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[redacted]

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。

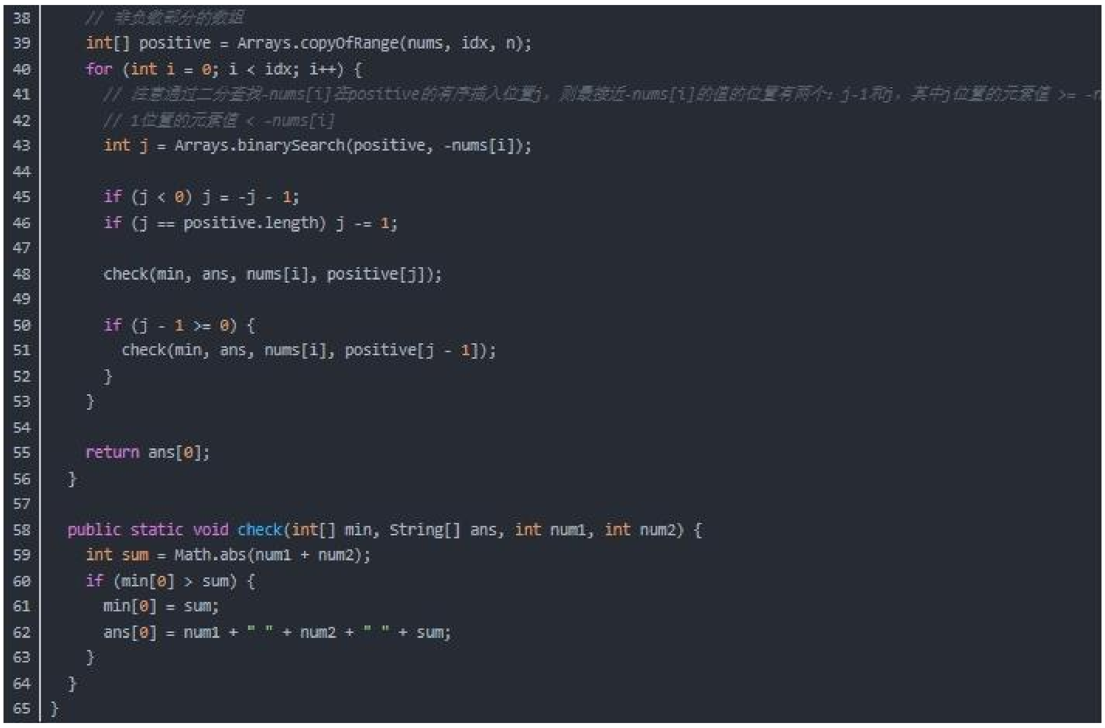
Java算法源码

暴力破解

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int[] nums = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
    System.out.println(getResult(nums));
  }

  public static String getResult(int[] nums) {
    int min = Integer.MAX_VALUE;
    String ans = "";

    for (int i = 0; i < nums.length; i++) {
      for (int j = i + 1; j < nums.length; j++) {
        int sum = Math.abs(nums[i] + nums[j]);
        if (min > sum) {
          min = sum;
          ans = nums[i] + " " + nums[j] + " " + sum;
        }
      }
    }

