1. Title and Author

Assignment 2

110704008 羅瑋翔

xianglo9121.mg10@nycu.edu.tw

2. Statement of The Problem

In this assignment, we are discussing about the efficiency of different sorting algorithms. (1) Insertion sort (2) Selection sort (3) Quick sort (4) Merge sort (5) Heap sort, respectively. And the difference algorithms will sort n elements, where the range of n is from 10000 to 100000. To calculate the CPU time that each sorting algorithm cost, I include the head file <ctime> and create the variable with the data type **clock_t.** Then print the result on a txt file and save it.

3. Main Results.

(a) Description of the design of My Program

(In the main function)

First create the not ordered sequence randomly. And I let different algorithms sort the same sequence, this way, we can prevent different algorithms run different cases. Such as, some runs the better cases, some runs the worse cases.

Then the following part used to calculate the CPU time that the algorithm used while sorting.

Then the last part is the debug mode. I print the array after sorting to check if the algorithm is right or not.

(functions)

The whole code is in the part (3 –c). Here, I only introduce what are the function used for, and the functions are called while the CPU time calculating.

- iSort() : Pass the array, and the size of the array to this function. Then use Insertion sort to sort it.
- sSort() : Pass the array, and the size of the array to this function. Then, use selection sort to sort it.
- qSort() : Sent the array, the index of left, and the index of right to this function. Then use the method, recursion, to call this function until the stopping case.
- mSort() : Sent the array, the index of left, and the index of right to this function. (Divide part) Continuously call the function itself until facing with the stopping case.
- merge() : (Conquer part) This part deal with the sub ordered

subarray. Enables the subarray to comparing value and sent the ordered values to the array which is needed to be sorted.

- hSort() : Sent the array and its size to this function, and call the heapify function.
- heapify() : Sent the array, its size, and the index of the root to this function. To change nodes, enabling the tree to become max heap.

(b) Array is the data structure I used in this program. This data structure is used to save the sequence which is randomly generated. I use array because it helps me easily save the sequence one-by-one in a row. And do the same thing only changing the index of the array. If not using it, I might create a lot of variables, and the code would be very long.

