Lab03 Sorting and Selection

VE281 - Data Structures and Algorithms, Xiaofeng Gao, Autumn 2019

Name:Wu Jiayao Student ID:517370910257 Email: jiayaowu1999@sjtu.edu.cn

1 Comparison between five sorting algorithm

The running time of bubble sort is much more longer than other sorting algorithm, followed by selection sort, insert sort, merge sort and quick sort. The difference becomes much more obvious and significant when the size of the array grows.

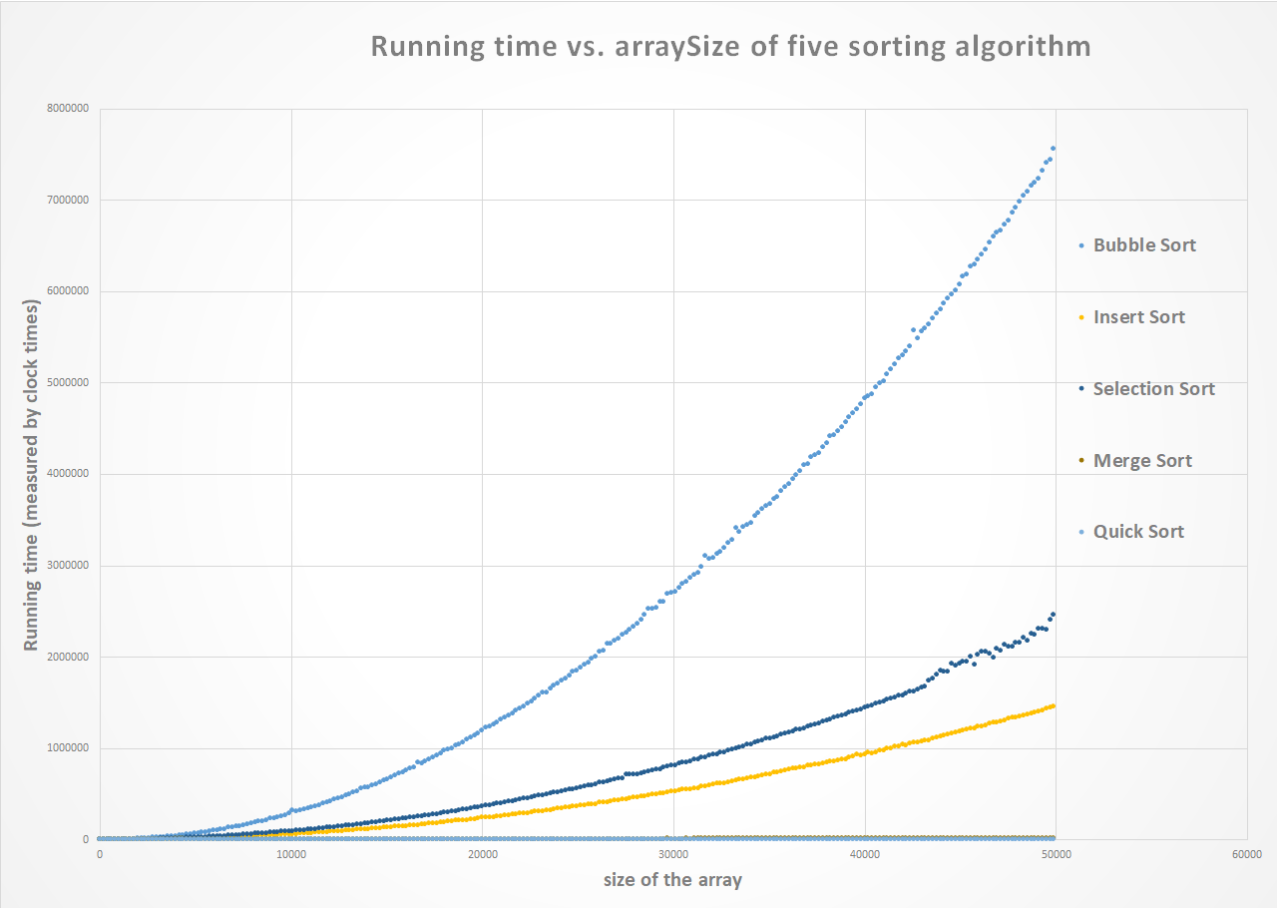

Figure 1: Running time vs. size of array for five sorting algorithms

Since the running time of merge sort and quick sort are too closely plotted to be recognized, here is a plot which only contains running time of merge sort and quick sort. It can be seen in the figure that quick sort tends to perform better than merge sort overall.