    return ans;
  }
}
```

二分查找优化

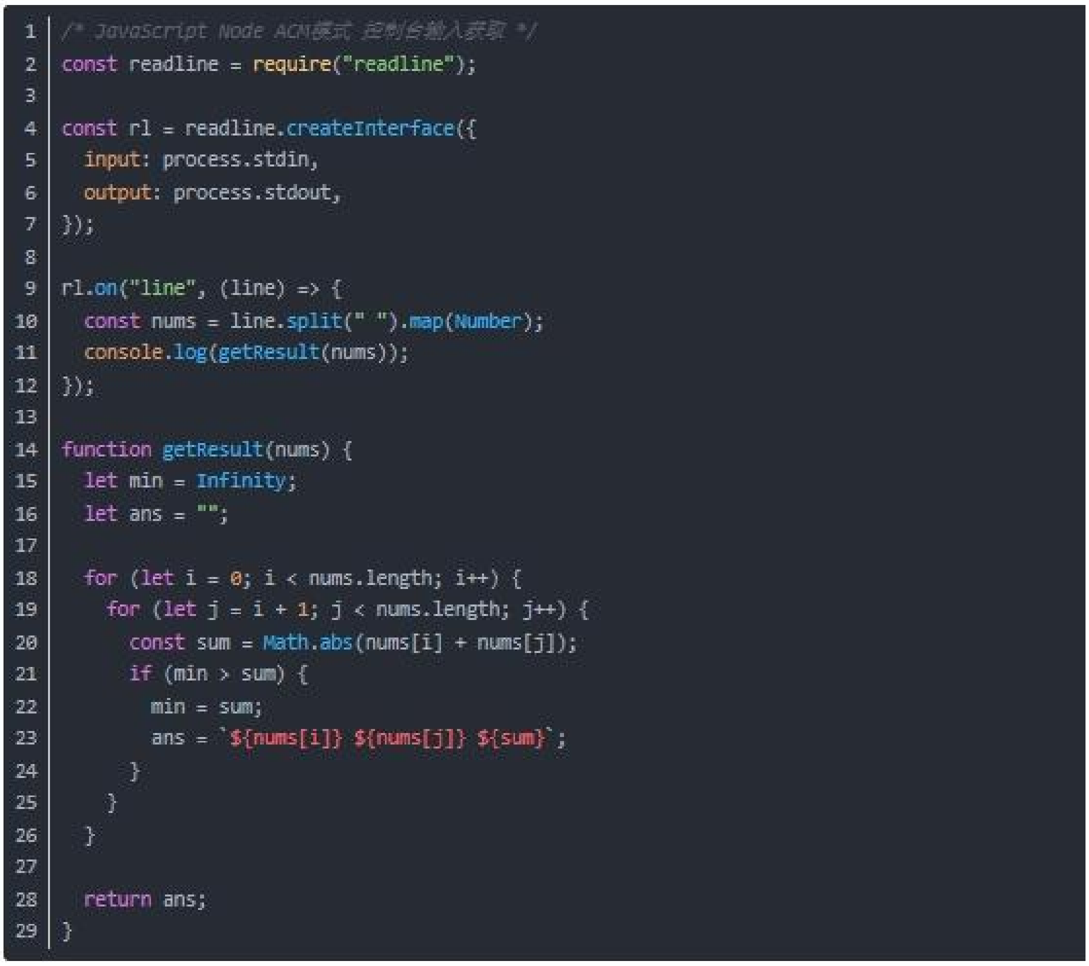
```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int[] nums = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
    System.out.println(getResult(nums));
  }

  public static String getResult(int[] nums) {
    //    Arrays.sort(nums);

    int idx = Arrays.binarySearch(nums, 0);
    if (idx < 0) idx = -idx - 1;

    // 全正数，或者 0+多个正数
    if (idx == 0) return nums[0] + " " + nums[1] + " " + (nums[0] + nums[1]);

    int n = nums.length;
    // 全负数，或者 多个负数+0
    if (idx >= n - 1) return nums[n - 2] + " " + nums[n - 1] + " " + (nums[n - 2] + nums[n - 1]);

    // 下面是有正有负的处理逻辑
    int[] min = {Integer.MAX_VALUE};
    String[] ans = {""};

    // 负数部分最后两个值
    if (idx >= 2) {
      check(min, ans, nums[idx - 2], nums[idx - 1]);
    }

    // 非负数部分的前两个值
    if (idx < n - 1) {
      check(min, ans, nums[idx], nums[idx + 1]);
    }

```



```java
    // 非负数部分的数组
    int[] positive = Arrays.copyOfRange(nums, idx, n);
    for (int i = 0; i < idx; i++) {
      // 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]的值的位置有两个：j-1和j，其中j位置的元素值 >= -n
      // 1位置的元素值 < -nums[i]
      int j = Arrays.binarySearch(positive, -nums[i]);

      if (j < 0) j = -j - 1;
      if (j == positive.length) j -= 1;

      check(min, ans, nums[i], positive[j]);

      if (j - 1 >= 0) {
        check(min, ans, nums[i], positive[j - 1]);
      }
    }

