

实验二

一、设计选择排序算法

1. 任务目标:

- 设计一个函数，它的输入是一个列表，目标是对该列表的元素进行升序排序。

2. 初始化:

- 对于给定的（或用户输入的）列表 `arr`，使用函数获得列表的长度 `n`，以便在遍历和排序过程中使用。

3. 选择排序步骤:

- 外层循环**：遍历列表中的每一个元素。外层循环的索引 `i` 表示当前正在处理的元素位置。
- 内层循环**：在当前元素位置 `i` 之后的子列表中找到最小的元素。内层循环的索引 `j` 用于遍历从 `i+1` 到 `n-1` 的所有元素。
- 选择最小元素**：在内层循环结束时，`min_index` 指向当前子列表中最小元素的位置。
- 交换元素**：将当前外层循环位置 `i` 的元素与内层循环找到的最小元素进行交换。这样，`i` 位置上的元素就是当前未排序部分的最小元素。

4. 重复执行:

- 重复执行上述步骤，直到所有元素都被正确排序。每次外层循环结束时，已排序部分逐渐增大，未排序部分逐渐减小。

5. 返回结果:

- 返回排序后的列表，所有元素已按照升序排列；特别的，若输入为空列表，则返回空列表。

代码实现:

```
def selection_sort(arr):  
    """  
    对给定的列表使用选择排序算法进行升序排序。  
  
    参数:  
    arr (list): 待排序的列表  
  
    返回:  
    list: 排序后的列表  
    """  
    n = len(arr)  
    for i in range(n):  
        min_index = i  
        for j in range(i + 1, n):  
            if arr[j] < arr[min_index]:  
                min_index = j  
        arr[i], arr[min_index] = arr[min_index], arr[i]  
    return arr
```

代码执行结果：

```
问题 输出 调试控制台 终端 窗口
排序后的列表: ['avn', 'ki']
PS C:\Users\19368> & E:/develop/python/python.exe d:/VSC_code/python_code/homework2.py
请输入一组以空格分隔的数据: ko1 acb
排序后的列表: ['acb', 'ko1']
PS C:\Users\19368> & E:/develop/python/python.exe d:/VSC_code/python_code/homework2.py
请输入一组以空格分隔的数据: 1 2 3
排序后的列表: [1.0, 2.0, 3.0]
PS C:\Users\19368> & E:/develop/python/python.exe d:/VSC_code/python_code/homework2.py
请输入一组以空格分隔的数据: 9 5 7 21 56 17 -2 0
排序后的列表: [-2.0, 0.0, 5.0, 7.0, 9.0, 17.0, 21.0, 56.0]
PS C:\Users\19368> & E:/develop/python/python.exe d:/VSC_code/python_code/homework2.py
请输入一组以空格分隔的数据: k l a d m b
排序后的列表: ['a', 'b', 'd', 'k', 'l', 'm']
PS C:\Users\19368>
```

二、确保程序安全性

该程序的理想输入是以空格分割的一串字符（串）或一串数字，当用户混合输入字符与数字或者空输入时，可能导致程序崩溃，故考虑设计一个 `get_valid_input` 函数，实现对输入的检测。

代码实现：

```
def get_valid_input():
    """
    获取用户输入，并将其分为数值和字符型数据，确保输入数据不混合。

    返回：
    tuple: (排序后的数值列表，排序后的字符列表)
    """
    user_input = input("请输入一组以空格分隔的数据：")
    if not user_input.strip():
        # 如果输入为空或仅包含空白字符，返回空列表
        return ([], [])

    parts = user_input.split()
    numbers = []
    strings = []

    for part in parts:
        try:
            number = float(part)
            numbers.append(number)
        except ValueError:
            strings.append(part)

    # 检查是否有混合的情况
    if numbers and strings:
        print("输入数据混合了数值和字符，请分开输入。")
        return ([], [])

    return (selection_sort(numbers), selection_sort(strings))
```

三、设计测试用例

为这个选择排序程序设计测试用例是确保其正确性和鲁棒性的重要步骤。测试用例可以覆盖不同的输入场景，以验证排序算法的正确性。以下是设计思路和十组测试用例：

设计思路

- 测试用例 1 和 2: 检查程序是否能正确排序纯数值或纯字符串输入。
- 测试用例 3: 验证程序是否能够正确处理和返回空输入。
- 测试用例 4 和 5: 测试程序如何处理混合输入，并确保输出为两个空列表。
- 测试用例 6: 测试程序是否能处理并正确排序包含负数的数值。
- 测试用例 7: 确保程序可以处理大小写字母的字符串并正确排序。
- 测试用例 8 和 9: 检查程序是否能处理单一数据项的输入并正确排序（对于单个数值或单个字符串）。
- 测试用例 10: 测试程序在混合数据类型的情况下能否给出正确提示，并返回两个空列表。