(c) The whole program is below.

```
#include<iostream>
#include<iomanip>
#include<ctime>
#include<cstring>
#include<cstdlib>
#include<fstream>
using namespace std;

int data1[100000], data2[100000], data3[100000], data4[100000],
data5[100000];
```

```
void iSort(int data[], int n);

void sSort(int data[], int n);

void qSort(int data[], int left, int right);

void mSort(int data[], int l, int r);

void merge(int data[], int l, int m, int r);

void hSort(int data[], int n);

void heapify(int data[], int n, int i);

int main(){
    ifstream inFile;
    ofstream outFile;
    //cout the result in result.txt file. It can help me easily and
clearly compare the result of each algorithms
    outFile.open("result1.txt");
    const int n1 = 10000, n2 = 100000;
    srand(time(NULL));
    outFile << setw(10) << "n" <<  setw(20) << "Insertion Sort" <<
setw(20) << "Selction Sort" << setw(15)
        << "Quick Sort"<< setw(15) << "Merge Sort" << setw(15) <<
"Heap Sort" << endl;

    //the following step calculate the CPU time that each sorting
methods cost
    for(int i = n1; i <= n2; i += 1000){
        double iTime = 0, sTime = 0, qTime = 0, mTime = 0, hTime = 0;
        int used[i + 10] = {0};
        int j;
        for(j = 0; j < i; j++){
            //since the random number(the compiler offered) is range
from 0 ~ 32767 so I do the methods below to deal with it.
            int temp = ((rand() << 15) + rand()) % i + 1;
```

```
            //the used array record the numbers which had been
generated.
            if(used[temp] == 0){
                data1[j] = temp;
                data2[j] = temp;
                data3[j] = temp;
                data4[j] = temp;
                data5[j] = temp;
                used[temp] = 1;
            }
            else{
                j--;
            }
        }
        //this part only calculate the time used in sorting
        clock_t clicks1 = clock();
        iSort(data1, i);
        clicks1 = clock() - clicks1;
        iTime += (double)clicks1 / CLOCKS_PER_SEC;

        clock_t clicks2 = clock();
        sSort(data2, i);
        clicks2 = clock() - clicks2;
        sTime += (double)clicks2 / CLOCKS_PER_SEC;

        clock_t clicks3 = clock();
        qSort(data3, 0, i - 1);
        clicks3 = clock() - clicks3;
        qTime += (double)clicks3 / CLOCKS_PER_SEC;

        clock_t clicks4 = clock();
        mSort(data4, 0, i - 1);
        clicks4 = clock() - clicks4;
        mTime += (double)clicks4 / CLOCKS_PER_SEC;

        clock_t clicks5 = clock();
        hSort(data5, i);
        clicks5 = clock() - clicks5;
```

```
        hTime += (double)clicks5 / CLOCKS_PER_SEC;
        outFile << setw(10) << i << setw(20) << iTime << setw(20) <<
sTime << setw(15)
            << qTime << setw(15) << mTime << setw(15) << hTime <<
endl;
        //debug mode.
        for(int j = 0; j < i; j++){
            cout << setw(10) << data1[j] << setw(10) << data2[j] <<
setw(10) << data3[j] << setw(10)
                << data4[j] << setw(10) << data5[j] << endl;
        }
    }
    outFile.close();

    system("PAUSE");
    return 0;
}

void iSort(int data1[], int n){
    for(int i = 1; i < n; i++){
        int y = data1[i];
        int j = i - 1;
        while(y < data1[j] && j >= 0){
            data1[j + 1] = data1[j];
            j--;
        }
        data1[j + 1] = y;
    }
}

void sSort(int data2[], int n){
    for(int i = 0; i < n; i++){
        for(int j = i + 1; j < n; j++){
            if(data2[i] > data2[j])
            {
                int tmp = data2[i];
                data2[i] = data2[j];
                data2[j] = tmp;
```

```
            }
        }
    }
}

void qSort(int data3[], int left, int right){
    int i = left, j = right + 1, tmp;
    if(left < right){
        while(i < j){
            while(data3[++i] < data3[left]);
            while(data3[--j] > data3[left]);
            if(i >= j)
                break;
            tmp = data3[i];
            data3[i] = data3[j];
            data3[j] = tmp;
        }
        tmp = data3[j];
        data3[j] = data3[left];
        data3[left] = tmp;
        qSort(data3, left, j - 1);
        qSort(data3, j + 1, right);
    }
}

void merge(int data[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = data[l + i];
    for (j = 0; j < n2; j++)
        R[j] = data[m + 1 + j];
    i = 0;
    j = 0;
    k = l;
```

```
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            data[k] = L[i];
            i++;
        }
        else {
            data[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        data[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        data[k] = R[j];
        j++;
        k++;
    }
}

void mSort(int data[], int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;
        //the devide part
        mSort(data, l, m);
        mSort(data, m + 1, r);
        //the conquer part
        merge(data, l, m, r);
    }
}

void hSort(int data5[], int n){
    for(int i = n / 2 - 1; i >= 0; i--)
        heapify(data5, n, i);
```

```
    for(int i = n - 1; i > 0; i--){
        int temp = data5[0];
        data5[0] = data5[i];
        data5[i] = temp;
        heapify(data5, i, 0);
    }
}

void heapify(int data5[], int n, int i){
    //initialize largest as root
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if(l < n && data5[l] > data5[largest])
        largest = l;
    if(r < n && data5[r] > data5[largest])
        largest = r;
    if(largest != i){
        int temp = data5[i];
        data5[i] = data5[largest];
        data5[largest] = temp;
        heapify(data5, n, largest);
    }
}
```

