

Performances of Two Selection Algorithm

Bingcheng HU

516021910219

Introduction

In order to study the performances of these two selection algorithms, I generated different size of arrays and compared the running speed of them. Small size of arrays were run for several times so that the result can be more accurate.

Comparison of algorithms

There is no limitation of runtime for all algorithms, because all these two algorithms have similar time complexity $O(n)$. Then I used DataGrapg to plot two graphs, one of small test cases, and another of all cases.

Loop several times

With `#define LOOP_TIME 20` we can run it for 20 times so that the result can be more accurate. (Please check `performance.cpp`)

```
1  long time_all = 0;
2  for (int lo = 0; lo < LOOP_TIME; lo++)
3  {
4      int arr_copy[lines];
5      //use deep copy to make arr_copy evry turn
6      memset(arr_copy,0, lines*sizeof(int));
7      memcpy(arr_copy,arr, lines*sizeof(int));
8      start = clock();
9      fn[i](arr_copy, lines);
10     end = clock();
11     time_all += (end - start);
12 }
```

```

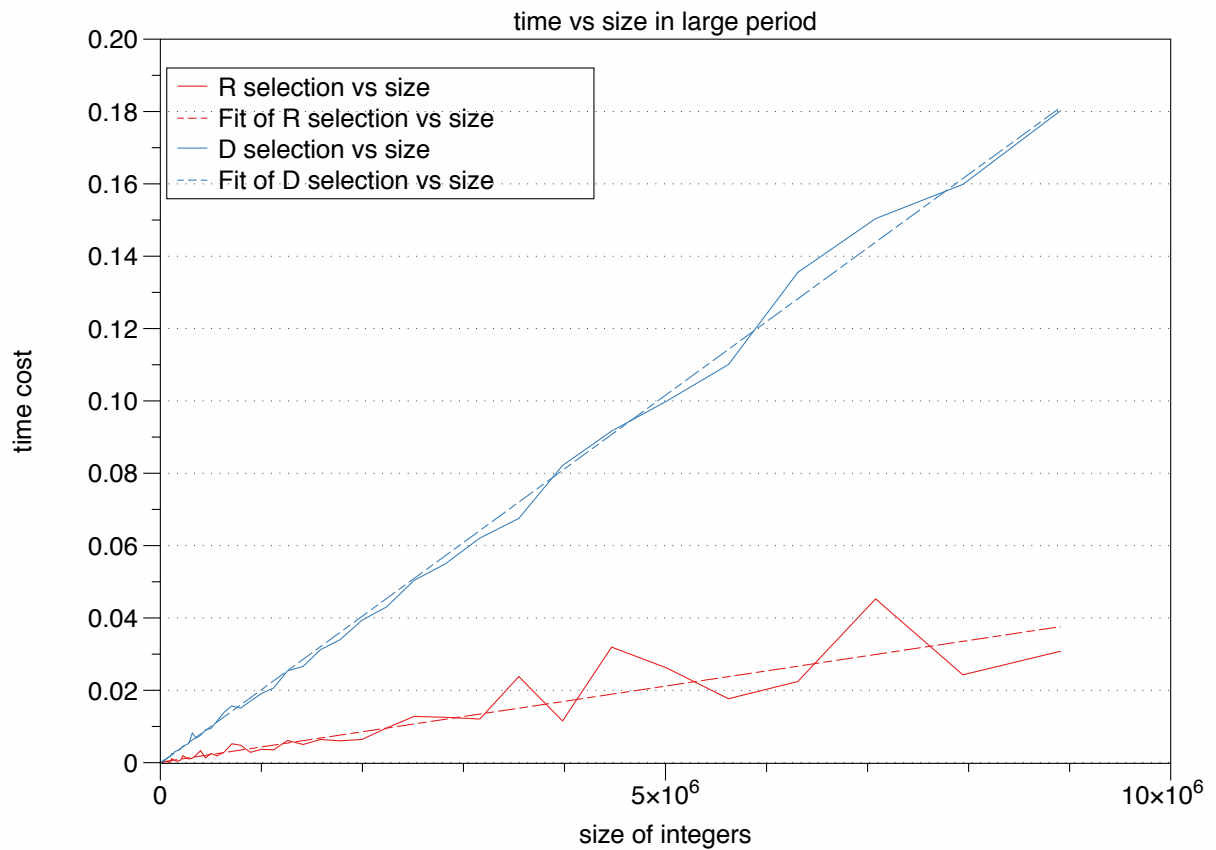
13 cout<<"Sort algorithm is ["<<sortName[i]<<"],";
14 double time_run = (double)time_all / CLOCKS_PER_SEC / LOOP_TIME;
15 cout << "Running time: " <<time_run<< endl;

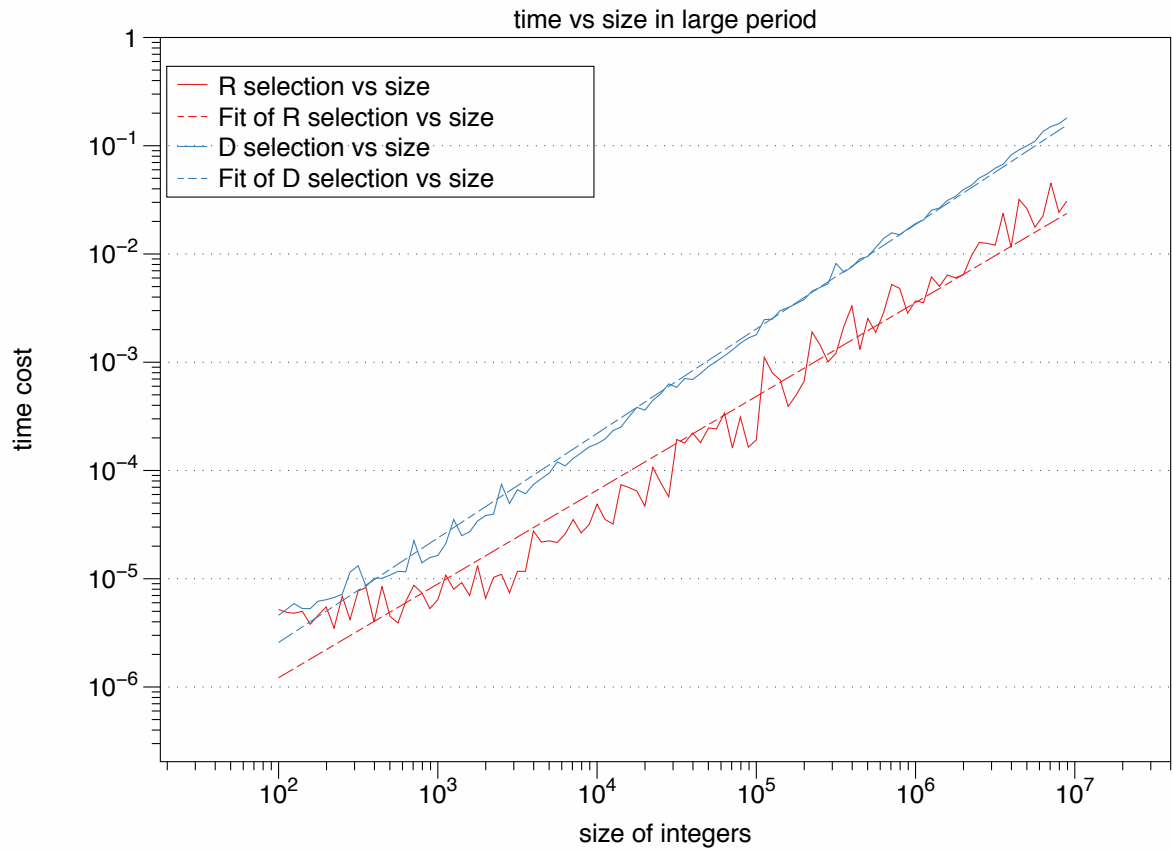
```

Big data analysis

From **Figure 1**, all these line looks similar, so it's meaning less. So we make *log* at both *x* and *y* axis. As **Figure 2** shows, we can find that both Random Selection and Deterministic Selection Algorithm have the runtime $O(n)$, and they are parallel when the size of the numbers is larger than 10^4 .

