题目描述

有一个特异性的双端队列，该队列可以从头部或尾部添加数据，但是只能从头部移出数据。

小A依次执行2n个指令往队列中添加数据和移出数据。其中n个指令是添加数据（可能从头部添加、也可能从尾部添加），依次添加1到n；n个指令是移出数据。

现在要求移除数据的顺序为1到n。

为了满足最后输出的要求，小A可以在任何时候调整队列中数据的顺序。

请问 小A 最少需要调整几次才能够满足移除数据的顺序正好是1到n；

输入描述

第一行一个数据n，表示数据的范围。

接下来的2n行，其中有n行为添加数据，指令为：

- "head add x" 表示从头部添加数据 x，
- "tail add x" 表示从尾部添加数据x，

另外 n 行为移出数据指令，指令为："remove" 的形式，表示移出1个数据；

1 ≤ n ≤ 3 * 10^5。

所有的数据均合法。

输出描述

一个整数，表示 小A 要调整的最小次数。

用例

输入	5 head add 1 tail add 2 remove head add 3 tail add 4 head add 5 remove remove remove remove
输出	1
说明	无

题目解析

本题重在题目意思理解，本题最后要求：最小的调整顺序次数。而不是最小的交换次数。因此本题的难度大大降低了。

比如用例：

1	head add 1	queue = [1]
2	tail add 2	queue = [1,2]
3	remove	此时删除头部，顺序符合要求，因此不需要调整顺序，删除后，queue=[2]
4	head add 3	queue = [3,2]
5	tail add 4	queue = [3,2,4]
6	head add 5	queue = [5,3,2,4]
7	remove	此时删除头部的元素应该是2，但实际是5，因此需要调整顺序，queue=[2,3,4,5]，然后再删除头部，queue

题目解析

本题重在题目意思理解，本题最后要求：最小的调整顺序次数。而不是最小的交换次数。因此本题的难度大大降低了。

比如用例：

1	head add 1	queue = [1]
2	tail add 2	queue = [1,2]
3	remove	此时删除头部，顺序符合要求，因此不需要调整顺序，删除后，queue=[2]
4	head add 3	queue = [3,2]
5	tail add 4	queue = [3,2,4]
6	head add 5	queue = [5,3,2,4]
7	remove	此时删除头部的元素应该是2，但实际是5，因此需要调整顺序，queue=[2,3,4,5]，然后再删除头部，queue = [3,4,5]
8	remove	此时删除头部3
9	remove	此时删除头部4
10	remove	此时删除头部5

因此，只需要在第7步调整一次顺序。

本题不需要模拟出一个队列，因为那样需要频繁的验证队列元素顺序，以及调整顺序，非常不划算。

我们可以总结规律：

如果队列为空，即size===0，此时无论head add，还是tail add，都不会破坏队列顺序性。

如果队列不为空，即size!==0，此时tail add不会破坏顺序性，head add会破坏顺序性。

我们定义一个变量isSorted表示队列是否有序，初始时isSorted = true，表示初始时队列有序。当有序性被破坏，即isSorted = false。

head add和tail add会导致size++，remove会导致size--。

我们定义一个count变量来记录调整顺序的次数，初始为0。

当remove时，如果isSorted为false，则我们需要调整顺序，即count++，并更新isSorted = true。

当head add时，如果size为0，则不破坏顺序性，isSorted为true，如果size不为0，则会破坏顺序性，即isSorted=false。另外size++。

当tail add时，仅size++。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = parseInt(lines[0]);
  }

  if (n && lines.length === 1 + 2 * n) {
    lines.shift();
    console.log(getResult(lines));
    lines.length = 0;
  }
});

function getResult(cmds) {
  let size = 0;
  let isSorted = true;
  let count = 0;

  for (let i = 0; i < cmds.length; i++) {
    const cmd = cmds[i];
    if (cmd.startsWith("head add")) {
      if (size && isSorted) isSorted = false;
      size++;
    } else if (cmd.startsWith("tail add")) {
      size++;
    } else {
      if (!size) continue;
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = parseInt(lines[0]);
  }

  if (n && lines.length === 1 + 2 * n) {
    lines.shift();
    console.log(getResult(lines));
    lines.length = 0;
  }
});

function getResult(cmds) {
  let size = 0;
  let isSorted = true;
  let count = 0;

  for (let i = 0; i < cmds.length; i++) {
    const cmd = cmds[i];
    if (cmd.startsWith("head add")) {
      if (size && isSorted) isSorted = false;
      size++;
    } else if (cmd.startsWith("tail add")) {
      size++;
    } else {
      if (!size) continue;
      if (!isSorted) {
        count++;
        isSorted = true;
      }
      size--;
    }
  }

  return count;
}
```

Java算法源码

```java
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = Integer.parseInt(sc.nextLine());
    int m = n * 2;

    String[] cmds = new String[m];
    for (int i = 0; i < m; i++) {
      cmds[i] = sc.nextLine();
    }

    System.out.println(getResult(cmds));
  }

  public static int getResult(String[] cmds) {
    int size = 0;
    boolean isSorted = true;
    int count = 0;

    for (int i = 0; i < cmds.length; i++) {
      String cmd = cmds[i];
      if (cmd.startsWith("head add")) {
        if (size > 0 && isSorted) isSorted = false;
        size++;
      } else if (cmd.startsWith("tail add")) {
        size++;
      } else {
        if (size == 0) continue;
        if (!isSorted) {
          count++;
          isSorted = true;
        }
        size--;
      }
    }

    return count;
```

Java算法源码

```java
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = Integer.parseInt(sc.nextLine());
    int m = n * 2;

    String[] cmds = new String[m];
    for (int i = 0; i < m; i++) {
      cmds[i] = sc.nextLine();
    }

    System.out.println(getResult(cmds));
  }

  public static int getResult(String[] cmds) {
    int size = 0;
    boolean isSorted = true;
    int count = 0;

    for (int i = 0; i < cmds.length; i++) {
      String cmd = cmds[i];
      if (cmd.startsWith("head add")) {
        if (size > 0 && isSorted) isSorted = false;
        size++;
      } else if (cmd.startsWith("tail add")) {
        size++;
      } else {
        if (size == 0) continue;
        if (!isSorted) {
          count++;
          isSorted = true;
        }
        size--;
      }
    }

    return count;
  }
}
```

Python算法源码

```python
# 输入获取
n = int(input())
cmds = [input() for i in range(2 * n)]


# 算法入口
def getResult(cmds):
    size = 0
    isSorted = True
    count = 0

    for cmd in cmds:
        if cmd.startswith("head add"):
            if size > 0 and isSorted:
                isSorted = False
            size += 1
        elif cmd.startswith("tail add"):
            size += 1
        else:
            if size <= 0:
                continue

            if not isSorted:
                count += 1
                isSorted = True

            size -= 1

    return count


# 算法调用
print(getResult(cmds))
```