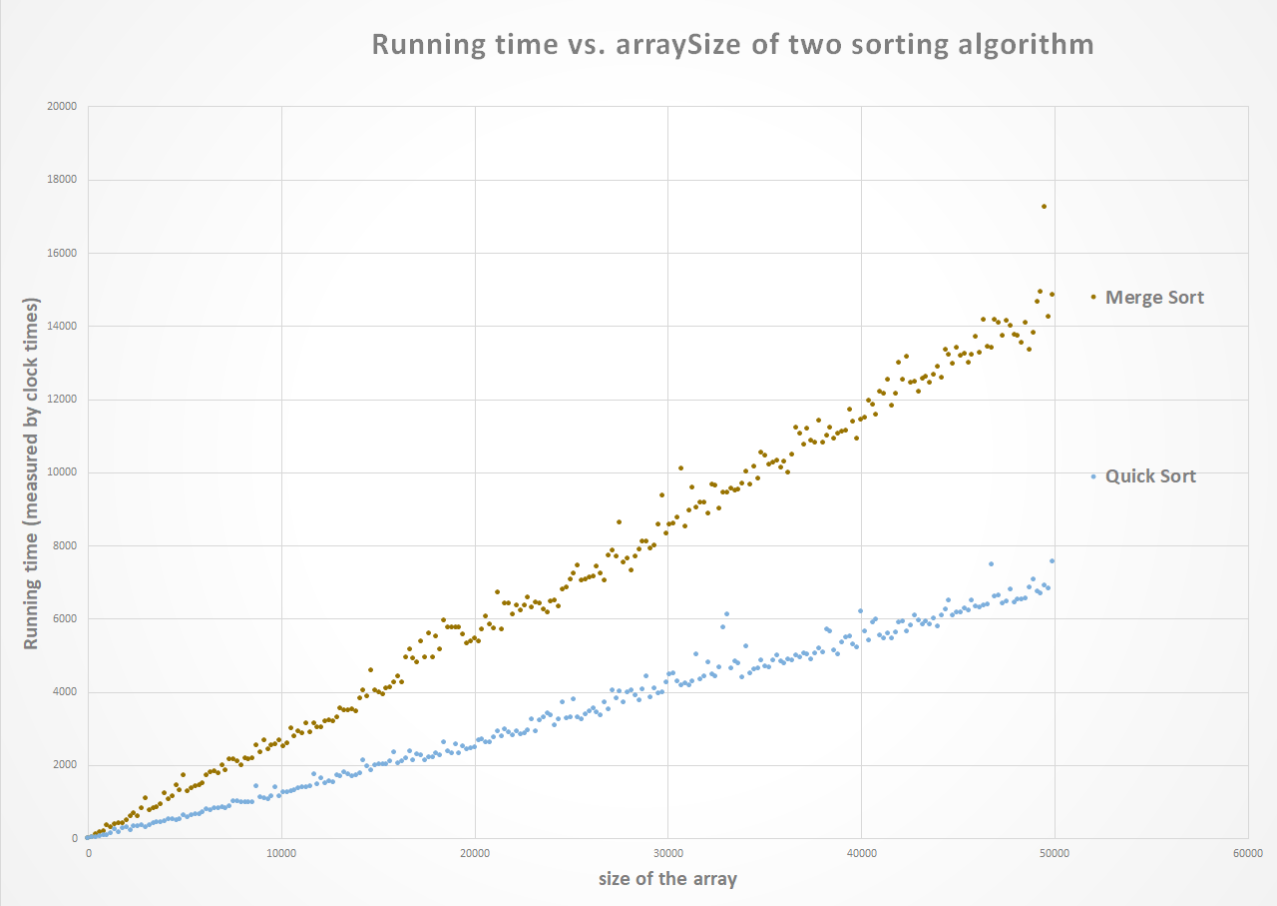

Figure 2: Running time vs. size of array for two sorting algorithms

2 Comparison between two selection algorithm

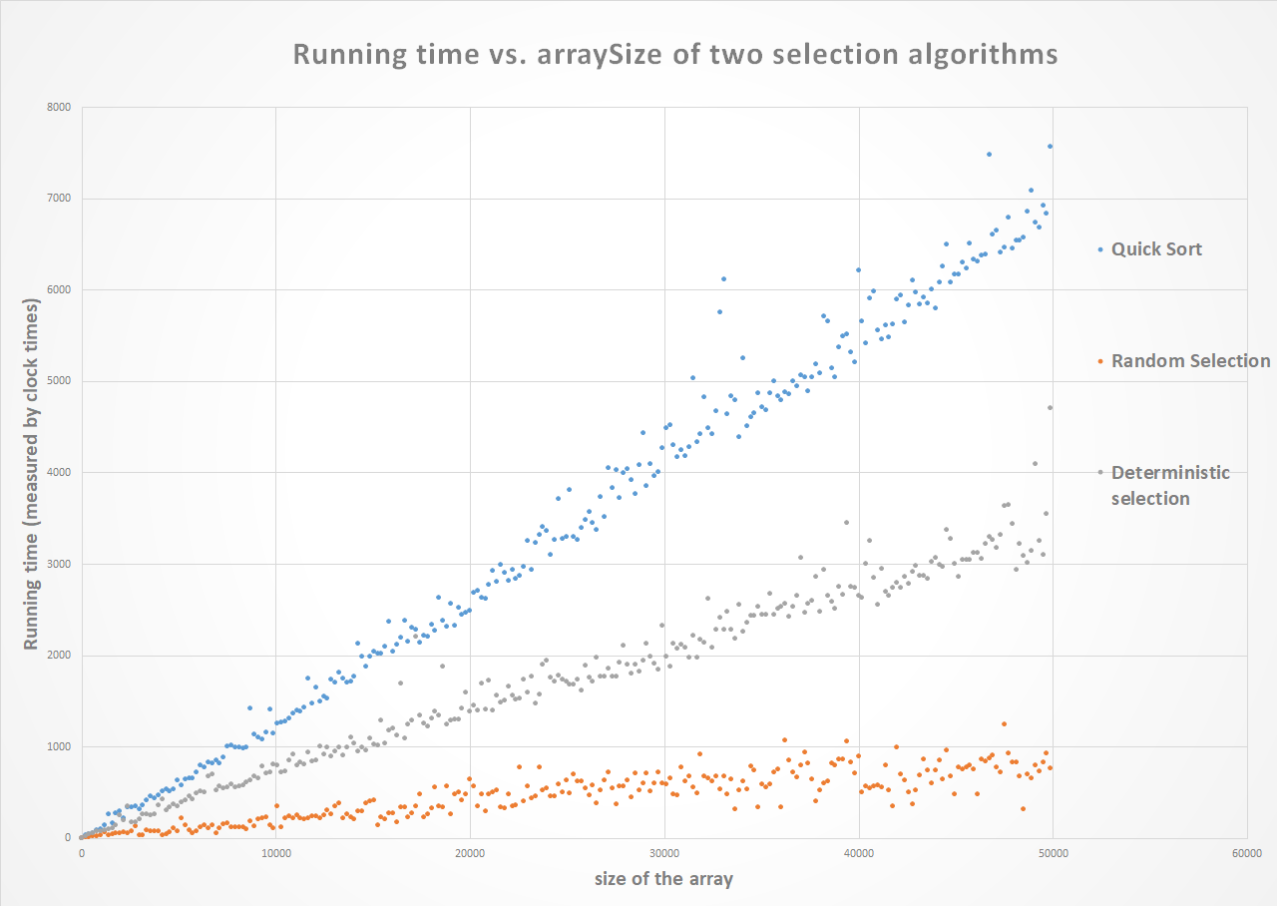

Figure 3: Running time vs. size of array for two selection algorithms with quick sort

In this test, random selection performs better than deterministic selection in less running time. Both two selection algorithms performs better than quick sort in less running time. Differences become much more obvious and significant when the size of the array grows.