    return ans[0];
  }

  public static void check(int[] min, String[] ans, int num1, int num2) {
    int sum = Math.abs(num1 + num2);
    if (min[0] > sum) {
      min[0] = sum;
      ans[0] = num1 + " " + num2 + " " + sum;
    }
  }
}
```

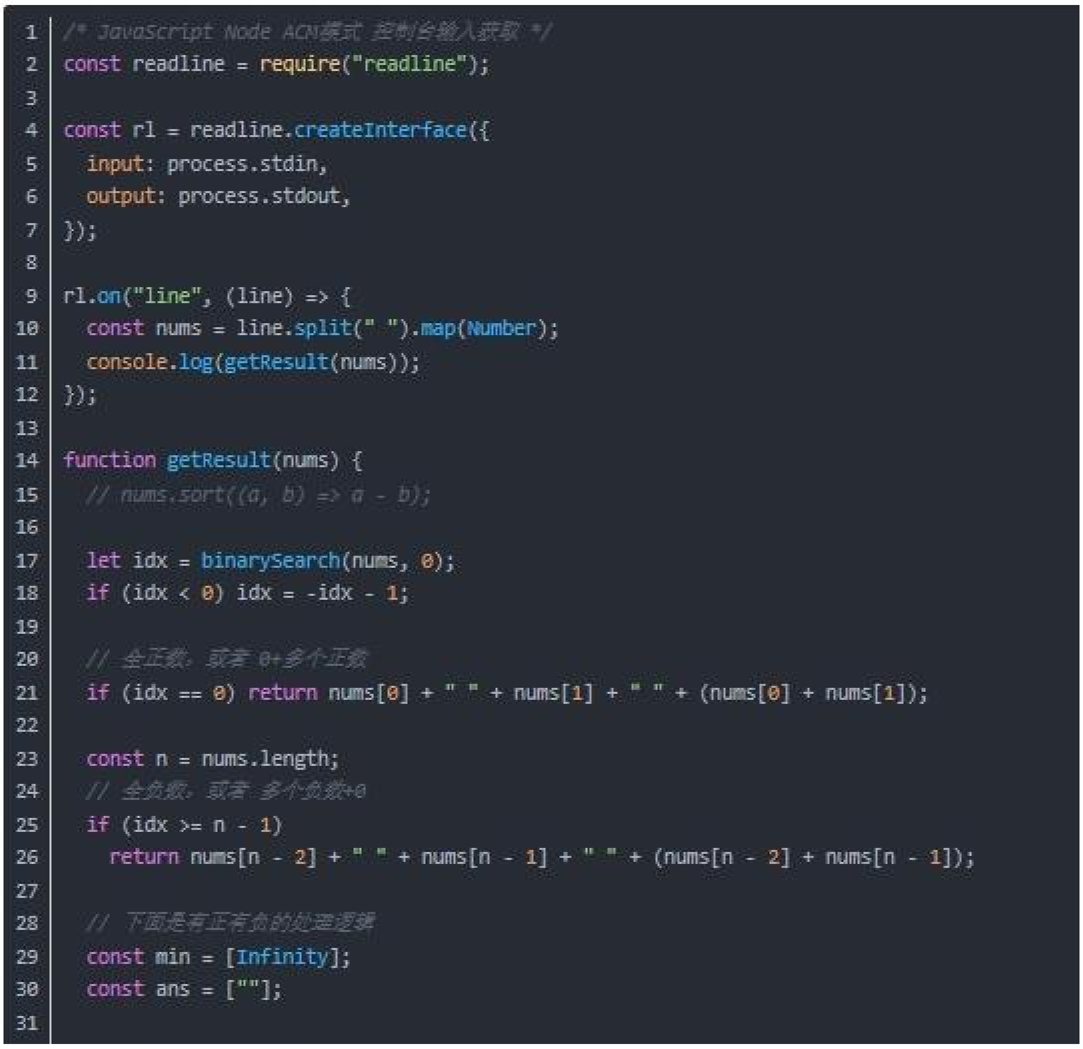
JS算法源码

暴力破解

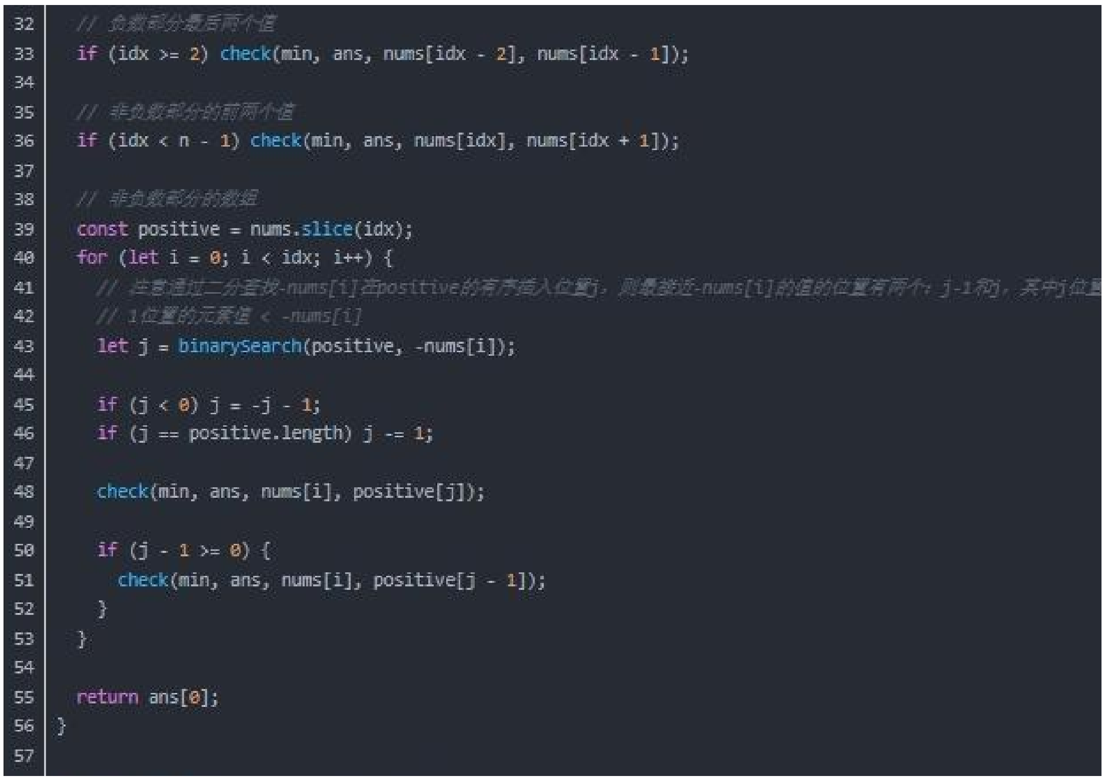


