The Chinese University of Hong Kong, Shenzhen

CSC 3100

Data Structures

# Programming Assignment 1

*Author:*
Shi Wenlan

*Student Number:*
119010265

June 19, 2021

# Contents

# 1 Findingkth

## 1.1 What are the possible solutions for this problem?

The basic idea to solve this problem is: 1.sort the input data. 2. take out the kth data.

There are many sorting algorithms, just choose one to sort the data. The complexity of sorting algorithms to be discussed is listed below:

| Sorting complexity | Best | Average | Worst |
|---|---|---|---|
| Selection sort | $N^2$ | $N^2$ | $N^2$ |
| Insertion sort | $N$ | $N^2$ | $N^2$ |
| Bubble sort | $N$ | $N^2$ | $N^2$ |
| Merge sort | $N \log N$ | $N \log N$ | $N \log N$ |
| Quicksort | $N \log N$ | $N \log N$ | $N^2$ |

**Figure 1:** Complexity of Various Sorting Algorithms

## 1.2 How do you solve this problem?

I adopted the quick select sort, which is an extended algorithm of quick sort. The algorithm idea is shown in the figure:

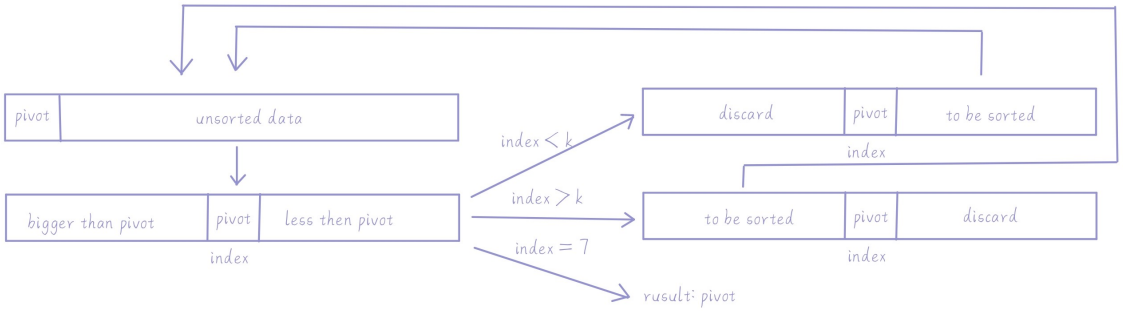


**Figure 2:** Algorithm Idea

## 1.3 Why is your solution better than others?

My goal is to find an algorithm that can sort out the top k largest numbers in the data in the shortest time.

**Insertion sort vs. Merge sort vs. Quick sort:** All these three algorithms need to sort all the data before they can determine what the kth largest data is. According to the speed test data(Figure 3) on the Internet, the quick sort is the fastest. In addition, the quick sort can be optimized for this problem. Everytime after the data is classified to two sides of pivot, the half that does not contain kth position can be discarded, and only the other half need to be recursively sorted, thus reducing the sorting time.


**Figure 3:** Speed Test Data 1

**Selection sort vs. Bubble sort:** Both of these algorithms find the maximum value of unsorted data by traversing the data and put it into sorted data. In this way, we can terminate the algorithm early after finding the kth largest data in the kth loop, thus reducing the sorting time. According to the speed test data(Figure 4) on the Internet, the selection sort is faster. This is because the bubble sort spends a lot of time exchanging elements.

| | Bubble | Selection |
|---|---|---|
| 1000 | 15 | 4 |
| 10000 | 1342 | 412 |
| 100000 | 125212 | 40794 |

**Figure 4:** Speed Test Data 2

**Selection sort(harfway) vs. Quick sort(harfway):**

I wrote the codes of two algorithms respectively, and the results are as Figure 5 and Figure 6. The test results show that the running time of the two algorithms is similar, but because the running time of the former is greatly influenced by k, the source file I submitted is the latter.

| | | | | |
|---|---|---|---|---|
| 测试情况 #1: | AC | 0.022s | 1.62 MB | (10/10) |
| 测试情况 #2: | AC | 0.021s | 1.62 MB | (10/10) |
| 测试情况 #3: | AC | 0.027s | 1.62 MB | (10/10) |
| 测试情况 #4: | AC | 0.021s | 1.62 MB | (10/10) |
| 测试情况 #5: | AC | 0.049s | 3.09 MB | (10/10) |
| 测试情况 #6: | AC | 0.189s | 6.44 MB | (10/10) |
| 测试情况 #7: | AC | 0.205s | 6.44 MB | (10/10) |
| 测试情况 #8: | AC | 0.191s | 6.44 MB | (10/10) |
| 测试情况 #9: | AC | 0.194s | 6.44 MB | (10/10) |
| 测试情况 #10: | AC | 0.355s | 10.31 MB | (10/10) |

**Figure 5:** Result of Selection sort(harfway)

| | | | | |
|---|---|---|---|---|
| 测试情况 #1: | AC | 0.017s | 1.62 MB | (10/10) |
| 测试情况 #2: | AC | 0.018s | 1.62 MB | (10/10) |
| 测试情况 #3: | AC | 0.027s | 1.62 MB | (10/10) |
| 测试情况 #4: | AC | 0.019s | 1.62 MB | (10/10) |
| 测试情况 #5: | AC | 0.053s | 3.09 MB | (10/10) |
| 测试情况 #6: | AC | 0.177s | 6.44 MB | (10/10) |
| 测试情况 #7: | AC | 0.213s | 6.44 MB | (10/10) |
| 测试情况 #8: | AC | 0.202s | 6.44 MB | (10/10) |
| 测试情况 #9: | AC | 0.181s | 6.44 MB | (10/10) |
| 测试情况 #10: | AC | 0.374s | 10.31 MB | (10/10) |

**Figure 6:** Result of Quick Sort(harfway)

### 1.4 How do you test your program?

Submit codes to OJ to see the results.

