return "0";
    }else{
        if (num <= 7){
            logNum = log2(num+0.0);
            if(logNum == 1){
                result += "2";
            }else{
                result += "2("+to_string(logNum)+")";
            }
            num -= pow(2, logNum);
            while(num > 0){
                logNum = log2(num+0.0);
                if(logNum == 1){
                    result += "+2";
                }else{
                    result += "+2("+to_string(logNum)+")";
                }
                num -= pow(2, logNum);
            }
            return result;
        }else{
            logNum = log2(num+0.0);
            result += "2("+representation02(logNum)+")";
            num -= pow(2, logNum);
            while(num > 0){
                logNum = log2(num+0.0);
                if(logNum == 1){
                    result += "+2";
                }else{
                    result += "+2("+representation02(logNum)+")";
                }
                num -= pow(2, logNum);
            }
            return result;
        }
    }
}

int main(){
    int num;
    cin >> num;
    cout << representation02(num) << endl;
}
```

3. Thinking:

(1) represent num as 2^(a1)+2^(a2)+2^(a3)... s.t.a1>a2>a3...

(2) print 2(representation02(a1))+2(representation02(a2))+... //representation02(a1) means representing a1 as 2^(a1')+2^(a2')+2^(a3')... s.t.a1'>a2'>a3'...

(3) use this recursion utill the number in the representation() is less than 8, because the number must be represented by 2(2), 2 and 2(0)

p.s. we can get a1 by log2(num) (get the integer part), and get a2 by num -= 2^(a1) and log2(num)(get the integer part).

Q2

1. Question description:

2.1 Description

A terminal is a row of n equal segments numbered 1 to n in order. There are two parallel terminals, one above the other.

You are given an array a of length n. For all $i = 1, 2, , n$, there should be a straight wire from some point on segment i of the top terminal to some point on segment a_i of the bottom terminal. You can't select the endpoints of a segment. For example, the following pictures show two possible wirings if $n = 7$ and $a = [4, 1, 4, 6, 7, 7, 5]$.

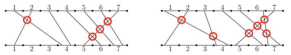

Figure 1: A crossing occurs when two wires share a point in common. In the picture above, crossings are circled in red.

What is the maximum number of crossings there can be if you place the wires optimally?

2.2 Input

The first line contains an integer n ($1 \le n \le 10^5$), representing length of the array.

Then follows 1 line with n numbers, representing the array. It is guaranteed ($1 \le a_i \le n$)

2.3 Output

Output one integer representing the maximum number of crosses.

Hint (For C/C++ and Java users): The result may exceed the range of `int` type, you are recommended to use `long long` (in C/C++) or `long` (in Java) to store the result.

Sample Input I

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

Sample Output I

```
6
```

Sample Input II

```
3
2 2 2
```

Sample Output II

```
3
```

Problem Scale & Subtasks

For 30% of the testing data, $n \le 1,000$

For 100% of the testing data, $1 \le n \le 100,000$

Test Case No.	Constraints
1-4	$n \le 1000$
5-8	a is a a permutation, i.e., a contains all integers from 1 to n exactly once.
9-10	No additional constraints

2. Solution:

```cpp
#include<iostream>
#include<string>
#include<sstream>
using namespace std;

int arr[100000];

long long inversions(int start, int end){
    int mid = (start + end) / 2;
    int sortedArr[end+1];
    int leftIndex = start;
    int rightIndex = mid + 1;
    int sortedArrIndex = 0;
    long long ans = 0;
    if ((end-start+1) <= 1){
        return 0;
    }
    ans += inversions(start, mid)+inversions(mid + 1, end);
    while (leftIndex <= mid && rightIndex <= end){
        if (arr[leftIndex] >= arr[rightIndex]) sortedArr[sortedArrIndex++] = arr[rightIndex++];
        else{
            sortedArr[sortedArrIndex++] = arr[leftIndex++];
            ans += rightIndex - mid - 1;
        }
    }
    while (leftIndex <= mid){
        sortedArr[sortedArrIndex++] = arr[leftIndex++];
        ans += rightIndex - mid - 1;
    }
    while (rightIndex <= end) sortedArr[sortedArrIndex++] = arr[rightIndex++];
    for (int i = 0, j = start; j <= end; ++i, ++j){
        arr[j] = sortedArr[i];
    }
    return ans;
    }


