PJ1 Report

24302010060 张玳瑗

Running Method：

> MacBook Air Intel Core i5, 16 GB
> macOS Sequoia 15.3.1
> compiled by `clang++ -std=c++17`

Results：

1. for question 1,2,3,5:

For `data.txt` containing `2 4 6 2 5 8 4 9 1 0 4 55 33 77 6 88 99` as the input, following are outputs for sorting methods respectively.

result in InsertSort :

```
zhangdaiyuan@danyuanchangzuibuyuanxing-Mac Data-Structure_class % cd ""/Users/zhang
daiyuan/Code/GitHub/Data-Structure_class/"" && clang++ -std=c++17 "InsertSort.cpp"
-o "InsertSort" && ""/Users/zhangdaiyuan/Code/GitHub/Data-Structure_class/"/InsertS
ort"
0 1 2 2 4 4 4 5 6 6 8 9 33 55 77 88 99
```

result in MergeSort :

```
zhangdaiyuan@danyuanchangzuibuyuanxing-Mac Data-Structure_class % cd ""/Users/zhang
daiyuan/Code/GitHub/Data-Structure_class/"" && clang++ -std=c++17 "MergeSort.cpp" -
o "MergeSort" && ""/Users/zhangdaiyuan/Code/GitHub/Data-Structure_class/"/MergeSort
"
0 1 2 2 4 4 4 5 6 6 8 9 33 55 77 88 99
```

result in QuickSort :

```
zhangdaiyuan@danyuanchangzuibuyuanxing-Mac Data-Structure_class % cd ""/Users/zhang
daiyuan/Code/GitHub/Data-Structure_class/"" && clang++ -std=c++17 "QuickSort.cpp" -
o "QuickSort" && ""/Users/zhangdaiyuan/Code/GitHub/Data-Structure_class/"/QuickSort
"
0 1 2 2 4 4 4 5 6 6 8 9 33 55 77 88 99
```

result in the combination of insert&quickSort :

```
zhangdaiyuan@danyuanchangzuibuyuanxing-Mac Data-Structure_class % cd ""/Users/zhang
daiyuan/Code/GitHub/Data-Structure_class/"" && clang++ -std=c++17 "CombSort.cpp" -o
 "CombSort" && ""/Users/zhangdaiyuan/Code/GitHub/Data-Structure_class/"/CombSort"
0 1 2 2 4 4 4 5 6 6 8 9 33 55 77 88 99
```

All these are correct.

2. for question 4:

k should be picked as the following table:

input scale	theoretical k	practical k
10^6	19.93	100 - 250

10^7	23.25	40 - 60

Process:

1. code for insertion sort

```cpp
template <typename T>
void InsertSort(vector<T>& v){
    T key;
    int i;
    for(int j = 1;j < v.size();j++){
        key = v.at(j);
        i = j-1;
        while(i >= 0 && v.at(i) > key){
            v[i+1] = v[i];
            i--;
        }
        v[i+1] = key;
    }
}
```

Using nested loop, and we can find time complexity equals to $O(n^2)$.

2. code for mergesort

```cpp
template <typename T>
vector<T> Merge(const vector<T> &v1, const vector<T> &v2) {
  vector<T> mer;
  int a = 0, b = 0;
  while (a < v1.size() && b < v2.size()) {
    if (v1[a] < v2[b]) {
      mer.push_back(v1[a++]);
    } else {
      mer.push_back(v2[b++]);
    }
  }
  while (a < v1.size())
    mer.push_back(v1[a++]);
  while (b < v2.size())
    mer.push_back(v2[b++]);
  return mer;
}

template <typename T> vector<T> MergeSort(vector<T> &v) {
  if (v.size() == 1)
    return v;
  int mid = v.size() / 2;
  vector<T> left(v.begin(), v.begin() + mid);
  vector<T> right(v.begin() + mid, v.end());
  return Merge(v1: MergeSort(&v: left), v2: MergeSort(&v: right));
}
```