## 2 MergerSort

### 2.1 What are the possible solutions for this problem?

There are two implementations in the merge section.

**implementation 1:** Compare dequeued elements from the left and the right parts of the original array, store the smaller one in the tempArray. After one part is empty, store all the data in the other part into the tempArray, and then cover the original array with the contents of tempArray.

**implementation 2:** Add an infinite value(such as 9223372036854775807 in C++) to the end of each part in the original array, then store it in the tempArray and clear the original array. After that, compare dequeued elements from the left and the right parts in the tempArray, store the smaller one in the original array.

### 2.2 How do you solve this problem?

I adopted the implementation 2, and the core code is shown in the figure:

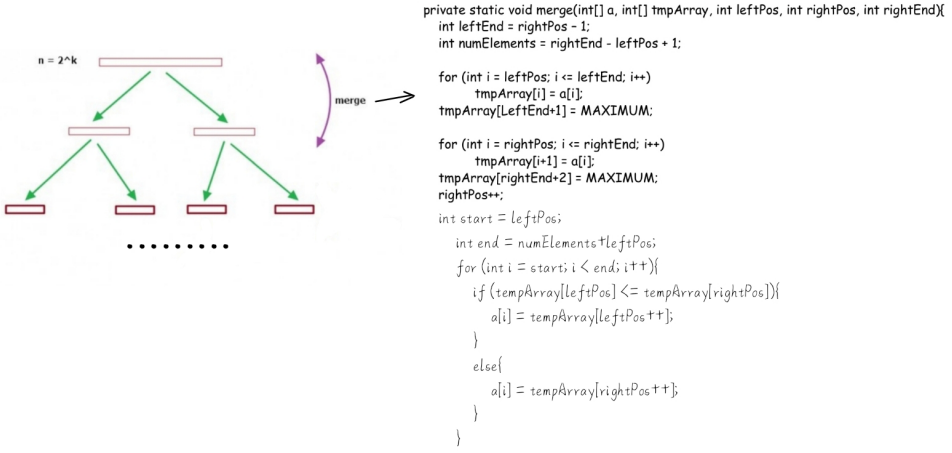


**Figure 7:** Algorithm Idea

## 2.3 Why is your solution better than others?

I wrote the codes of two algorithms respectively, and the results are as Figure 8 and Figure 9. The test results show that the running time of implementation 2 is shorter than that of implementation 1 in most of the cases.

| | | | | |
|---|---|---|---|---|
| 测试情况 #1: | AC | 0.024s | 1.62 MB | (10/10) |
| 测试情况 #2: | AC | 0.024s | 1.62 MB | (10/10) |
| 测试情况 #3: | AC | 0.024s | 1.62 MB | (10/10) |
| 测试情况 #4: | AC | 0.085s | 4.12 MB | (10/10) |
| 测试情况 #5: | AC | 0.068s | 4.12 MB | (10/10) |
| 测试情况 #6: | AC | 0.067s | 4.12 MB | (10/10) |
| 测试情况 #7: | AC | 0.067s | 4.12 MB | (10/10) |
| 测试情况 #8: | AC | 0.121s | 5.67 MB | (10/10) |
| 测试情况 #9: | AC | 0.204s | 8.76 MB | (10/10) |
| 测试情况 #10: | AC | 0.245s | 10.31 MB | (10/10) |

**Figure 8:** Result of Implementation 1

| | | | | |
|---|---|---|---|---|
| 测试情况 #1: | AC | 0.021s | 1.62 MB | (10/10) |
| 测试情况 #2: | AC | 0.021s | 1.62 MB | (10/10) |
| 测试情况 #3: | AC | 0.021s | 1.62 MB | (10/10) |
| 测试情况 #4: | AC | 0.062s | 4.12 MB | (10/10) |
| 测试情况 #5: | AC | 0.062s | 4.12 MB | (10/10) |
| 测试情况 #6: | AC | 0.064s | 4.12 MB | (10/10) |
| 测试情况 #7: | AC | 0.061s | 4.12 MB | (10/10) |
| 测试情况 #8: | AC | 0.123s | 5.67 MB | (10/10) |
| 测试情况 #9: | AC | 0.193s | 8.76 MB | (10/10) |
| 测试情况 #10: | AC | 0.282s | 10.31 MB | (10/10) |

**Figure 9:** Result of Implementation 2

## 2.4 How do you test your program?

Submit codes to OJ to see the results.

# 3 PrefixExp

## 3.1 What are the possible solutions for this problem?

There are two implementations for data input/ouput and storage:

**implementation 1:** Use scanf() to read data input, and use printf() to output data. Use a char array of size 2000000*64 to store data.

**implementation 2:** Use cin to read data input, and use cout to output data. Use a string array of size 2000000 to store data.

## 3.2 How do you solve this problem?

1. Read and store data into string array.
2. Read data from array in reverse order.
3. If the read data is a number, then store it in a long long array. If the read data is an operator, then pop two numbers from the long long array, push $first\_pop$ $operater$ $second_pop$ to the long long array.
4. If there are not enough numbers in the long long array to pop, then output $Invalid$.

5. When all the data in the array are read and processed without error, output the first element of the long long array.
p.s. To pass the test, every input and intermediante calculation result need to mod 1000000007, and theh final result need add 1000000007 firstly and then mod 1000000007.

## 3.3 Why is your solution better than others?

If I choose implementation 1, then I need to store data char by char rather than line by line when use $for$ loop. I also need to write a method to translate char array to long long number. The code might run faster, but it is too complex to write. So I prefer implementation 1 for its code simplicity.

## 3.4 How do you test your program?

Submit codes to OJ to see the results.