```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

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

rl.on("line", (line) => {
  const nums = line.split(" ").map(Number);
  console.log(getResult(nums));
});

function getResult(nums) {
  let min = Infinity;
  let ans = "";

  for (let i = 0; i < nums.length; i++) {
    for (let j = i + 1; j < nums.length; j++) {
      const sum = Math.abs(nums[i] + nums[j]);
      if (min > sum) {
        min = sum;
        ans = `${nums[i]} ${nums[j]} ${sum}`;
      }
    }
  }

  return ans;
}
```

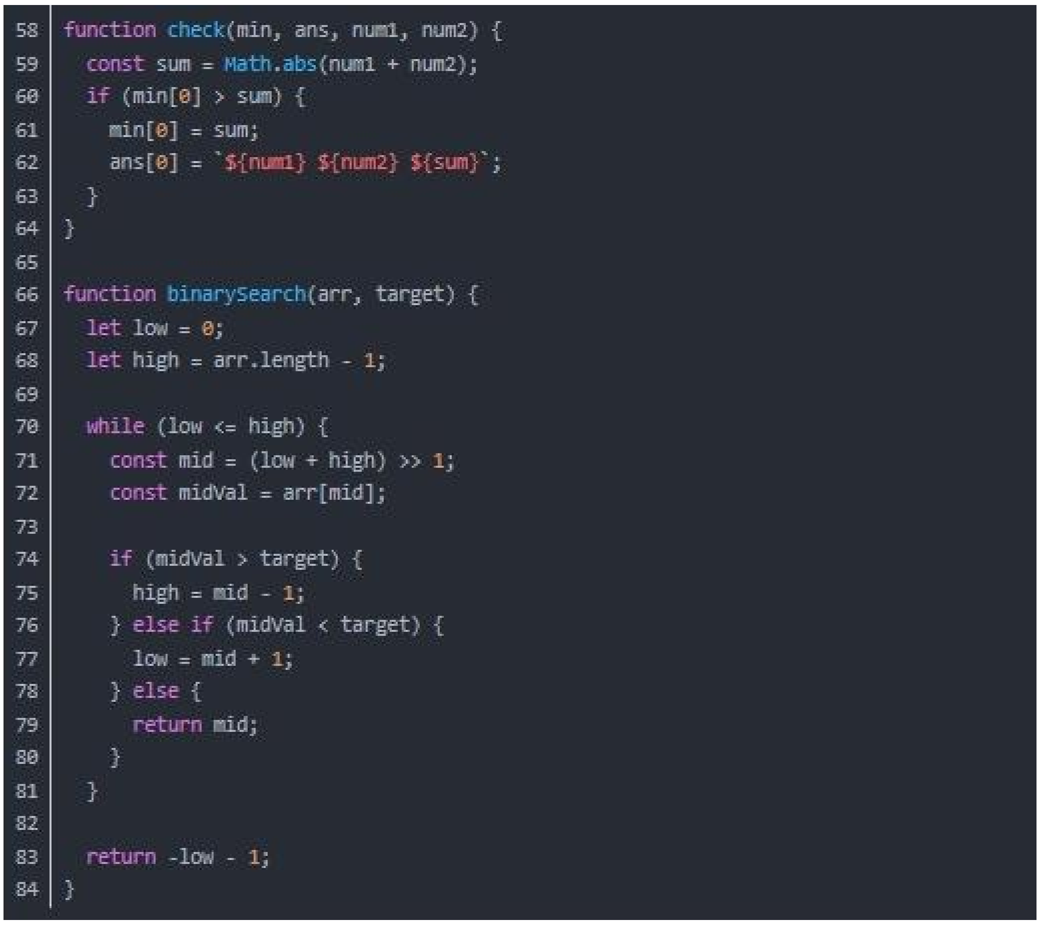
二分查找优化



```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,
});