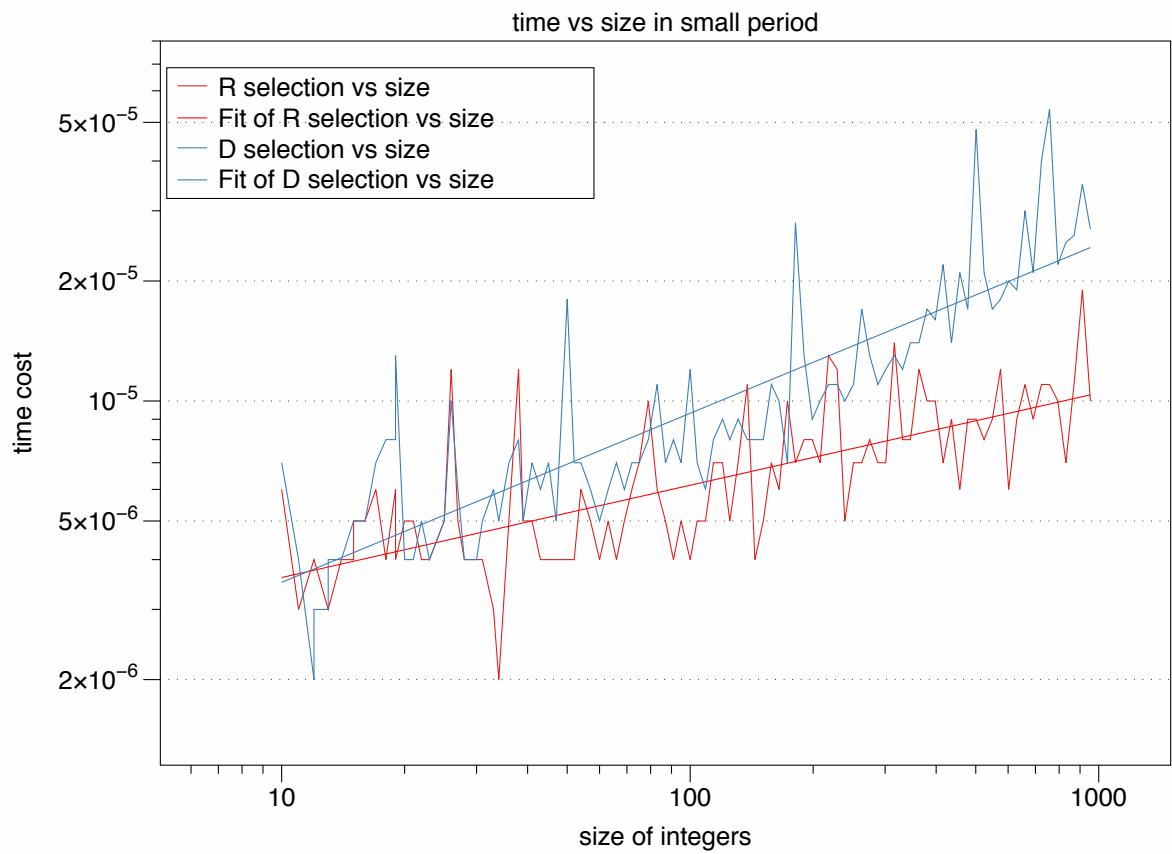
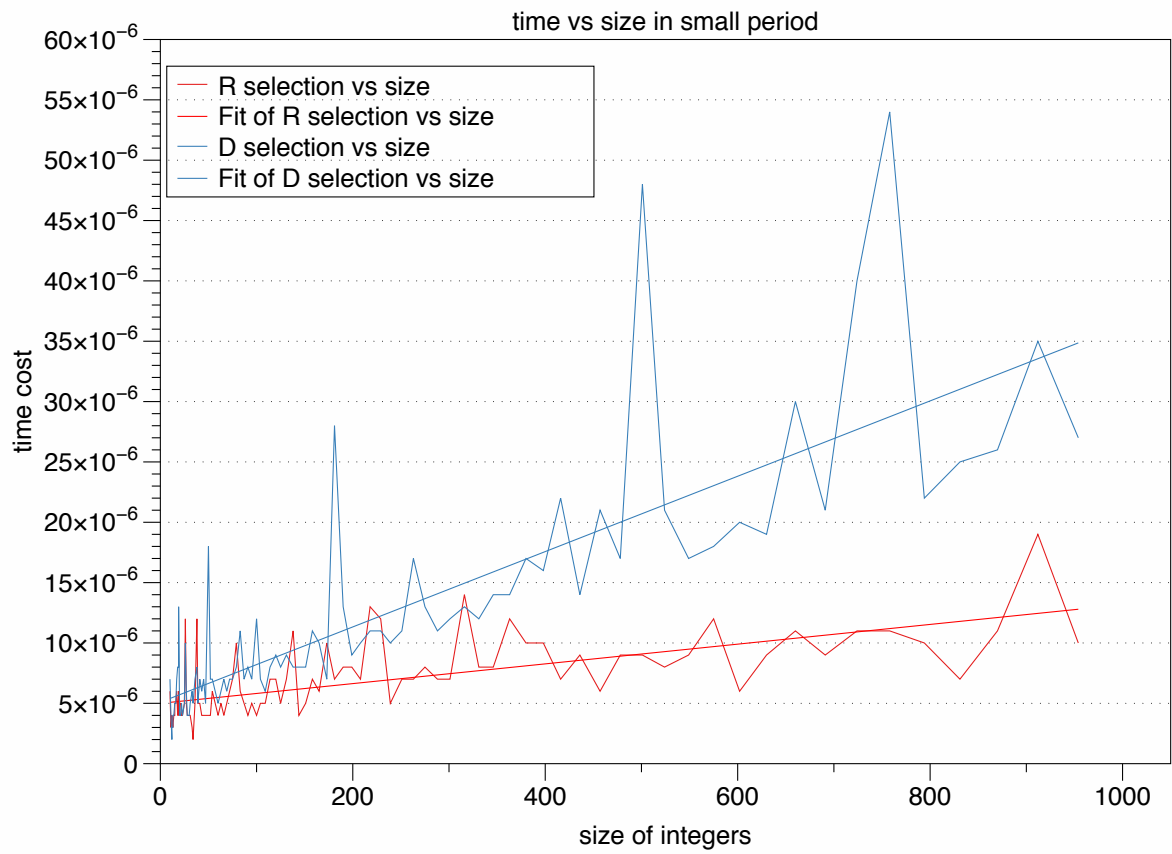
So What happend when the size of the numbers is smaller than 10^3 ? it seems that these two lines are connected to each other. Let's see the Small data analysis.