Appendix

2.1 Source code

sort.h

```
#ifndef SRC_SORT_H
#define SRC_SORT_H

#include <cstdlib>
#include <iostream>
//#define TIMING
void swap(int, int, int *);
void printArray(int *, int);
void printSearch(int, int);

void bubbleSort(int *, int);

void insertSort(int *, int);

void selectionSort(int *, int);

void mergeSort(int *, int);

void quickSort(int *, int, int);

int rSelect(int *, int, int, int);
int dSelect(int *, int, int, int);

#endif //SRC_SORT_H
```

bubbleSort.cpp

```
    #include "sort.h"
    void bubbleSort(int *nums, int length)
    {
        for(int i = 0 ; i<length-1;i++)
        {
            for (int j = 0; j < length-1-i; j++)
            {
                if(nums[j] > nums[j+1])
                {
                    swap(j,j+1,nums);
                }
            }
        }
    }
```

insertionSort.cpp

```
    #include "sort.h"
    void insertSort(int* nums, int length){
        for(int i = 1 ; i<length;i++)
        {
            int tmp = nums[i],j = i-1;
            while(j>=0 && tmp < nums[j])
            {
```

```
                nums[j+1] = nums[j];
                j--;
            }
            nums[j+1] = tmp;
        }

    }
```

selectionSort.cpp

```
    #include "sort.h"

    void selectionSort(int*nums,int length)
    {
        for(int i = 0 ;i < length - 1;i++)
        {
            int min = INT32_MAX;
            int minAt = i;
            for(int j = i ; j< length;j++)
            {
                if(nums[j]<min)
                {
                    min = nums[j];
                    minAt = j;
                }
            }
            swap(i,minAt,nums);
        }
    }
```

mergeSort.cpp

```
    #include "sort.h"
    static void merge(int *nums, int *l, int lSize, int *r, int rSize)
    {
        int i = 0, j = 0, itr = 0;
        while (i < lSize && j < rSize)
        {
            if (l[i] < r[j])
            {
                nums[itr++] = l[i++];
            }
            else
            {
                nums[itr++] = r[j++];
            }
        }
        while (i < lSize)
        {
            nums[itr++] = l[i++];
        }
        while (j < rSize)
        {
            nums[itr++] = r[j++];
        }
    }
    void mergeSort(int *nums, int length)
```

```
    {
        if (length < 2)
        {
            return;
        }
        int mid = length / 2;
        int *l = new int[mid];
        int *r = new int[length - mid];
        for (int i = 0; i < mid; i++)
        {
            l[i] = nums[i];
        }
        for (int i = mid; i < length; i++)
        {
            r[i - mid] = nums[i];
        }
        mergeSort(l, mid);
        mergeSort(r, length - mid);
        merge(nums, l, mid, r, length - mid);
        delete[] l;
        delete[] r;
    }
```

quickSort.cpp

```
#include "sort.h"
using namespace std;
void quickSort(int* nums,int left,int right)
{
    if(left >= right)
    {
       return ;
    }
    int pivotIndex = rand() % (right - left + 1) + left;
    swap(left,pivotIndex,nums);
    int pivotat = left;
    int i = left + 1, j = i;
    for (; j <= right; j++)
    {
        if (nums[j] < nums[pivotat])
        {
            swap(i,j,nums);
            i++;
        }
    }
    swap(i-1,pivotat,nums);
    quickSort(nums,left, i - 2);
    quickSort(nums,i, right);
}
```