rl.on("line", (line) => {
  const nums = line.split(" ").map(Number);
  console.log(getResult(nums));
});

function getResult(nums) {
  // nums.sort((a, b) => a - b);

  let idx = binarySearch(nums, 0);
  if (idx < 0) idx = -idx - 1;

  // 全正数，或者 0+多个正数
  if (idx == 0) return nums[0] + " " + nums[1] + " " + (nums[0] + nums[1]);

  const n = nums.length;
  // 全负数，或者 多个负数+0
  if (idx >= n - 1)
    return nums[n - 2] + " " + nums[n - 1] + " " + (nums[n - 2] + nums[n - 1]);

  // 下面是有正有负的处理逻辑
  const min = [Infinity];
  const ans = [""];

```



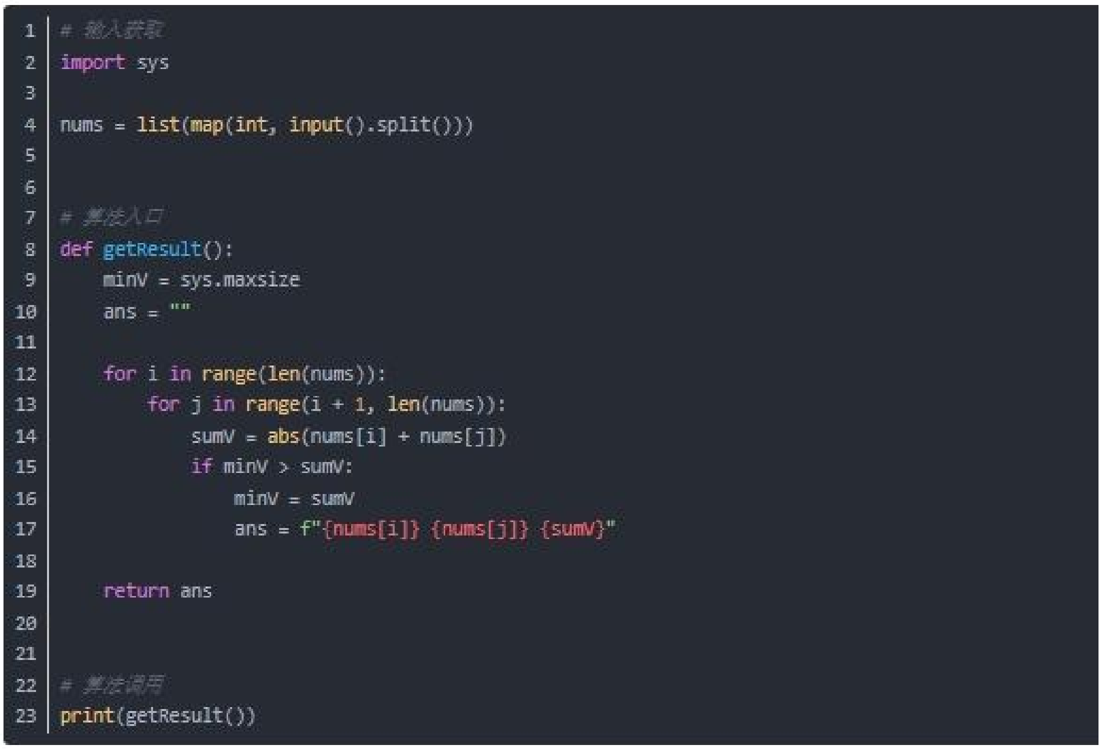
```javascript
  // 负数部分最后两个值
  if (idx >= 2) check(min, ans, nums[idx - 2], nums[idx - 1]);