Small data analysis

From **Figure 4** we can see that when the data size is small, it's so hard to recognize the runtime because the datas are so unstable. There are so many data hopping all the time, though I used Loop several times in my code, it doesn't work because the performance of my computer is not stable.



Conclusion

According to the data analysis, Dselect is not as good as Rselect in practice, because it has worse constants and it's not in-place.

With worse constants it will cost more time and you can find that in *Figure 1*. It's not in-place so it will cost more time to acquire for datas, which is inefficiency.

So at most conditions, we should use Rselect in practice.

Appendix

The project files

selection.h

```
1  #ifndef SELECTION_H
2  #define SELECTION_H
3
4  int random_selection(int* arr, const int n, const int order);
5  // Randomized selection algorithm
6  // MODIFIES: *arr
7  // EFFECTS: select i-th smallest element in the array
8
9  int deterministic_selection(int* arr, const int n, const int order);
10 // Deterministic selection algorithm
11 // MODIFIES: *arr
12 // EFFECTS: select i-th smallest element in the array
13
14 #endif
```

selection.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <cassert>
4  #include <ctime>
5  #include "selection.h"
6
7  using namespace std;
8
9  static void int_append(int *arrA, const int *arrB, const int s){
10     // MODIFIES: *arrA
11     // EFFECTS: append first "s" int in arrB to the beginning of arrA.
12     assert(s >= 0);
13     if(s == 0) return;
14     for (int i = 0; i < s; ++i)
15     {
16         arrA[i] = arrB[i];
17     }
```

```

18 }
19
20 static int random_pivot(int* arr, const int n){
21 // Choose pivot p from arr uniformly at random;
22 // Partition arr using pivot p;
23 // Let j be the index of p, return j;
24     const int size = n;
25     int BL = 0, BR = size-1;
26     int * B = new int[size];
27     int * A = arr;
28     srand((unsigned)time(NULL));
29     const int pivotat = rand()%size;
30     const int t = A[pivotat];
31     for (int i = 0; i < size; ++i)
32     {
33         if(i == pivotat) continue;
34         if(A[i] > t) B[BR--] = A[i];
35         else B[BL++] = A[i];
36     }
37     assert(BL == BR);
38     B[BL] = t;
39     int_append(A, B, size);
40     delete[] B;
41     return BL;
42 }
43
44 static void insertion_sort(int *arr, const int n){
45 // MODIFIES: *arr
46 // EFFECTS: sort integers arr[] in ascending order with
47 // insertion_sort.
48     for (int i = 1; i < n; ++i)
49     {
50         int t = arr[i];
51         int j = i;
52         while (j >= 1)
53         {
54             if (arr[j - 1] > t)
55             {
56                 arr[j] = arr[j - 1];
57                 j--;
58             }
59             else break;
60         }
61         arr[j] = t;
62     }
63 }
64
65 static int Deterministic_pivot_helper(int* arr, int n){
66     if(n == 1) return arr[0];
67     int full_bucket = n/5;
68     int arr_medians_size = full_bucket+(n%5+4)/5;

```

```

69     int* arr_medians = new int [arr_medians_size];
70     int incomplete_bucket = arr_medians_size - full_bucket;
71     for (int i = 0; i < full_bucket; ++i)
72     {
73         int* arr_break_5 = arr + i*5;
74         insertion_sort(arr_break_5, 5);
75         arr_medians[i] = arr_break_5[2];
76     }
77     if (incomplete_bucket != 0)
78     {
79         int incomplete_bucket_size = n%5;
80         int* arr_break_5 = arr + full_bucket*5;
81         insertion_sort(arr_break_5, incomplete_bucket_size);
82         arr_medians[full_bucket] =
arr_break_5[incomplete_bucket_size/2];
83     }
84     int pivot = Deterministic_pivot_helper(arr_medians,
arr_medians_size);
85     return pivot;
86 }
87
88 static int partition_array(int *arr, const int n, const int pivot){
89     // MODIFIES: *arr
90     // EFFECTS: choose a pivotat then Move pivot to its correct place
in the array.
91     const int size = n;
92     int BL = 0, BR = size-1;
93     int * B = new int[size];
94     int * A = arr;
95     const int t = pivot;
96     for (int i = 0; i < size; ++i)
97     {
98         // cerr<<arr[i]<<" ";
99         if(A[i] == t) continue;
100        if(A[i] > t) B[BR--] = A[i];
101        else B[BL++] = A[i];
102    }
103    // cerr<<endl;
104    // cerr<<"pivot = "<<pivot<<"", size = "<<size<<"", BL = "<<BL<<"",
BR = "<<BR<<endl;
105    assert(BL <= BR);
106
107    for (int i = BL; i <= BR; ++i)
108    {
109        B[i] = t;
110    }
111    int_append(A, B, size);
112    delete[] B;
113    return BL;
114 }
115
116 static int Deterministic_pivot(int* arr, int n){

```