十组测试用例

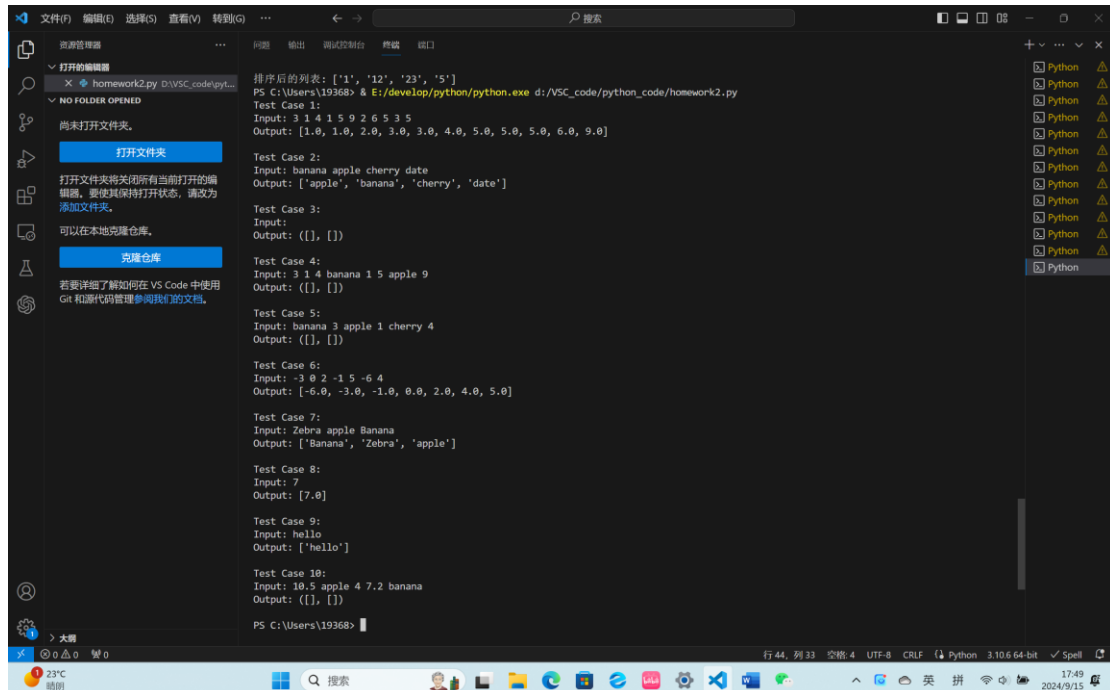
1. 测试用例 1: 纯数值输入
 - 输入: 3 1 4 1 5 9 2 6 5 3 5
 - 预期输出: [1.0, 1.0, 2.0, 3.0, 3.0, 4.0, 5.0, 5.0, 5.0, 6.0, 9.0]
2. 测试用例 2: 纯字符串输入
 - 输入: banana apple cherry date
 - 预期输出: ['apple', 'banana', 'cherry', 'date']
3. 测试用例 3: 空输入
 - 输入: (空字符串)
 - 预期输出: ([], [])
4. 测试用例 4: 混合数值和字符串（数值在前）
 - 输入: 3 1 4 banana 1 5 apple 9
 - 预期输出: ([], []) (提示用户数据混合)
5. 测试用例 5: 混合数值和字符串（字符串在前）
 - 输入: banana 3 apple 1 cherry 4
 - 预期输出: ([], []) (提示用户数据混合)
6. 测试用例 6: 纯数字但包含负数
 - 输入: -3 0 2 -1 5 -6 4
 - 预期输出: [-6.0, -3.0, -1.0, 0.0, 2.0, 4.0, 5.0]
7. 测试用例 7: 纯字符串，包含大小写字母
 - 输入: Zebra apple Banana
 - 预期输出: ['Banana', 'Zebra', 'apple']
8. 测试用例 8: 单个数字输入
 - 输入: 7
 - 预期输出: [7.0]
9. 测试用例 9: 单个字符串输入
 - 输入: hello

- 预期输出: ['hello']
10. 测试用例 10: 混合的不同类型的数值和字符串
- 输入: 10.5 apple 4 7.2 banana
 - 预期输出: ([], []) (提示用户数据混合)

程序设计:

```
test_cases = [  
    "3 1 4 1 5 9 2 6 5 3 5",  
    "banana apple cherry date",  
    "",  
    "3 1 4 banana 1 5 apple 9",  
    "banana 3 apple 1 cherry 4",  
    "-3 0 2 -1 5 -6 4",  
    "Zebra apple Banana",  
    "7",  
    "hello",  
    "10.5 apple 4 7.2 banana"  
]  
  
for i, test_case in enumerate(test_cases, 1):  
    result = process_input(test_case)  
    print(f"Test Case {i}:")  
    print(f"Input: {test_case}")  
    print(f"Output: {result}\n")
```

执行结果:



