

题目描述

给定一个从小到大的有序整数序列（存在正整数和负整数）数组 `nums`，请你在该数组中找出两个数，其和的绝对值($|nums[x]+nums[y]|$)为最小值，并返回这个绝对值。
每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。

输入描述

一个通过空格分割的有序整数序列字符串，最多1000个整数，且整数数值范围是 -65535~65535。

输出描述

两数之和绝对值最小值

用例

输入	-3 -1 5 7 11 15
输出	2
说明	因为 $ nums[0] + nums[2] = -3 + 5 = 2$ 最小，所以返回 2。

题目解析

这题和 [https://leetcode-cn.com/problems/minimum-absolute-difference-in-array/](#)

几乎一致。只是本题整数序列是升序的。

本题解析就是上面链接算法题的解析。

Java算法源码

暴力破解

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         int[] nums = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
8         System.out.println(getResult(nums));
9     }
10
11     public static String getResult(int[] nums) {
12         int min = Integer.MAX_VALUE;
13         String ans = "";
14
15         for (int i = 0; i < nums.length; i++) {
16             for (int j = i + 1; j < nums.length; j++) {
17                 int sum = Math.abs(nums[i] + nums[j]);
18                 if (min > sum) {
19                     min = sum;
20                     ans = nums[i] + " " + nums[j] + " " + sum;
21                 }
22             }
23         }
24
25         return ans;
26     }
27 }
```

二分查找优化

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         int[] nums = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
8         System.out.println(getResult(nums));
9     }
10
11     public static String getResult(int[] nums) {
12         // Arrays.sort(nums);
13
14         int idx = Arrays.binarySearch(nums, 0);
15         if (idx < 0) idx = -idx - 1;
16
17         // 全正数，或者 0+多个正数
18         if (idx == 0) return nums[0] + " " + nums[1] + " " + (nums[0] + nums[1]);
19
20         int n = nums.length;
21         // 全负数，或者 多个负数+0
22         if (idx >= n - 1) return nums[n - 2] + " " + nums[n - 1] + " " + (nums[n - 2] + nums[n - 1]);
23
24         // 下面是有正有负的处理逻辑
25         int[] min = {Integer.MAX_VALUE};
26         String[] ans = {" "};
27
28         // 负数部分最后两个值
29         if (idx >= 2) {
30             check(min, ans, nums[idx - 2], nums[idx - 1]);
31         }
32
33         // 非负数部分的前两个值
34         if (idx < n - 1) {
35             check(min, ans, nums[idx], nums[idx + 1]);
36         }
37     }
```

```

38 // 非负数部分的数组
39 int[] positive = Arrays.copyOfRange(nums, idx, n);
40 for (int i = 0; i < idx; i++) {
41     // 注意通过二分查找-nums[i]在positive的有序插入位置j，则最接近-nums[i]的值的有两个: j-1和j，其中j位置的元素值 >= -nums[i]
42     // i位置的元素值 < -nums[i]
43     int j = Arrays.binarySearch(positive, -nums[i]);
44
45     if (j < 0) j = -j - 1;
46     if (j == positive.length) j -= 1;
47
48     check(min, ans, nums[i], positive[j]);
49
50     if (j - 1 >= 0) {
51         check(min, ans, nums[i], positive[j - 1]);
52     }
53 }
54
55 return ans[0];
56 }
57
58 public static void check(int[] min, String[] ans, int num1, int num2) {
59     int sum = Math.abs(num1 + num2);
60     if (min[0] > sum) {
61         min[0] = sum;
62         ans[0] = num1 + " " + num2 + " " + sum;
63     }
64 }
65 }

```

JS算法源码

暴力破解

```

1  /* JavaScript Node ACN模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10     const nums = line.split(" ").map(Number);
11     console.log(getResult(nums));
12 });
13
14 function getResult(nums) {
15     let min = Infinity;
16     let ans = "";
17
18     for (let i = 0; i < nums.length; i++) {
19         for (let j = i + 1; j < nums.length; j++) {
20             const sum = Math.abs(nums[i] + nums[j]);
21             if (min > sum) {
22                 min = sum;
23                 ans = `${nums[i]} ${nums[j]} ${sum}`;
24             }
25         }
26     }
27
28     return ans;
29 }