$f(n) = 2f(n/2) + n$
$T(n) = O(n \log n)$

3. code for quicksort

```cpp
template <typename T> vector<T> Sort(vector<T> &v){
    T key = v.at(0);
    for(int i = 1; i < v.size(); i++){
        if(key > v.at(i)){
            rotate(v.begin(), v.begin() + i, v.begin() + i + 1);
        }
    }
    return v;
}

template <typename T> vector<T> QuickSort(const vector<T> &v) {
  if (v.size() == 1)
    return v;
  if (v.size() == 0)
    return v;
  T key = v[0];
  vector<T> left, right;
  for (int i = 1; i < v.size(); i++) {
    if (v[i] < key)
      left.push_back(v[i]);
    else
      right.push_back(v[i]);
  }
  left = QuickSort(v: left);
  right= QuickSort(v: right);
    left.push_back(key);
  left.insert(left.end(), right.begin(), right.end());
  return left;
}
```

worst case :

$O(n^2)$

average case :

$f(n) = 2f(n/2) + n$

$T(n) = O(n \log n)$

4. code for combsort

```cpp
template <typename T> vector<T> QuickSort(const vector<T> &v,int k) {
  if (v.size() == 1)
    return v;
  if (v.size() == 0)
    return v;
  if(v.size() <= k)
  return InsertSort(v);
  T key = v[0];
  vector<T> left, right;
  for (int i = 1; i < v.size(); i++) {
    if (v[i] < key)
      left.push_back(v[i]);
    else
      right.push_back(v[i]);
  }
  left = QuickSort(v: left,k);
  right= QuickSort(v: right,k);
    left.push_back(key);
  left.insert(left.end(), right.begin(), right.end());
  return left;
}
```

5.how to pick k in combsort

To get k, we need to test the running time with different scales of input.

Firstly, **in theory** we have calculated their time complexity, which are $O(n^2)$ and $O(n \log_2 n)$.In combsort, we have $T(n) = O(nlog(n/k)) + O(nk)$. Make $nk \approx nlog(n/k)$, and we will find that $k = \Theta(logn)$. Facing 10^6 numbers in group 1 , k should approximately be 19.93 , and 23.25 in group 2.

Then, we test it **in practice** with 10^6 and 10^7 random numbers. To compare with the theoretical k, we will test k both start from theoretical results.

- numbers are got from another cpp code given by ChatGPT which produces random numbers and stores them in `data.txt`.
- time is tested by `time ./CombSort`, for results like `2.42s user 0.09s system 77% cpu 3.224 total`, we use `user + system` to respect $T(n)$. And the print process is ignored.
- Every time we repeat three times and then use the average time.

data.txt	k	average time (s)
Group 1	10	1.88
num : 10^6	20	1.60
$exp[k] = 19.93$	100	**1.35**

	150	1.30
	200	1.32
	250	**1.35**
	1000	2.11
Group 2	25	19.94
num : 10^7	37	**16.62**
$exp[k] = 23.25$	50	16.23
	60	**16.99**
	70	18.08
	100	18.67
	1000	28.45

Now to sum up, in case that we sorting 10^6 numbers, k is actually better between 100 - 250 while it should be 19.93 in theory. And k performs faster from 40 - 60 dealing with 10^7 numbers, also differing from 23.25.

Additional Notes:

1.

I having difficulties enabling the sorting of different data type, the code always wants an obvious claim of data type.

```
InsertSort.cpp:36:14: error: no matching function for call to 'ReadFile'
   36 |     auto v = ReadFile(fin);
      |              ^~~~~~~~
```

As a result I have to claim `<int>` here. We can still sort different type by changing our code, though I'm failed in automatically doing it.

```
auto v = ReadFile<int>(fin);
```

2.

In k test I found that user and system time might generally decrease after some time of testing, which brought me trouble deciding the final statics.
Then I've found that by ignoring the process of printing out outcomes, the code obtained 99% cpu, which should make results more accurate than 70% cpu.

3.

When I want to try 10^8 numbers, I found it taking soo long to sort that I have no patience waiting. So I choose

10^7 numbers in group 2.

4.

To figure out why this difference exists between calculation and reality, I asked GPT. And here is what I learn：

- our model is kind of too simplified, for example we regard every step costs the same and our sorting will not be influenced by CPU cache / memory
- as modern CPU has splendid performance, insertion sort can have an advantage for a larger volume than we thought, which makes k bigger
- facing a larger number of figures, quick sort is efficient, which results in different gaps in two experiments