int main(){
    string firstLine;
    string secondLine;
    int tmp;
    int n;
    int arrIndex = 0;

    getline(cin, firstLine);
    istringstream is_n(firstLine);
    while(is_n>>tmp){
        n = tmp;
    }
    getline(cin, secondLine);
    istringstream is_v(secondLine);
    while(is_v>>tmp){
        arr[arrIndex] = tmp;
        arrIndex++;
    }

    cout << inversions(0, n-1) << endl;

    return 0;
}
```

3. Thinking:

(1) the nature of the question is counting “inversions”, but the two same number should also be counted.

(2) we use the method of like merge sort, firstly, divide them, and then merge them and sort them while counting the “inversions”

Q3

1. Question description:

3 Sierpiński carpet (20% of this assignment)

3.1 Description

You are required to write a program to print out the Sierpiński carpet of size $3^k \times 3^k$.

Construction of Sierpiński carpet (quoted from wikipedia):

The construction of the Sierpiński carpet begins with a square. The square is cut into 9 congruent subsquares in a 3-by-3 grid, and the central subsquare is removed. The same procedure is then applied recursively to the remaining 8 subsquares, ad infinitum.

3.2 Input

A positive integer k, ensure that $1 \le k \le 7$

3.3 Output

Sierpiński carpet of size $3^k \times 3^k$. ' ' (SPACE) indicates the area is removed, and '#' indicates the area remains.

Trailing spaces and newlines after the last character are ok.

Sample Input I

```
1
```

Sample Output I

```
###
# #
###
```

Sample Input II

```
2
```

Sample Output II

```
#########
# ## ## #
#########
###   ###
# #   # #
###   ###
#########
# ## ## #
#########
```

Problem Scale & Subtasks

For 100% of the test cases, $1 \le k \le 7$

Test Case No.	Constraints
1-3	$k \le 3$
4-7	No additional constraints

2. Solution:

```cpp
#include<iostream>
#include<cmath>

using namespace std;
bool arr[2187][2187];//2187 = 3^7, true is for "#", false is for " ".

void setArr(int k){
    if (k == 0){
        arr[0][0] = true;
    }
    else{
        setArr(k-1);
        //set first 3^(k-1) rows: copy arr[0,2,...,3^(k-1)-1][1,2,...,3^(k-1)-1] 2 times.
        for (int columnIndex = pow(3, k-1); columnIndex <= (int)(pow(3, k) - 1); columnIndex++){
            for (int rowIndex = 0; rowIndex <= (int)(pow(3, k-1) - 1); rowIndex++){
                arr[rowIndex][columnIndex] = arr[rowIndex][columnIndex - (int)pow(3, k-1)];
            }
        }
        //set first 3^(k-1) columns
        for (int rowIndex = pow(3, k-1); rowIndex <= (int)(pow(3, k) - 1); rowIndex++){
            for (int columnIndex = 0; columnIndex <= (int)(pow(3, k-1) - 1); columnIndex++){
                arr[rowIndex][columnIndex] = arr[rowIndex - (int)pow(3, k-1)][columnIndex];
            }
        }
        //set second 3^(k-1) columns
        for (int rowIndex = (int)(pow(3, k-1)); rowIndex <= (int)(2*pow(3, k-1) - 1); rowIndex++){
            for (int columnIndex = (int)(pow(3, k-1)); columnIndex <= (int)(2*pow(3, k-1) - 1); columnIndex++){
                arr[rowIndex][columnIndex] = false;
            }
        }
        for (int rowIndex = (int)(2*pow(3, k-1)); rowIndex <= (int)(pow(3, k) - 1); rowIndex++){
            for (int columnIndex = (int)(pow(3, k-1)); columnIndex <= (int)(2*pow(3, k-1) - 1); columnIndex++){
                arr[rowIndex][columnIndex] = arr[rowIndex - (int)(2*pow(3, k-1))][columnIndex];
            }
        }
        //set third 3 3^(k-1) columns
        for (int rowIndex = pow(3, k-1); rowIndex <= (int)(pow(3, k) - 1); rowIndex++){
            for (int columnIndex = (int)(2*pow(3, k-1)); columnIndex <= (int)(pow(3, k) - 1); columnIndex++){
                arr[rowIndex][columnIndex] = arr[rowIndex - (int)pow(3, k-1)][columnIndex];
            }
        }
    }
}


