

题目描述：

某部门计划通过结队编程来进行项目开发，已知该部门有 N 名员工，每个员工有独一无二的职级，每三个员工形成一个小组进行结队编程，结队分组规则如下：

从部门中选出序号分别为 i, j, k 的 3 名员工，他们的职级分别为 $level[i]$ 、 $level[j]$ 、 $level[k]$

结队小组需满足： $level[i] < level[j] < level[k]$ 或者 $level[i] > level[j] > level[k]$ ，

其中 $0 \leq i < j < k < n$

请你按上述条件计算可能组合的小组数量。同一员工可以参加多个小组。

输入描述：

第一行输入：员工总数 n

第二行输入：按序号依次排列的员工的职级 $level$ ，中间用空格隔开

限制：

$1 \leq n \leq 6000$

$1 \leq level[i] \leq 10^5$

输出描述：

可能组合的小组数量

示例 1

输入：

4

1 2 3 4

输出：

4

说明：

可能结队成的组合 (1,2,3)、(1,2,4)、(1,3,4)、(2,3,4)

示例 2

输入：

3

5 4 7

输出：

0

说明：

根据结队条件，我们无法为该部门组建小组

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Deque;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Queue;
import java.util.StringJoiner;
import java.util.stream.Collectors;
```

// 注意类名必须为 Main，不要有任何 package xxx 信息

```
public class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer st = new StreamTokenizer(br);
        PrintWriter pw = new PrintWriter(new OutputStreamWriter(System.out));

        st.nextToken();
```

```

int n = (int) st.nval;

int[] levels = new int[n];
for (int i = 0; i < n; i++) {
    st.nextToken();
    levels[i] = (int) st.nval;
}
for (int i = 0; i < levels.length - 2; i++) {
    for (int j = i + 1; j < levels.length - 1; j++) {
        if (levels[i] == levels[j]) {
            continue;
        }
        if (levels[i] > levels[j]) {
            backtrack2(levels, j, 2);
        } else if (levels[i] < levels[j]) {
            backtrack(levels, j, 2);
        }
    }
}
pw.println(ans);
pw.flush();
}

```

```

static int ans = 0;

static void backtrack(int[] levels, int idx, int count) {
    if (count == 3) {
        ans++;
        return;
    }
    for (int i = idx + 1; i < levels.length; i++) {
        if (levels[idx] < levels[i]) {
            backtrack(levels, i, count + 1);
        }
    }
}

```

```

static void backtrack2(int[] levels, int idx, int count) {
    if (count == 3) {
        ans++;
        return;
    }
    for (int i = idx + 1; i < levels.length; i++) {
        if (levels[idx] > levels[i]) {

```

```

        backtrack2(levels, i, count + 1);
    }
}

// public static void main(String[] args) throws IOException {
//     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
//     StreamTokenizer st = new StreamTokenizer(br);
//     PrintWriter pw = new PrintWriter(new OutputStreamWriter(System.out));

//     st.nextToken();
//     int n = (int) st.nval;

//     int[] levels = new int[n];
//     for (int i = 0; i < n; i++) {
//         st.nextToken();
//         levels[i] = (int) st.nval;
//     }
//     for (int i = 0; i < levels.length - 2; i++) {
//         backtrack(levels, i, 1);
//         backtrack2(levels, i, 1);
//     }
//     pw.println(ans);
//     pw.flush();
// }

// static int ans = 0;

// static void backtrack(int[] levels, int idx, int count) {
//     if (count == 3) {
//         ans++;
//         return;
//     }
//     for (int i = idx + 1; i < levels.length; i++) {
//         if (levels[idx] < levels[i]) {
//             backtrack(levels, i, count + 1);
//         }
//     }
// }

// static void backtrack2(int[] levels, int idx, int count) {
//     if (count == 3) {
//         ans++;
//         return;
//     }
// }

```

```
//      for (int i = idx + 1; i < levels.length; i++) {  
//          if (levels[idx] > levels[i]) {  
//              backtrack2(levels, i, count + 1);  
//          }  
//      }  
// }  
// }
```

}