```

117 // Choose pivot p from arr uniformly at deterministic;
118 // Partition arr using pivot p;
119 // Let j be the index of p, return j;
120     int pivot = Deterministic_pivot_helper(arr, n);
121     int j = partition_array (arr, n, pivot);
122     return j;
123 }
124
125 int random_selection(int* arr, const int n, const int order){
126     if(n == 1) return arr[0];
127     int j = random_pivot(arr, n);
128     if(j == order) return arr[order];
129     if(j > order) {
130         int* arr_left = arr;
131         int length = j;
132         return random_selection(arr_left, length, order);
133     }
134     else{
135         int* arr_right = arr + j + 1;
136         int length = n - j - 1;
137         return random_selection(arr_right, length, order-j-1);
138     }
139 }
140
141 int deterministic_selection(int* arr, const int n, const int order){
142     if(n == 1) return arr[0];
143     int j = Deterministic_pivot(arr, n);
144     if(j == order) return arr[order];
145     if(j > order) {
146         int* arr_left = arr;
147         int length = j;
148         return deterministic_selection(arr_left, length, order);
149     }
150     else{
151         int* arr_right = arr + j + 1;
152         int length = n - j - 1;
153         return deterministic_selection(arr_right, length, order-j-1);
154     }
155 }

```

main.cpp

```

1  #include <iostream>
2  #include <cstdlib>
3  #include <assert.h>
4  #include <ctime>
5  #include "selection.h"
6
7  #define SELECTION_WAY_SIZE 2
8  // #define SELECTION_DEBUG
9

```



```

10 using namespace std;
11
12 const string selectionName[] = {
13     "R selection", "D selection", "ERROR_SELECTION_Name"
14 };
15
16 #ifdef SELECTION_DEBUG
17 void insertion_sort(int *arr, const int n){
18     for (int i = 1; i < n; ++i)
19     {
20         int t = arr[i];
21         int j = i;
22         while (j >= 1)
23         {
24             if (arr[j - 1] > t)
25             {
26                 arr[j] = arr[j - 1];
27                 j--;
28             }
29             else break;
30         }
31         arr[j] = t;
32     }
33 }
34 #endif
35
36
37 int main(int argc, char *argv[]) {
38     int (*const fn[SELECTION_WAY_SIZE])(int*, const int, const int) = {
39         random_selection,
40         deterministic_selection
41     };
42     int selection_algorithm;
43     cin >> selection_algorithm;
44     assert(selection_algorithm >= 0 && selection_algorithm <
SELECTION_WAY_SIZE);
45     #ifdef SELECTION_DEBUG
46         cout<<"selection algorithm is ["
<<selectionName[selection_algorithm]<<"],"<<endl;
47     #endif
48
49     int n;
50     cin >> n;
51     int order;
52     cin >> order;
53     int *arr = new int[n];
54     for (int i = 0; i < n; ++i)
55     {
56         cin>>arr[i];
57     }
58     int i_th_small;
59     clock_t start, end;

```

```

60     start = clock();
61     i_th_small = fn[selection_algorithm](arr, n, order);
62     end = clock();
63
64     #ifdef SELECTION_DEBUG
65     int arr_copy[n];
66     //use deep copy to make arr_copy evry turn
67     memset(arr_copy,0, n*sizeof(int));
68     memcpy(arr_copy,arr, n*sizeof(int));
69     insertion_sort(arr_copy, n);
70     cout << "RunTime:" << (double)(end - start) / CLOCKS_PER_SEC <<
endl;
71     cout << "#["<<order<<"]smallest: ["<<i_th_small<<"], real:["
<<arr_copy[order]<<"]"<<endl;
72     #endif
73     cout<<"The order-"<<order<<" item is "<<i_th_small<<endl;
74     delete[] arr;
75     return 0;
76 }

```

Makefile