int main(){
```

```cpp
int main(){
    string line = "";
    int k;
    cin >> k;
    setArr(k);

    for (int rowIndex = 0; rowIndex < (int)(pow(3, k)); rowIndex++){
        for (int columnIndex = 0; columnIndex <= (int)(pow(3, k)); columnIndex++){
            if (columnIndex == (int)(pow(3, k))){
                line += "\n";
            }
            else{
                if (arr[rowIndex][columnIndex]){
                    line += "#";
                }else{
                    line += " ";
                }
            }
        }
        cout << line;
        line = "";
    }
    cout << endl;
}
```

3. Thinking:
   (1) we find that the figure of k=i is made by copying the figure of k=i-1 9 times ,displaying them as 3*3 square and delete the center one. In order to faster, we can directly copy 8 times, do not delete.
   (2) we use bool arr[][], true for “#”, false for “ “. we make the arr of k = 0 is arr[][]={ true }. Then the arr of k=1 is made by the following operations.
      ① true -> true true true -> true true true -> true true true -> true true true
      true true true true true
   (3) the similar to other k, just use recursion.
   (4) in the end, print them according to the array.

Q4

1. Question description:

4.1 Description

You are required to maintenance an array, which can do the following operations.

1. insert an element with value x after position k. (After this operation, element with value x will be the $k+1$-th element in the array and $k+1$-th element will be moved to position $k+2$ and so on).
2. delete the k-th element in the array. (After this operation, the k-th element will be removed and $k+1$-th element will be moved to position k and so on).
3. Calculate the sum from the l-th element to the r-th element.

In this problem, it is guaranteed that all the number k is randomly generated with equal possibility from all legal values.

(Hint: For a tree structure with n vertex whose root is vertex 1, if vertex i's parent is randomly chosen form $[1, i-1]$, then the expected height of the tree is $O(\log n)$).

4.2 Input

The first line contains an integer n ($1 \le n \le 2 \times 10^5$), representing the number of operations.

Then follows n lines, with each line contains several integers. The first integer is the type of operation.

- If it is 1, then follows two integers k ($0 \le k \le$ len(array)), x ($1 \le x \le 10^9$).
- If it is 2, then follows one integer k ($1 \le k \le$ len(array)).
- If it is 3, then follows two integers l, r ($1 \le l \le r \le$ len(array)).

Note: In operation of type 1, if $k = 0$, the new element x is inserted at the very beginning of the array, i.e., after the insertion, x should be the first element of the array.

4.3 Output

For each operation with type 3 output one integer in one line representing the answer of this operation.

Hint (For C/C++ and Java users): The result may exceed the range of `int` type, you are recommended to use `long long` (in C/C++) or `long` (in Java) to store the result.

Sample Input I

```
6
1 0 1
1 0 2
1 1 4
3 1 1
3 2 2
3 1 3
```

Sample Output I

```
2
4
7
```

Sample Input II

```
10
1 0 2
1 1 5
1 1 4
1 1 1
3 1 3
2 3
3 1 3
1 2 9
2 1
3 1 3
```

Sample Output II