selection.cpp

```
#include "sort.h"

int rSelect(int *nums, int left, int right, int k)
{
    if (left == right)
```

```
6           {
7               return nums[left];
8           }
9           int index = rand() % (right - left + 1) + left;
10          swap(index, left, nums);
11          int pivot = nums[left];
12          int i = left + 1, j = i;
13          for (; i <= right; i++)
14          {
15              if (nums[i] < pivot)
16              {
17                  swap(j, i, nums);
18                  j++;
19              }
20          }
21          swap(j - 1, left, nums);
22          int relative = (j - 1) - left;
23          if (relative == k)
24          {
25              return nums[j - 1];
26          }
27          else if (relative > k)
28          {
29              return rSelect(nums, left, j - 2, k);
30          }
31          else
32          {
33              return rSelect(nums, j, right, k - relative - 1);
34          }
35      }
36
37      static int choosePivot(int *nums, int left, int right)
38      {
39          if (left == right)
40          {
41              return nums[left];
42          }
43          int length = right - left + 1;
44          int divLength = length % 5 == 0 ? length / 5 : length / 5 + 1;
45          int *res = new int[divLength];
46          int itr = 0;
47          for (int i = 0; i < length; i = i + 5)
48          {
49              int tmpRight = (i + 4 >= length) ? length - 1 : i + 4;
50              int tmpLength = tmpRight - i + 1;
51              int *div = new int[tmpLength];
52              for (int j = i; j <= tmpRight; j++)
53              {
54                  div[j - i] = nums[j];
55              }
56              quickSort(div, 0, tmpLength - 1);
57              if (tmpLength % 2 == 1)
58              {
59                  res[itr++] = div[tmpLength / 2];
60              }
61              else
62              {
63                  res[itr++] = (div[tmpLength / 2] + div[tmpLength / 2 - 1]) / 2;
64              }
65              delete[] div;
```

```
        }
        int x = choosePivot(res, 0, divLength - 1);
        delete[] res;
        return x;
    }

    int dSelect(int *nums, int left, int right, int k)
    {
        if (left == right)
        {
            return nums[left];
        }
        int pivot = choosePivot(nums, left, right);
        int pivotat = 0;
        for (int i = left; i < right; i++)
        {
            if (nums[i] == pivot)
            {
                pivotat = i;
            }
        }
        swap(pivotat, left, nums);
        //int pivot = nums[left];
        int i = left + 1, j = i;
        for (; i <= right; i++)
        {
            if (nums[i] < pivot)
            {
                swap(j, i, nums);
                j++;
            }
        }
        swap(j - 1, left, nums);
        if (j - 1 == k)
        {
            return nums[k];
        }
        else if (j - 1 > k)
        {
            return rSelect(nums, left, j - 2, k);
        }
        else
        {
            return rSelect(nums, j, right, k - j + 1);
        }
    }
```

basic_operations.cpp

```
    #include "sort.h"
    using namespace std;
    void swap(int a,int b,int* nums)
    {
        int tmp = nums[a];
        nums[a] = nums[b];
        nums[b] = tmp;
    }
    void printArray(int* nums,int length)
    {
```

```
        for(int i = 0 ; i < length;i++)
        {
            cout << nums[i] << endl;
        }
    }

    void printSearch(int key,int value)
    {
        cout << "The order-" << key <<" item is " << value << endl;
    }
```

main.cpp

```
    #include "sort.h"
    #ifdef TIMING
    #include <time.h>
    #endif
    using namespace std;
    int main()
    {
        int mode = 0;
        cin >> mode;
        int length = 0;
        cin >> length;
        int select = 0;

        int *nums = new int[length];
        if (mode == 5 || mode == 6)
        {
            cin >> select;
        }
        for (int i = 0; i < length; i++)
        {
            int tmp = 0;
            cin >> tmp;
            nums[i] = tmp;
        }
    #ifdef TIMING
        clock_t start = clock();
    #endif
        switch (mode)
        {
        case 0:
        {
            bubbleSort(nums, length);
    #ifndef TIMING
            printArray(nums, length);
    #endif
            break;
        }
        case 1:
        {
            insertSort(nums, length);
    #ifndef TIMING
            printArray(nums, length);
    #endif
            break;
        }
        case 2:
```

```
47          {
48              selectionSort(nums, length);
49      #ifndef TIMING
50              printArray(nums, length);
51      #endif
52              break;
53          }
54          case 3:
55          {
56              mergeSort(nums, length);
57      #ifndef TIMING
58              printArray(nums, length);
59      #endif
60              break;
61          }
62          case 4:
63          {
64              quickSort(nums, 0, length - 1);
65      #ifndef TIMING
66              printArray(nums, length);
67      #endif
68              break;
69          }
70          case 5:
71          {
72              int res = rSelect(nums, 0, length - 1, select);
73      #ifndef TIMING
74              printSearch(select, res);
75      #endif
76              break;
77          }
78          case 6:
79          {
80              int res = dSelect(nums, 0, length - 1, select);
81      #ifndef TIMING
82              printSearch(select, res);
83      #endif
84              break;
85
86              break;
87          }
88
89          default:
90              break;
91          }
92      #ifdef TIMING
93          clock_t finish = clock();
94          int t = (int)(finish - start);
95          float x = ((float)t) / CLOCKS_PER_SEC;
96          cout << length << ",";
97          cout << t << endl;
98      #endif
99          free(nums);
100     }
```