  // 非负数部分的前两个值
  if (idx < n - 1) check(min, ans, nums[idx], nums[idx + 1]);

  // 非负数部分的数组
  const positive = nums.slice(idx);
  for (let i = 0; i < idx; i++) {
    // 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]的值的位置有两个：j-1和j，其中j位置
    // 1位置的元素值 < -nums[i]
    let j = binarySearch(positive, -nums[i]);

    if (j < 0) j = -j - 1;
    if (j == positive.length) j -= 1;

    check(min, ans, nums[i], positive[j]);

    if (j - 1 >= 0) {
      check(min, ans, nums[i], positive[j - 1]);
    }
  }

  return ans[0];
}

```



```javascript
function check(min, ans, num1, num2) {
  const sum = Math.abs(num1 + num2);
  if (min[0] > sum) {
    min[0] = sum;
    ans[0] = `${num1} ${num2} ${sum}`;
  }
}

function binarySearch(arr, target) {
  let low = 0;
  let high = arr.length - 1;

  while (low <= high) {
    const mid = (low + high) >> 1;
    const midVal = arr[mid];

    if (midVal > target) {
      high = mid - 1;
    } else if (midVal < target) {
      low = mid + 1;
    } else {
      return mid;
    }
  }

  return -low - 1;
}
```

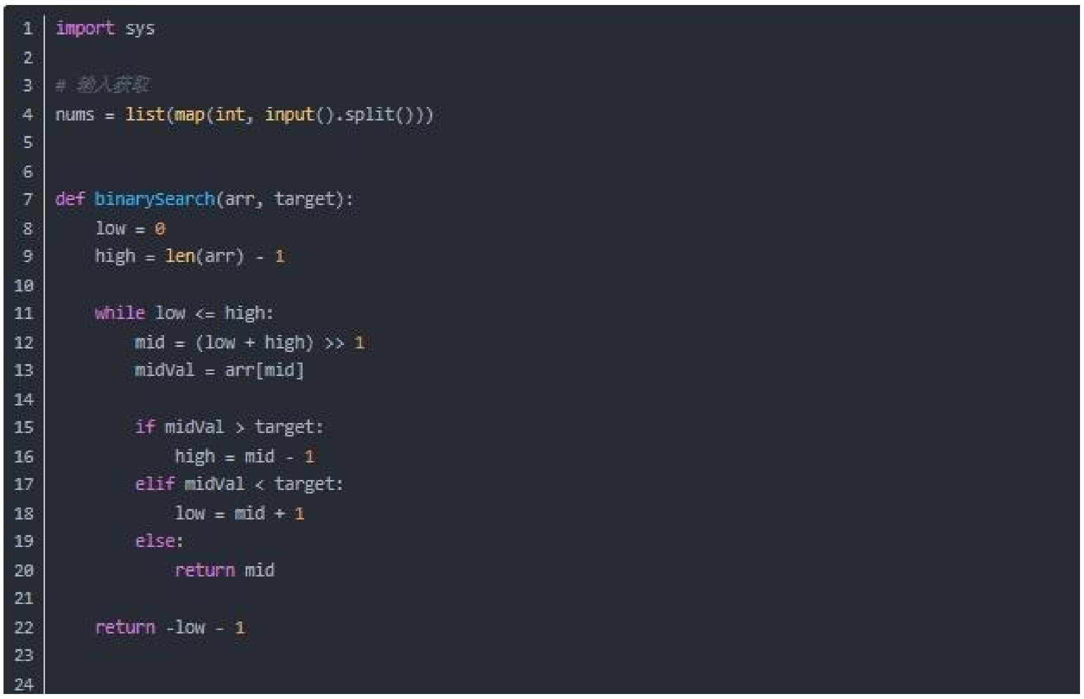
Python算法源码

暴力破解

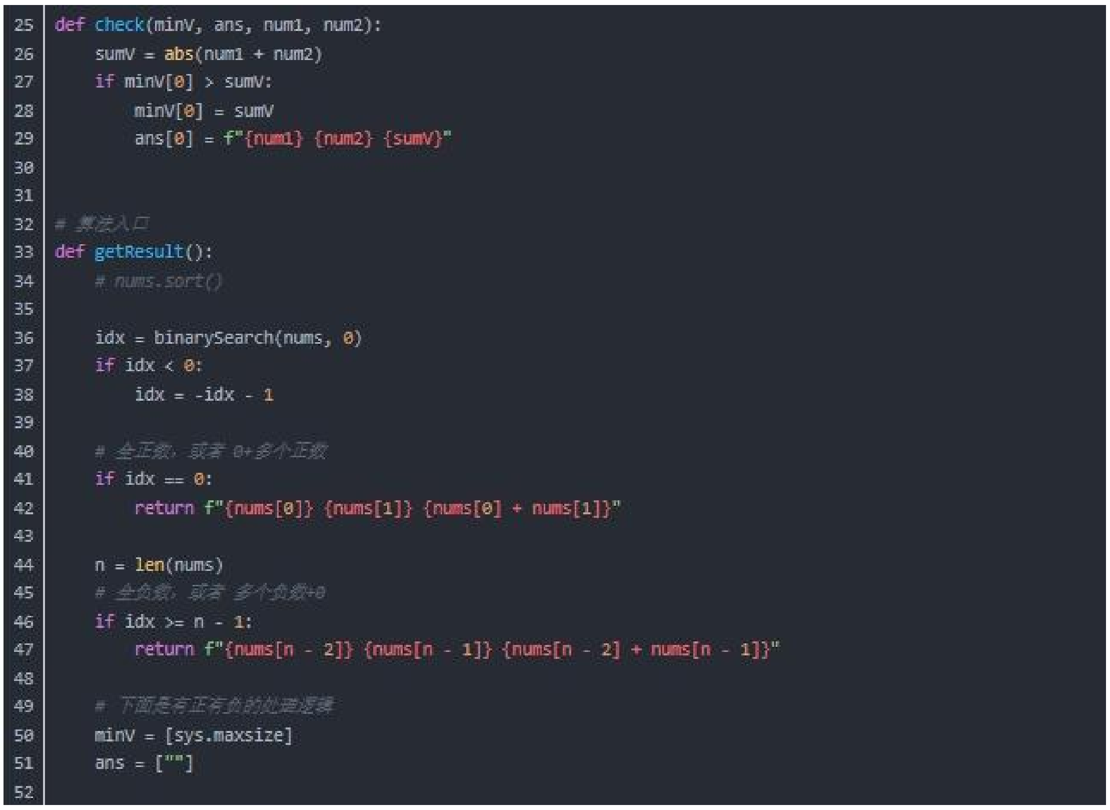


```python
# 输入获取
import sys

nums = list(map(int, input().split()))


# 算法入口
def getResult():
    minV = sys.maxsize
    ans = ""

    for i in range(len(nums)):
        for j in range(i + 1, len(nums)):
            sumV = abs(nums[i] + nums[j])
            if minV > sumV:
                minV = sumV
                ans = f"{nums[i]} {nums[j]} {sumV}"