```

1  all: main.o selection.o
2      g++ -std=c++11 -O3 -g -Wall -o main main.o selection.o
3
4  auto: make_auto
5      ./autogen
6
7  make_auto: auto_gen.o
8      g++ -std=c++11 -O3 -g -Wall -o autogen auto_gen.o
9
10 auto_gen.o: auto_gen.cpp
11     g++ -std=c++11 -O3 -g -Wall -c auto_gen.cpp
12
13 t: make_test
14     ./test
15
16 make_test: simple_test.o selection.o
17     g++ -std=c++11 -O3 -g -Wall -o test simple_test.o selection.o
18
19 simple_test.o: simple_test.cpp
20     g++ -std=c++11 -O3 -g -Wall -c simple_test.cpp
21
22 test0: gen all
23     ./generate 0 40000 2000 > input.data
24     ./main < input.data
25
26 test1: gen all
27     ./generate 1 40000 2000 > input.data
28     ./main < input.data
29

```

```

30 gen:
31     g++ -std=c++11 -O3 -Wall -o generate gen_rand.cpp
32
33 p: perf
34     ./perform
35
36 perf: performance.o selection.o
37     g++ -std=c++11 -O3 -g -Wall -o perform performance.o selection.o
38
39 performance.o: performance.cpp
40     g++ -std=c++11 -O3 -g -Wall -c performance.cpp
41
42 main.o: main.cpp
43     g++ -std=c++11 -O3 -g -Wall -c main.cpp
44
45 selection.o: selection.cpp
46     g++ -std=c++11 -O3 -g -Wall -c selection.cpp
47
48 v:
49     valgrind --leak-check=full ./main < input.data
50
51 tar:
52     tar czvf p2.tar main.cpp selection.cpp selection.h p2.pdf
53
54 clean:
55     rm -f ./main *.o *.data test generate autogen perform
56

```

auto_gen.cpp

```

1  #include <iostream>
2  #include <stdlib.h>
3  #include <sstream>
4  #include <assert.h>
5  #include <fstream>
6  #include <math.h>
7  using namespace std;
8  int main(int argc, char *argv[]) {
9      ofstream oFile;
10
11     for (int i = 0; i < 100; ++i)
12     {
13         ostringstream path_stream;
14         path_stream<<"./data_file/"<<i<<".data";
15         oFile.open(path_stream.str());
16         double k = 1.0 + 2.0/100*(i);
17         int ek = pow(10, k);
18         cerr<<ek<<" ";
19
20         oFile<<0<<endl;
21         oFile<<ek<<endl;

```

```

22         oFile<<0<<endl;
23         for (int i = 0; i < ek; ++i)
24         {
25             int k = rand48();
26             oFile << k <<endl;
27         }
28         oFile.close();
29     }
30 }

```

Generated files look like this.



performance.cpp

This program will produce a CSV table as 3.1.7 shows.

```

1  #include <iostream>
2  #include <iomanip>
3  #include <sstream>
4  #include <cstdlib>
5  #include <assert.h>
6  #include <ctime>
7  #include <fstream>
8  #include "selection.h"
9
10 using namespace std;
11
12 #define SELECTION_WAY_SIZE 2
13 #define file_num 100
14 #define LOOP_TIME 40
15 // #define SORT_DEBUG
16
17 void debug_print(char TAG, string deb_string){
18     if(TAG == 'v') cerr<< deb_string;
19 }
20
21 void Delay(int time){
22     clock_t now = clock();
23     while( clock() - now < time );
24 }
25

```

