Lab 1

1 Insersion Sort

1. The core code is shown as follows.(Written in C++)
2. Please check "Sort.cpp" for the complete code ,including calculating running time.
3. Please see "generator.py" for the parts of generating random datas, and checking results of the program.
4. Please see "Readme.pdf" to know more about the project,including how to run the program.
5. To prepare for the Combine Sort, this code is able to sort a specific part of the vector.

Algorithm 1: Insertion Sort

```
template <class T>
void Sort<T>::insertionSortPartly(vector<T> &a, int begin, int end)
{
        for (int j = begin + 1; j <= end; j++)
        {
                T temp = a[j];
                int i = j;
                while (i > begin && a[i - 1] >= temp)
                {
                        a[i] = a[i - 1];
                        i--;
                }
                a[i] = temp;
        }
}

template <class T>
void Sort<T>::insertionSort(vector<T> &a)
{
        if (a.empty()) return;
        insertionSortPartly(a, 0, a.size() - 1);
}
```

2 Merge Sort

1. The core code is shown as follows.(Written in C++)

Algorithm 2: Merge Sort

```
template <class T>
void Solution<T>::mergeTwoArrays(vector<T> &a, T *temp, int begin, int end)
```

```
3  {
4          int mid = ((end - begin) / 2) + begin;
5          int begin1 = begin;
6          int begin2 = mid + 1;
7          int t = begin;
8          while (begin1 <= mid && begin2 <= end)
9          {
10                 if (a[begin1] < a[begin2])
11                         temp[t++] = a[begin1++];
12                 else
13                         temp[t++] = a[begin2++];
14         }
15         while (begin1 <= mid)
16                 temp[t++] = a[begin1++];
17         while (begin2 <= end)
18                 temp[t++] = a[begin2++];
19         for (int t = begin; t <= end; t++)
20                 a[t] = temp[t];
21 }
22
23 template <class T>
24 void Solution<T>::mergeSortPartly(vector<T> &a, T *temp, int begin, int end)
25 {
26         int mid = ((end - begin) / 2) + begin;
27         if (begin >= end) return;
28         mergeSortPartly(a, temp, begin, mid);
29         mergeSortPartly(a, temp, mid + 1, end);
30         mergeTwoArrays(a, temp, begin, end);
31 }
32
33 template <class T>
34 void Solution<T>::mergeSort(vector<T> &a)
35 {
36         int begin = 0;
37         int end = a.size() - 1;
38         if (begin >= end) return;
39         int len = end - begin + 1;
40         T *temp = new T[len];
41         mergeSortPartly(a, temp, begin, end);
42 }
```

3 Theory and Practice

Turn to next page to see theoretical analysis.