```
7
8
15
```

Problem Scale & Subtasks

For 100% of the testing data, $1 \le n \le 2 \times 10^5$

Test Case No.	Constraints
1-3	$n \le 2,000$
4-6	there's no operation with type 2.
7-10	No additional constraints

2. Solution:

```python
class Node:
    def __init__(self, v):
        self.leftSum = v
        self.val = v
        self.left = None
        self.right = None
        self.pos = 1
root = None
def findNode(root, p):
    if root == None:
        return Node(0)
    else:
        if p < root.pos:
            return findNode(root.left, p)
        elif p == root.pos:
            return root
        else:
            return findNode(root.right, p-root.pos)

def insert(root, p, v):
    if root == None:
        return Node(v)
    else:
        if p < root.pos:
            root.pos += 1
            root.leftSum += v
            root.left = insert(root.left, p, v)
        else:
            root.right = insert(root.right, p - root.pos, v)
    return root

def remove(root, p):
    if root == None:
        return None
    else:
        v = findNode(root, p).val
        if p < root.pos:
            root.pos -= 1
            root.leftSum -= v
            root.left = remove(root.left, p)
        elif p > root.pos:
            root.right = remove(root.right, p -root.pos)
        else:
            if root.left != None and root.right != None:
                tmp = root.left
                root.pos -= 1
                root.leftSum -= v
                while tmp.right != None:
                    tmp = tmp.right
                root.val = tmp.val
                root.left = remove(root.left, p-1)
            elif (root.left == None and root.right == None):
                root = None
            else:
                if root.left != None:
                    root = root.left
                else:
                    root = root.right
    return root

def suml(root, l):
    if root.pos == l:
        return root.leftSum
    elif root.pos < l:
        return root.leftSum+suml(root.right, l-root.pos)
    else:
        return suml(root.left, l)
operationNum = int(input())
for i in range(operationNum):
    op = input().split()
    if int(op[0]) == 1:
        root = insert(root, int(op[1]), int(op[2]))
    elif int(op[0]) == 2:
        root = remove(root, int(op[1]))
    else:
        nodel = findNode(root, int(op[1]))
        print(suml(root, int(op[2])) - suml(root,int(op[1])) + nodel.val)
```

3. Thinking:

(1) we use binarySearchTree

① self.pos: let self.pos which is key be the relative position of node, namely, leftChild size + 1. Becase all the nodes whose key are less than the root's will go to the left to the root, the leftChild size is the number of nodes before root. leftChild size +1 is then the position of the root, namely, root is in the (leftChild+1)th element in the array. Similar to the other node, just consider the other node as root.

② self.leftSum: it is the sum of all the nodes in the left to the root and root itself. eg. If root is the ninth element in the array, namely, self.pos is 9. self.leftSum is then the sum of first 9 elements in the array. For the other node, it is recursive.

③ self.val is value, self.left is leftChild, self. right is rightChild.

(2) insert:

① it is similar to the normal bst, we check if position of the new element < nowNode.pos.

1) If yes, nowNode.pos++, nowNode.leftSum += value, nowNode move to left.

2) If no, position -= nowNode.pos, nowNode move to right directly.

(3) remove:

① it is similar to insert, but we first use findNode to get the value of the element which will be removed.

② we check if position of the new element < nowNode.pos.

1) If yes, nowNode.pos--, nowNode.leftSum -= value, nowNode move to left.

2) If no,position -= nowNode.pos, nowNode move to right .

(4) findNode is to find (p)th node in the array.

(5) suml is sum of first l elements in some tree or childTree. The sequence is according to infix order.

① If position of the new element < nowNode.pos. , return suml(nowNode.left, position)

② If equal, return nowNode.leftSum

③ if >, return nowNode.leftSum + suml(nowNode.right, position - nowNode.pos)

If any questions, email at 121090406@link.cuhk.edu.cn.