```

26  const string selectionName[] = {
27      "R selection", "D selection", "ERROR_SELECTION_Name"
28  };
29
30  bool safe_open_file(ifstream& i_file, string file_name){
31      ostringstream debug_stream;
32      i_file.open(file_name.c_str());
33      if (i_file.fail()) {
34          cout<<"Error: Cannot open file "<< file_name<<"!"<<endl;
35          exit(0);
36      }
37      debug_stream<<"file opened success!"<<endl;
38      debug_print('v', debug_stream.str());
39      debug_stream.clear();
40      return true;
41  }
42
43  static int int_size[] = {
44      10, 10, 10, 11, 12, 12, 13, 13, 14, 15, 15, 16, 17, 18, 19, 19,
45      20, 21, 22, 23, 25, 26, 27, 28, 30, 31, 33, 34, 36, 38, 39, 41, 43,
46      45, 47, 50, 52, 54, 57, 60, 63, 66, 69, 72, 75, 79, 83, 87, 91, 95,
47      100, 104, 109, 114, 120, 125, 131, 138, 144, 151, 158, 165, 173, 181,
48      190, 199, 208, 218, 229, 239, 251, 263, 275, 288, 301, 316, 331, 346,
49      363, 380, 398, 416, 436, 457, 478, 501, 524, 549, 575, 602, 630, 660,
50      691, 724, 758, 794, 831, 870, 912, 954 };
51
52  bool jump_j[] = {false, false, false, false, false, false};
53
54  int main(int argc, char *argv[]) {
55      int (*const fn[SELECTION_WAY_SIZE])(int*, const int, const int) =
56      {
57          random_selection,
58          deterministic_selection
59      };
60
61      clock_t start, end;
62      ofstream outFile;
63      outFile.open("data.csv", ios::out);
64      outFile << "size" << " " << selectionName[0] << " " << selectionName[1]
65      << " " << endl;
66      for (int j = 0; j < file_num; ++j)
67      {
68          ifstream iFile;
69
70          ostringstream path_stream;
71          path_stream<<".\\data_file\\"<<j<<".data";
72          safe_open_file(iFile, path_stream.str());
73          int lines;
74          iFile >> lines;
75          iFile >> lines;
76          int meaning_less;

```

```

70         iFile >> meanning_less;
71
72
73         int *arr = new int[lines];
74         int baz;
75         for (int i = 0; i < lines; ++i)
76         {
77             iFile >> baz;
78             arr[i] = baz;
79         }
80         outFile << int_size[j]<<",";
81         for (int i = 0; i < SELECTION_WAY_SIZE; ++i)
82         {
83             if(jump_j[i] == true){
84                 outFile << ""<<",";
85                 cerr<<"jump "<<selectionName[i]<<" with "<<int_size[j]
<<" size!"<<endl;
86                 continue;
87             }
88             if(int_size[j] < 1000){
89                 cerr<<"delay at "<<int_size[j]<<" size"<<endl;
90                 Delay(1000);
91             } else Delay(500);
92
93
94             // int arr_copy[lines];
95             // //use deep copy to make arr_copy evry turn
96             // memset(arr_copy,0, lines*sizeof(int));
97             // memcpy(arr_copy,arr, lines*sizeof(int));
98             // start = clock();
99             // fn[i](arr_copy, lines, 0);
100            // end = clock();
101
102            long time_all = 0;
103            for (int lo = 0; lo < LOOP_TIME; lo++)
104            {
105                int arr_copy[lines];
106                //use deep copy to make arr_copy evry turn
107                memset(arr_copy,0, lines*sizeof(int));
108                memcpy(arr_copy,arr, lines*sizeof(int));
109                start = clock();
110                fn[i](arr_copy, lines, 0);
111                end = clock();
112                time_all += (end - start);
113            }
114
115
116            cout<<"Sort algorithm is ["<<selectionName[i]<<"],";
117            double time_run = (double)time_all / CLOCKS_PER_SEC /
LOOP_TIME;
118            cout << "Running time: " <<time_run<< endl;
119            if (time_run >= 24.0)

```

```

120         {
121             jump_j[i] = true;
122         }
123         outFile << time_run<<",";
124     }
125     outFile <<endl;
126
127
128     iFile.close();
129     delete[] arr;
130 }
131 outFile.close();
132 return 0;
133 }

```

scv Table

You can get full CSV table at [HERE](#).

This table was generated by MacBook Pro 2015, i5, 256G.

size	R selection	D selection
100	4.2e-06	8.4e-06
112	2.8e-06	6.4e-06
125	2.8e-06	1.66e-05
141	3.8e-06	6.6e-06
158	2.6e-06	6.6e-06
177	4.2e-06	6.6e-06
199	3.6e-06	8e-06
223	4e-06	8e-06
251	6e-06	9e-06
281	7e-06	1.38e-05
...
1778279	0.0146598	0.0363128
...
7943282	0.0283762	0.157991

DataGraph

I used [DataGraph](#) to generate images on MacBook Pro.

