# The battle of TAs

The TAs of **ISS** and **DSA** are locked in a fierce battle! Neel and Kevin, the ISS TAs, are struggling to keep up and are very bad with giving marks, while Mr. P and Mr. J, the DSA TAs, are very good and helpful. You, as a loyal supporter of the DSA TAs, want to help Mr. P and Mr. J crush Neel and Kevin in this epic clash.

The game works as follows:

1. **DSA TAs' turn (Mr. P and Mr. J):** They append an integer to an initially empty array and sort it.
2. **ISS TAs' turn (Neel and Kevin):** After the array is sorted, Neel and Kevin challenge the DSA TAs with a pair of integers `(x, k)` and ask them to find the element that appears `k` positions after the element `x` in the sorted array.

If `x` is not present in the array, or if the required position goes out of bounds, the DSA TAs should return `-1`. Help Mr. P and Mr. J solve this challenge and secure victory!

## Input Format

1. The first line contains an integer `q` (1 ≤ `q` ≤ `10^5`), the number of queries.
2. Each of the next `q` lines contains one of the following:
   - `1 x` (1 ≤ `x` ≤ `10^9`): Insert `x` into the BST.
   - `2 x k` (-`10^5` ≤ `k` ≤ `10^5`): Find the element at index `k + index of x` in the inorder traversal.
3. The elements given will be unique. Also the input would be given such that the tree height remains log(n). Skewed tree won't be given.

## Output Format

For each query of type `2 x k`, print the required element in a new line. If `x` is not found or the index is out of bounds, print `-1`.

## Constraints

The problem has two batches:

Submit solution

My submissions
All submissions
Best submissions

**Points:** 100 (partial)
**Time limit:** 1.0s
**Memory limit:** 256M

**Allowed languages**
C

2. **Batch 2:** No constraint on `nq`, but `n` ≤ `10^5` and `q` ≤ `2*10^5`.

---

## Helper Code

You may use the following snippets for insertion and find.

Copy

```
    int data;
    int size;
    struct node *left;
    struct node *right;
};

struct node *create_node(int data) {
    struct node *new_node = (struct node *)malloc(sizeof(struct
node));
    new_node->data = data;
    new_node->size = 1;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}


struct node *insert(struct node *root, int data) {
    if (root == NULL) {
        return create_node(data);
    }

    if (root->data > data) {
        root->left = insert(root->left, data);
    } else {
        root->right = insert(root->right, data);
    }

    root->size++;
    return root;
}

struct node *find(struct node *root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }

    if (root->data > data) {
        return find(root->left, data);
    } else {
        return find(root->right, data);
    }
}
```

### Sample Input 1:

```
6
1 10
2 10 1
1 5
1 15
2 5 -1
2 10 0
```

Copy

### Sample Output 1:

```
-1
-1
10
```

Copy

### Explanation:

After the insertions, the sorted array becomes `[10]`, then `[5, 10]`, then `[5, 10, 15]`.

- Query `2 10 1`: `10` is at index `0`, but `k + index(x) = 0 + 1 = 1`, which is out of bounds initially, so the answer is `-1`.
- Query `2 5 -1`: `5` is at index `0`, `k + index(x) = 0 - 1 = -1`, which is out of bounds initially, so the answer is `-1`.
- Query `2 10 0`: `10` is at index `1`, `k + index(x) = 1 + 0 = 1`, so the answer is `10`.

### Sample Input 2:

```
7
1 20
2 20 0
1 10
2 10 1
1 30
2 30 -1
2 20 1
```

Copy

### Sample Output 2:

```
20
20
30
```

### Explanation:

After the insertions, the sorted array becomes `[20]`, then `[10, 20]`, then `[10, 20, 30]`.

- Query `2 20 0`: `20` is at index `0`, `k + index(x) = 0 + 0 = 0`, so the answer is `20`.
- Query `2 10 1`: `10` is at index `0`, `k + index(x) = 0 + 1 = 1`, so the answer is `20`.
- Query `2 30 -1`: `30` is at index `2`, `k + index(x) = 2 - 1 = 1`, so the answer is `20`.
- Query `2 20 1`: `20` is at index `1`, `k + index(x) = 1 + 1 = 2`, so the answer is `30`.

---

### Sample Input 3

Copy

```
6
1 10
1 5
1 15
1 3
2 10 1
2 5 -1
```

### Sample Output 3

Copy

```
15
3
```

---

### Sample Input 4

Copy

```
1 20
1 10
1 30
2 10 2
2 30 -1
```

### Sample Output 4

Copy

```
30
20
```

## Clarifications

Report an issue

No clarifications have been made at this time.