    return ans


# 算法调用
print(getResult())
```

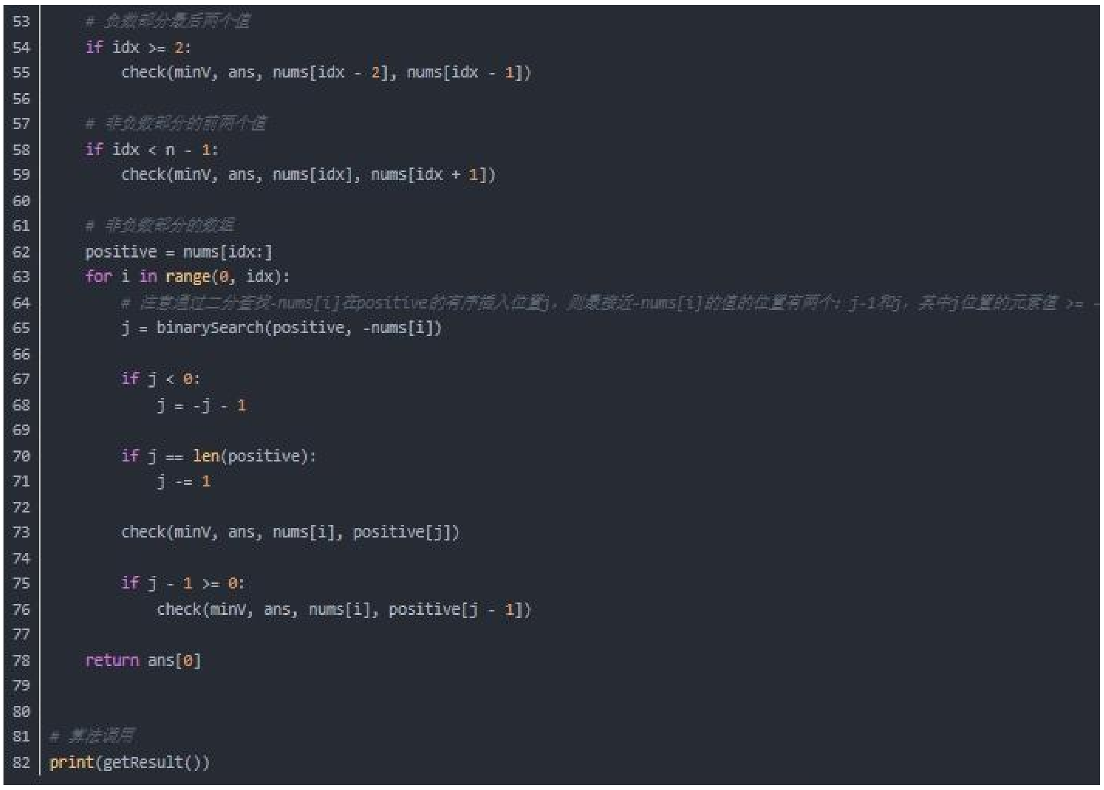
二分查找优化



```python
import sys

# 输入获取
nums = list(map(int, input().split()))


def binarySearch(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) >> 1
        midVal = arr[mid]

        if midVal > target:
            high = mid - 1
        elif midVal < target:
            low = mid + 1
        else:
            return mid

    return -low - 1


```



```python
def check(minV, ans, num1, num2):
    sumV = abs(num1 + num2)
    if minV[0] > sumV:
        minV[0] = sumV
        ans[0] = f"{num1} {num2} {sumV}"


# 算法入口
def getResult():
    # nums.sort()

    idx = binarySearch(nums, 0)
    if idx < 0:
        idx = -idx - 1

    # 全正数，或者 0+多个正数
    if idx == 0:
        return f"{nums[0]} {nums[1]} {nums[0] + nums[1]}"

    n = len(nums)
    # 全负数，或者 多个负数+0
    if idx >= n - 1:
        return f"{nums[n - 2]} {nums[n - 1]} {nums[n - 2] + nums[n - 1]}"

    # 下面是有正有负的处理逻辑
    minV = [sys.maxsize]
    ans = [""]

```



```python
    # 负数部分最后两个值
    if idx >= 2:
        check(minV, ans, nums[idx - 2], nums[idx - 1])

    # 非负数部分的前两个值
    if idx < n - 1:
        check(minV, ans, nums[idx], nums[idx + 1])

    # 非负数部分的数组
    positive = nums[idx:]
    for i in range(0, idx):
        # 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]的值的位置有两个：j-1和j，其中j位置的元素值 >= -
        j = binarySearch(positive, -nums[i])

        if j < 0:
            j = -j - 1

        if j == len(positive):
            j -= 1

        check(minV, ans, nums[i], positive[j])

        if j - 1 >= 0:
            check(minV, ans, nums[i], positive[j - 1])

    return ans[0]


# 算法调用
print(getResult())
```