The screenshot shows a VS Code terminal window with the following output:

```
排序后的列表: ['1', '12', '23', '5']  
PS C:\Users\19368> & E:/develop/python/python.exe d:/VSC_code/python_code/homework2.py  
Test Case 1:  
Input: 3 1 4 1 5 9 2 6 5 3 5  
Output: [1.0, 1.0, 2.0, 3.0, 3.0, 4.0, 5.0, 5.0, 6.0, 9.0]  
  
Test Case 2:  
Input: banana apple cherry date  
Output: ['apple', 'banana', 'cherry', 'date']  
  
Test Case 3:  
Input:  
Output: ([], [])  
  
Test Case 4:  
Input: 3 1 4 banana 1 5 apple 9  
Output: ([], [])  
  
Test Case 5:  
Input: banana 3 apple 1 cherry 4  
Output: ([], [])  
  
Test Case 6:  
Input: -3 0 2 -1 5 -6 4  
Output: [-6.0, -3.0, -1.0, 0.0, 2.0, 4.0, 5.0]  
  
Test Case 7:  
Input: Zebra apple Banana  
Output: ['Banana', 'Zebra', 'apple']  
  
Test Case 8:  
Input: 7  
Output: [7.0]  
  
Test Case 9:  
Input: hello  
Output: ['hello']  
  
Test Case 10:  
Input: 10.5 apple 4 7.2 banana  
Output: ([], [])  
  
PS C:\Users\19368>
```

所有测试结果与预期相符。

四、遇到的困难

1. 同时处理字符与数字型元素:

- 问题: 用户输入的内容可能是数字或字符, 如果将元素全转换为数字进行比较则会发生转换错误导致程序崩溃; 如果全当成字符比较在比较数字时会出现仅按数字的第一位排序的错误(如“15”<“2”)。
- 解决方案: 设计函数检测用户输入的是字符还是数字。

2. 语法差异:

- 问题: 在 C++ 中, 选择排序涉及到复杂的数组操作和指针, 但在 Python 中, 操作更加简洁和直观。转换这些概念有些困难。
- 解决方案: 熟悉 Python 的基本语法和数据结构。了解 Python 中列表(list)的操作方式, 例如如何交换元素和访问列表的索引。

3. 列表操作:

- 问题: 在 Python 中处理列表的方式与 C++ 中的数组有所不同, 比如交换元素的语法和方法。
- 解决方案: 学习 Python 列表的基本操作, 例如如何使用索引访问元素、使用元组进行元素交换(a, b = b, a)等。

4. 范围和迭代:

- 问题: Python 使用 range() 函数生成迭代范围, 而 C++ 使用传统的 for 循环。
- 解决方案: 通过实践学习 range() 的用法, 并了解如何用它生成循环范围。

5. 调试和错误处理:

- 问题: Python 的调试方式与 C++ 不同, 例如 Python 使用异常处理机制来捕捉错误, 而 C++ 使用不同的错误处理方法。
- 解决方案: 学习 Python 的异常处理机制, 使用 try 和 except 语句来捕捉和处理错误, 确保代码的健壮性。

6. 代码结构和风格:

- 问题: Python 强调缩进来定义代码块, 而 C++ 使用大括号。
- 解决方案: 熟悉 Python 的代码风格和最佳实践, 确保代码缩进正确。

五、算法优化

使用选择排序的时间复杂度为 $O(n^2)$, 这是一个相对较高的复杂度, 对于大数据集效率很低。且选择排序不是稳定的排序算法, 相等元素的相对顺序可能会发生变化。

尝试使用归并排序替代选择排序, 归并排序的时间复杂度为 $O(n \log n)$, 这是比较高效的排序时间复杂度, 适用于大多数情况。

设计思路:

1. 递归拆分:

- 将待排序的数组递归地拆分成两个较小的子数组, 直到每个子数组的大小为 1 或 0 (即已经是有序的)。

2. 合并:

- 定义一个合并函数来将两个已排序的子数组合并成一个有序的数组。通过逐一比较两个子数组的元素, 将较小的元素添加到新数组

中。

3. 递归合并:

- 递归地将左右两个子数组排序, 并利用合并函数将它们合并成一个完整的有序数组。

程序设计:

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])

    return merge(left_half, right_half)

def merge(left, right):
    sorted_arr = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_arr.append(left[i])
            i += 1
        else:
            sorted_arr.append(right[j])
            j += 1

    # Append remaining elements
    sorted_arr.extend(left[i:])
    sorted_arr.extend(right[j:])

    return sorted_arr
```

测试用例执行结果:

```
PS C:\Users\19368> & E:/develop/python/python.exe d:/VSC_code/python_code/homework2.py
Test Case 1:
Input: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
Output: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

Test Case 2:
Input: [-3, 0, 2, -1, 5, -6, 4]
Output: [-6, -3, -1, 0, 2, 4, 5]

Test Case 3:
Input: [7]
Output: [7]

Test Case 4:
Input: [10.5, 4, 7.2]
Output: [4, 7.2, 10.5]
```