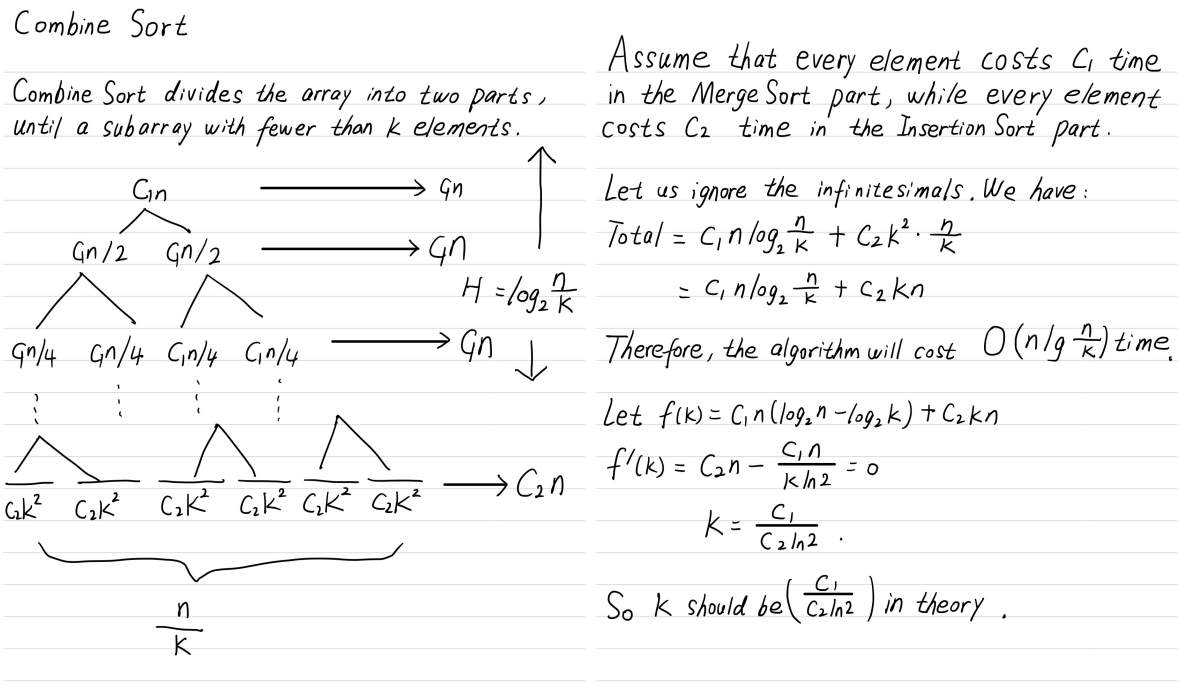

Figure 1: Calculation Process

Turn to next page to see practice by experiments.
Every experiment will repeat 3 times.
The unit is nanoseconds.

N	Insertion1	Insertion2	Insertion3	Merge1	Merge2	Merge3
10	500	500	500	1800	1500	1500
20	1200	1200	1300	2500	2600	2500
50	5600	5200	4700	6200	6100	5800
80	10700	11400	19900	9900	9900	17800
100	18900	20400	17700	12600	12100	12100
1000	1551500	1555100	1564100	155400	154500	153200

Table 1 : Sorting Integers (roughly try)

N	Insertion1	Insertion2	Insertion3	Merge1	Merge2	Merge3
56	6700	6300	6200	6700	6800	6600
58	7100	6800	7100	6900	7100	6900
60	7000	7100	7400	7200	6700	7300
62	9600	7000	6400	7400	7500	7100
64	7700	7700	7800	7600	7600	7700

Table 2 : Sorting Integers (precisely find)

N	Insertion1	Insertion2	Insertion3	Merge1	Merge2	Merge3
56	5500	6700	5300	6600	6700	6800
58	7400	6300	6900	7200	7200	7100
60	7600	7500	7600	8000	7600	7700
62	7900	8500	7600	7500	7900	7500
64	8200	8200	7400	8100	7400	8200

Table 3 : Sorting Floats (precisely find)

N	Insertion1	Insertion2	Insertion3	Merge1	Merge2	Merge3
28	18900	20800	19500	20000	20700	20700
30	21000	21500	21400	22100	23200	22400
32	24300	24100	21800	23700	24000	23500
34	26400	31600	32400	26100	26000	26300
36	24900	29700	30900	26700	28500	27000

Table 4 : Sorting Strings (precisely find)

So the k in practice of experiments of sorting integers or floats is about 60. But the k in practice of experiments of sorting strings is about 32.

4 Combine Sort

1. The core code is shown as follows.(Written in C++)
2. In the code K is a const int number. It can be defined by users to fit their own machines.

Algorithm 3: Combine Sort

```
template <class T>
void Sort<T>::mergeTwoArrays(vector<T> &a, T *temp, int begin, int end)
{
        int mid = ((end - begin) / 2) + begin;
        int begin1 = begin;
        int begin2 = mid + 1;
        int t = begin;
        while (begin1 <= mid && begin2 <= end)
        {
                if (a[begin1] < a[begin2])
                        temp[t++] = a[begin1++];
                else
                        temp[t++] = a[begin2++];
        }
        while (begin1 <= mid)
                temp[t++] = a[begin1++];
        while (begin2 <= end)
                temp[t++] = a[begin2++];
        for (int t = begin; t <= end; t++)
                a[t] = temp[t];
}

template <class T>
void Sort<T>::combineSortPartly(vector<T> &a, T *temp, int begin, int end)
{
        if (end - begin < K)
        {
                insertionSortPartly(a, begin, end);
        }
        else
        {
                int mid = ((end - begin) / 2) + begin;
                combineSortPartly(a, temp, begin, mid);
                combineSortPartly(a, temp, mid + 1, end);
                mergeTwoArrays(a, temp, begin, end);
        }
}

template <class T>
void Sort<T>::combineSort(vector<T> &a)
{
        int begin = 0;
```

```
43          int end = a.size() - 1;
44          if (begin >= end)
45          return;
46          int len = end - begin + 1;
47          T *temp = new T[len];
48          combineSortPartly(a, temp, begin, end);
49  }
```