

# Dynamic Collection

[Problem](#)[Submissions](#)[Leaderboard](#)[Discussions](#)

Se cuenta con una colección de elementos que son números enteros positivos, sobre la cual se permiten elementos repetidos. Sobre la colección se permiten dos tipos de operaciones. La operación **1** para actualizar la colección y la operación **2** para consultar la colección. El formato de las dos operaciones es el siguiente:

- **1  $k$**
- **2  $a$   $b$**

Para la operación **1**,  $k$  es un número entero sobre el cual se realiza una de las siguientes tres acciones:

- Si el elemento  $k$  se encuentra en la colección entonces no se hace nada
- Si el elemento  $k$  es más grande que cualquiera de los elementos en la colección entonces se adiciona a la colección
- Si el elemento  $k$  no encuentra en la colección entonces se reemplaza por  $k$  la primera ocurrencia del elemento más pequeño que es más grande que  $k$

Para la operación **2**,  $a$  y  $b$  son dos números enteros ( $a \leq b$ ) sobre los cuales se debe consultar en la colección cuantos elementos están en dicho rango.

Por ejemplo, sea la colección que inicialmente tiene diez elementos:

$$C = [7, 1, 7, 1, 3, 9, 7, 9, 10, 4]$$

después de la operación "**2 2 8**" el resultado generado es **5**.

Después de la operación "**1 8**" la colección queda como:

$$C = [7, 1, 7, 1, 3, 8, 7, 9, 10, 4]$$

Ahora al volver a lanzar la consulta "**2 2 8**" el resultado generado es **6**.

Después de la operación "**1 20**" la colección queda con once elementos:

$$C = [7, 1, 7, 1, 3, 8, 7, 9, 10, 4, 20]$$

## Input Format

La entrada consiste de un único caso de prueba. La primera línea contiene dos números enteros positivos  $n$   $q$  ( $1 \leq n, q \leq 10^6$ ) los cuales representan respectivamente la cantidad de elementos de la colección y el total de operaciones a realizarse sobre la colección. La segunda línea contiene los  $n$  elementos de la colección separados por un único espacio en blanco, cada elemento de la colección es un número entero positivo en el rango  $[1, 10^9]$ . Las siguientes  $q$  líneas del caso de prueba son las operaciones, las

cuales pueden ser del tipo **1** o del tipo **2**, con el siguiente formato: **1**  $k$  o **2**  $a$   $b$ , con  $k, a, b \in [1, 10^9]$  y  $a \leq b$ .

### Constraints

$1 \leq n, q \leq 10^6$   
 $k, a, b \in [1, 10^9]$

### Output Format

Para cada operación del tipo **2** (operación de consulta) en la entrada del caso de prueba se debe imprimir una sola línea que contenga un número entero positivo  $r$  el cual representa el total de elementos en la colección que están en el rango  $[a, b]$ .

### Sample Input 0

```
10 11
7 1 7 1 3 9 7 9 10 4
2 2 8
1 8
2 2 8
2 1 20
1 20
2 1 20
2 7 12
1 5
2 7 12
1 12
2 7 12
```

### Sample Output 0

```
5
6
10
11
6
5
6
```

[f](#) [t](#) [in](#)

Submissions: 50

Max Score: 100

Difficulty: Medium

Rate This Challenge:

☆☆☆☆☆

[More](#)

C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #define MAXN 2000000
5 #define infinity 2147483647
6
```

```
7 int collection[MAXN + 1];
8
9 int BinarySearch(int A[], int i, int j, int k) {
10     int m, result = -1;
11
12     while (i <= j) {
13         m = (i + j) >> 1;
14
15         if (A[m] == k) {
16             result = m;
17             break;
18         } else if (k > A[m]) {
19             i = m + 1;
20         } else {
21             j = m - 1;
22         }
23     }
24
25     if (result == -1)
26         result = (-1) * i - 1;
27
28     return result;
29 }
30
31 int BinarySearchFirstOccurrence(int A[], int i, int j, int k) {
32     int result, result2;
33     result = BinarySearch(A, i, j, k);
34
35     if (result >= 0) {
36         result2 = BinarySearch(A, i, result - 1, k);
37         while (result2 >= 0) {
38             result = result2;
39             result2 = BinarySearch(A, i, result - 1, k);
40         }
41     }
42     return result;
43 }
44
45 int BinarySearchLastOccurrence(int A[], int i, int j, int k) {
46     int result, result2;
47     result = BinarySearch(A, i, j, k);
48
49     if (result >= 0) {
50         result2 = BinarySearch(A, result + 1, j, k);
51         while (result2 >= 0) {
52             result = result2;
53             result2 = BinarySearch(A, result + 1, j, k);
54         }
55     }
56     return result;
57 }
58
59 void myMerge (int A[], int p, int q, int r) {
60     int n1 = q - p + 1, n2 = r - q, i, j, k;
61     int L[n1+2], R[n2+2];
62
63     for (i = 1; i <= n1; i++) L[i] = A[p+i-1];
64
65     for (j = 1; j <= n2; j++) R[j] = A[q+j];
66 }
```

```
67     L[n1+1] = infinity;
68     R[n2+1] = infinity;
69     i = 1;
70     j = 1;
71
72     for(k = p; k <= r; k++) {
73         if (L[i] <= R[j]) {
74             A[k] = L[i];
75             i++;
76         } else {
77             A[k] = R[j];
78             j++;
79         }
80     }
81 }
82
83 void MergeSort (int A[], int p, int r) {
84     int q;
85     if (p < r) {
86         q = (p + r) / 2;
87         MergeSort(A, p, q);
88         MergeSort(A, q + 1, r);
89         myMerge(A, p, q, r);
90     }
91 }
92
93 int option1(int k, int collection[], int n){
94     int position = BinarySearch(collection, 1, n, k), isAdded = 0;
95
96     if (position < 0){
97         position = (-1) * position - 1;
98
99         if(position > n){
100             collection[n + 1] = k;
101             isAdded = 1;
102         }
103         else
104             collection[position] = k;
105     }
106     return isAdded;
107 }
108
109 int main (){
110     int n, q, a, b, k, selection, i, idQuery, newSize, left, right, r;
111
112     scanf("%d %d", &n, &q);
113
114
115     for(i = 1; i <= n; i++)
116         scanf("%d", &collection[i]);
117
118     MergeSort(collection, 1, n);
119
120     for(idQuery = 1; idQuery <= q; idQuery++){
121
122         scanf("%d", &selection);
123
124         if(selection == 1){
125
126             scanf("%d", &k);
```

```
127     newSize = option1(k, collection, n);
128     if (newSize == 1)
129         n = n+1;
130 }
131
132 else{
133     scanf("%d %d", &a, &b);
134     left = BinarySearchFirstOccurrence(collection, 1, n, a);
135     righth = BinarySearchLastOccurrence(collection, 1, n, b);
136
137     if(left < 0)
138         left = -1 * left - 1;
139
140     if(righth < 0)
141         righth = -1 * righth - 2;
142
143     r = righth - left + 1;
144
145     printf("%d\n", r);
146 }
147 }
148 return 0;
149 }
```

Line: 1 Col: 1

[Upload Code as File](#) ☐ **Test against custom input**

Run Code

Submit Code