2.2 Test case generation – generate.cpp

```
1   #include <algorithm>
2   #include <cstdlib>
```

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

void randperm(int Num)
{
    vector<int> temp;
    for (int i = 0; i < Num; ++i)
    {
        temp.push_back(i + 1);
    }

    random_shuffle(temp.begin(), temp.end());

    for (int i = 0; i < temp.size(); i++)
    {
        cout << temp[i] << endl;
    }
}

int main(int argc, char **argv)
{
    cout << argv[1] << endl;
    int num = stoi(argv[2]);
    cout << argv[2] << endl;
    if (argc == 3)
    {
        if (num >= 6)
        {
            cout << rand() % 5 << endl;
        }
        else
        {
            cout << "0" << endl;
        }
    }

    randperm(num);
}
```

2.3 Test case processing – app.sh

```
start=1
end=50000
add=198

# Keep this this file fold named testcase, and filefold testcase should be in the same dir with Makefile

cd ../
make
cp main testcase
cd testcase

rm generate
g++ -o generate generate.cpp

rm -r stdInput/
```

```
16
17      mkdir stdInput
18
19      # ================================================================================
20      cd stdInput
21
22      for ((j = 0; j <= 6; j++)); do
23          mkdir $j
24      done
25
26      for ((j = 0; j <= 4; j++)); do
27          cd $j
28          for ((i = $start; i <= $end; i = $i + $add)); do
29              echo $i
30              ../../generate $j $i >"M${j}N${i}"
31          done
32          cd ../
33      done
34
35      for ((j = 5; j <= 6; j++)); do
36          cd $j
37          for ((i = $start; i <= $end; i = $i + $add)); do
38              echo $i
39              ../../generate $j $i 1 >"M${j}N${i}"
40
41          done
42          cd ../
43      done
44
45      cd ../
46
47      # ================================================================================
48
49      rm -r stdOutput
50      mkdir stdOutput
51
52      # ================================================================================
53      cd stdOutput
54
55      for file in $(ls ../stdInput/0); do
56          echo $file
57          ../main <../stdInput/0/$file >>0_out.csv
58      done
59
60      for file in $(ls ../stdInput/1); do
61          echo $file
62          ../main <../stdInput/1/$file >>1_out.csv
63      done
64
65      for file in $(ls ../stdInput/2); do
66          echo $file
67          ../main <../stdInput/2/$file >>2_out.csv
68      done
69
70      for file in $(ls ../stdInput/3); do
71          echo $file
72          ../main <../stdInput/3/$file >>3_out.csv
73      done
74
75      for file in $(ls ../stdInput/4); do
```

```
76          echo $file
77          ../main <../stdInput/4/$file >>4_out.csv
78      done
79
80      for file in $(ls ../stdInput/5); do
81          echo $file
82          ../main <../stdInput/5/$file >>5_out.csv
83      done
84
85      for file in $(ls ../stdInput/6); do
86          echo $file
87          ../main <../stdInput/6/$file >>6_out.csv
88      done
89
90      cd ../
91      # ==========================================================================
```