(d) Outputs of the compilation and the executions

result.txt

檔案 編輯 檢視

n	Insertion Sort	Selction Sort	Quick Sort	Merge Sort	Heap Sort
10000	0.057	0.205	0.001	0.001	0.002
11000	0.074	0.258	0.001	0.002	0.002
12000	0.093	0.307	0.001	0.002	0.002
13000	0.102	0.351	0.001	0.002	0.002
14000	0.112	0.429	0.001	0.002	0.003
15000	0.132	0.483	0.002	0.002	0.002
16000	0.14	0.551	0.001	0.003	0.002
17000	0.167	0.597	0.003	0.002	0.004
18000	0.177	0.678	0.002	0.003	0.003
19000	0.201	0.771	0.002	0.003	0.004
20000	0.221	0.88	0.003	0.003	0.004
21000	0.273	0.982	0.002	0.004	0.003
22000	0.273	1.063	0.002	0.003	0.003
23000	0.287	1.144	0.003	0.003	0.003
24000	0.321	1.231	0.003	0.003	0.005
25000	0.366	1.358	0.002	0.004	0.003
26000	0.367	1.484	0.002	0.004	0.005
27000	0.4	1.6	0.003	0.004	0.004
28000	0.484	1.723	0.003	0.003	0.005
29000	0.456	1.826	0.004	0.004	0.005
30000	0.498	1.92	0.002	0.004	0.006
31000	0.549	2.099	0.003	0.005	0.006
32000	0.572	2.249	0.004	0.004	0.005
33000	0.597	2.355	0.003	0.004	0.006
34000	0.617	2.545	0.003	0.005	0.006
35000	0.693	2.611	0.004	0.004	0.006
36000	0.677	2.732	0.004	0.005	0.006
37000	0.727	2.93	0.003	0.005	0.006
38000	0.759	3.075	0.004	0.005	0.005
39000	0.798	3.208	0.004	0.005	0.007
40000	0.861	3.442	0.004	0.006	0.007
41000	0.894	3.594	0.005	0.005	0.008
42000	0.933	3.758	0.004	0.005	0.007
43000	0.993	4.075	0.006	0.006	0.008
44000	1.039	4.219	0.005	0.006	0.008
45000	1.054	4.326	0.005	0.006	0.008
46000	1.129	4.564	0.005	0.006	0.008
47000	1.174	4.744	0.005	0.007	0.007
48000	1.204	4.997	0.005	0.006	0.009