```

二分查找优化

```
1  /* JavaScript Node ACN模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10    const nums = line.split(" ").map(Number);
11    console.log(getResult(nums));
12  });
13
14  function getResult(nums) {
15    // nums.sort((a, b) => a - b);
16
17    let idx = binarySearch(nums, 0);
18    if (idx < 0) idx = -idx - 1;
19
20    // 全正数, 或者 0+多个正数
21    if (idx == 0) return nums[0] + " " + nums[1] + " " + (nums[0] + nums[1]);
22
23    const n = nums.length;
24    // 全负数, 或者 多个负数+0
25    if (idx >= n - 1)
26      return nums[n - 2] + " " + nums[n - 1] + " " + (nums[n - 2] + nums[n - 1]);
27
28    // 下面是有正有负的处理逻辑
29    const min = [Infinity];
30    const ans = [""];
31
```

```

32 // 负数部分最后两个值
33 if (idx >= 2) check(min, ans, nums[idx - 2], nums[idx - 1]);
34
35 // 非负数部分的前两个值
36 if (idx < n - 1) check(min, ans, nums[idx], nums[idx + 1]);
37
38 // 非负数部分的数组
39 const positive = nums.slice(idx);
40 for (let i = 0; i < idx; i++) {
41     // 注意通过二分查找-nums[i]在positive的有序插入位置j，则最接近-nums[i]的值的有两个：j-1和j，其中j位置
42     // 1位置的元素值 < -nums[i]
43     let j = binarySearch(positive, -nums[i]);
44
45     if (j < 0) j = -j - 1;
46     if (j == positive.length) j -= 1;
47
48     check(min, ans, nums[i], positive[j]);
49
50     if (j - 1 >= 0) {
51         check(min, ans, nums[i], positive[j - 1]);
52     }
53 }
54
55 return ans[0];
56 }
57

```

```

58 function check(min, ans, num1, num2) {
59     const sum = Math.abs(num1 + num2);
60     if (min[0] > sum) {
61         min[0] = sum;
62         ans[0] = `${num1} ${num2} ${sum}`;
63     }
64 }
65
66 function binarySearch(arr, target) {
67     let low = 0;
68     let high = arr.length - 1;
69
70     while (low <= high) {
71         const mid = (low + high) >> 1;
72         const midVal = arr[mid];
73
74         if (midVal > target) {
75             high = mid - 1;
76         } else if (midVal < target) {
77             low = mid + 1;
78         } else {
79             return mid;
80         }
81     }
82
83     return -low - 1;
84 }

```

Python算法源码

暴力破解

```
1 # 输入获取
2 import sys
3
4 nums = list(map(int, input().split()))
5
6
7 # 算法入口
8 def getResult():
9     minV = sys.maxsize
10    ans = ""
11
12    for i in range(len(nums)):
13        for j in range(i + 1, len(nums)):
14            sumV = abs(nums[i] + nums[j])
15            if minV > sumV:
16                minV = sumV
17                ans = f"{nums[i]} {nums[j]} {sumV}"
18
19    return ans
20
21
22 # 算法调用
23 print(getResult())
```

二分查找优化

```
1 import sys
2
3 # 输入获取
4 nums = list(map(int, input().split()))
5
6
7 def binarySearch(arr, target):
8     low = 0
9     high = len(arr) - 1
10
11    while low <= high:
12        mid = (low + high) >> 1
13        midVal = arr[mid]
14
15        if midVal > target:
16            high = mid - 1
17        elif midVal < target:
18            low = mid + 1
19        else:
20            return mid
21
22    return -low - 1
23
24
```

```

25 def check(minV, ans, num1, num2):
26     sumV = abs(num1 + num2)
27     if minV[0] > sumV:
28         minV[0] = sumV
29         ans[0] = f"{num1} {num2} {sumV}"
30
31 # 算法入口
32 def getResult():
33     # nums.sort()
34
35     idx = binarySearch(nums, 0)
36     if idx < 0:
37         idx = -idx - 1
38
39     # 全正数, 或者 0+多个正数
40     if idx == 0:
41         return f"{nums[0]} {nums[1]} {nums[0] + nums[1]}"
42
43     n = len(nums)
44     # 全负数, 或者 多个负数+0
45     if idx >= n - 1:
46         return f"{nums[n - 2]} {nums[n - 1]} {nums[n - 2] + nums[n - 1]}"
47
48     # 下面是有正有负的处理逻辑
49     minV = [sys.maxsize]
50     ans = [""]
51
52

```

```

53     # 负数部分最后两个值
54     if idx >= 2:
55         check(minV, ans, nums[idx - 2], nums[idx - 1])
56
57     # 非负数部分的前两个值
58     if idx < n - 1:
59         check(minV, ans, nums[idx], nums[idx + 1])
60
61     # 非负数部分的数组
62     positive = nums[idx:]
63     for i in range(0, idx):
64         # 注意通过二分查找 nums[i] 在 positive 的有序插入位置, 则最接近 -nums[i] 的位置有两个: j-1 和 j, 其中 j 位置的元素值 >= -
65         j = binarySearch(positive, -nums[i])
66
67         if j < 0:
68             j = -j - 1
69
70         if j == len(positive):
71             j -= 1
72
73         check(minV, ans, nums[i], positive[j])
74
75         if j - 1 >= 0:
76             check(minV, ans, nums[i], positive[j - 1])
77
78     return ans[0]
79
80
81 # 算法调用
82 print(getResult())

```