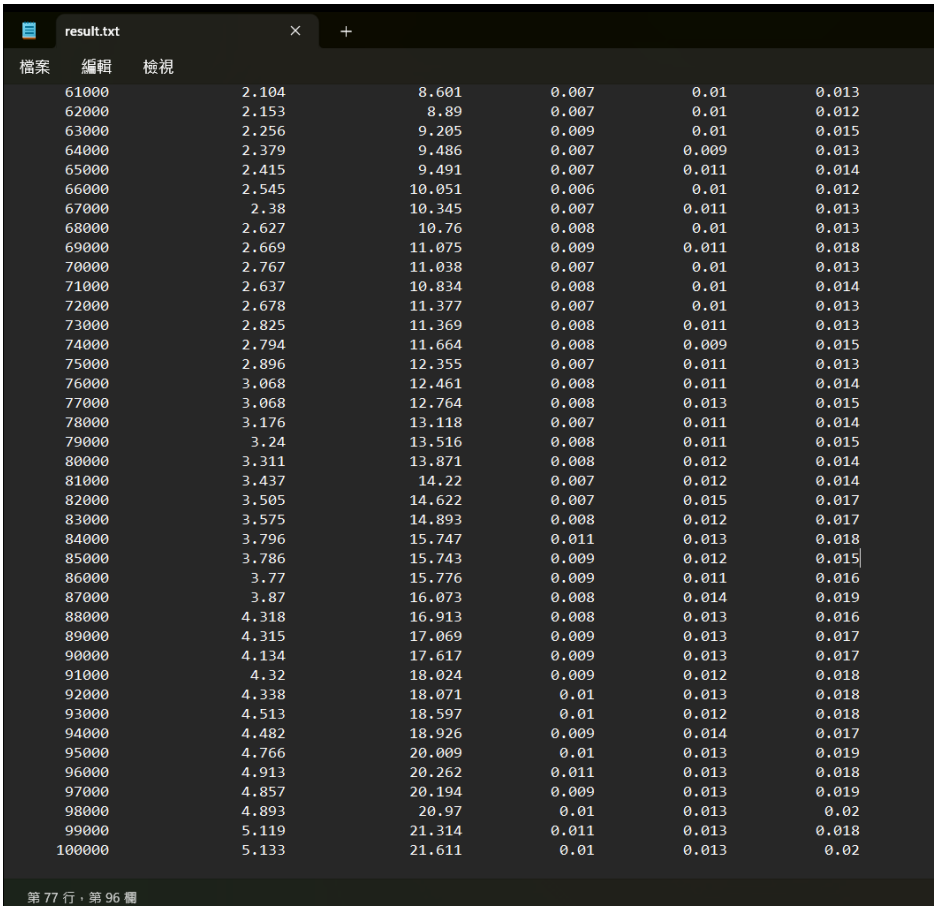
第 84 行，第 96 欄

result.txt

檔案 編輯 檢視

48000	1.204	4.997	0.005	0.006	0.009
49000	1.284	5.126	0.005	0.007	0.009
50000	1.32	5.364	0.005	0.007	0.009
51000	1.351	5.592	0.005	0.007	0.009
52000	1.46	5.77	0.005	0.007	0.01
53000	1.507	5.988	0.006	0.007	0.009
54000	1.523	6.158	0.006	0.007	0.009
55000	1.574	6.47	0.005	0.008	0.009
56000	1.647	6.646	0.006	0.007	0.012
57000	1.716	6.962	0.006	0.007	0.011
58000	1.76	7.421	0.007	0.01	0.011
59000	1.955	8.102	0.008	0.009	0.014
60000	1.989	8.215	0.007	0.008	0.012
61000	2.104	8.601	0.007	0.01	0.013
62000	2.153	8.89	0.007	0.01	0.012
63000	2.256	9.205	0.009	0.01	0.015
64000	2.379	9.486	0.007	0.009	0.013
65000	2.415	9.491	0.007	0.011	0.014
66000	2.545	10.051	0.006	0.01	0.012
67000	2.38	10.345	0.007	0.011	0.013
68000	2.627	10.76	0.008	0.01	0.013
69000	2.669	11.075	0.009	0.011	0.018
70000	2.767	11.038	0.007	0.01	0.013
71000	2.637	10.834	0.008	0.01	0.014
72000	2.678	11.377	0.007	0.01	0.013
73000	2.825	11.369	0.008	0.011	0.013
74000	2.794	11.664	0.008	0.009	0.015
75000	2.896	12.355	0.007	0.011	0.013
76000	3.068	12.461	0.008	0.011	0.014
77000	3.068	12.764	0.008	0.013	0.015
78000	3.176	13.118	0.007	0.011	0.014
79000	3.24	13.516	0.008	0.011	0.015
80000	3.311	13.871	0.008	0.012	0.014
81000	3.437	14.22	0.007	0.012	0.014
82000	3.505	14.622	0.007	0.015	0.017
83000	3.575	14.893	0.008	0.012	0.017
84000	3.796	15.747	0.011	0.013	0.018
85000	3.786	15.743	0.009	0.012	0.015
86000	3.77	15.776	0.009	0.011	0.016
87000	3.87	16.073	0.008	0.014	0.019
88000	4.318	16.913	0.008	0.013	0.016

第 85 行，第 96 欄



(4) Conclusions.