UM–SJTU Joint Institute

Data Structures and Algorithms

(VE281)

Homework 2

Name: Ji Xingyou ID: 515370910197
Date: 10 October 2017

Contents

1 Theoretical Data **3**

2 Result Analysis **3**

3 Appendix **5**

3.1 Linear time selection algorithms 5

3.2 Run-time calculation 13

3.3 Visualization 22

1 Theoretical Data

As is discussed in the class, we can get the following table summarizing the time complexity for each algorithms.

	Worst case complexity	Average case complexity	In place	Stable
Quick sort in place	$O(N^2)$	$O(NlogN)$	Yes	No
Rselect	$\Theta(n^2)$	$O(N)$	Yes	Yes
Dselect		$O(N)$	No	Yes

Table 1: Time complexity of comparison sorting

In this report, we will first implement all these three algorithms, and then test the run time for each of them to see whether the above table makes sense.

The implementation of the algorithms is attached in the appendix.

2 Result Analysis

After finishing implementing the above three algorithms, I wrote another program to test the run time of each algorithm. In this program, I set two clocks, noting the starting and finishing instance. To avoid uncertainty in the data, I wrote a while loop to create 10 different i each time so that I could get the average value. There is one thing which requires extra carefulness. That is, we must ensure that every time inside the while loop, all the six selection should meet the identical array.

The array size I chose is 10, 100, 1000, 5000, 10000, 50000, 100000 and 1000000.

The run time data for each algorithms is listed in table 2.

Array size	Quick sort in place	Rselect	Dselect
10	7	3	9
100	57	15	69
1000	983	191	1089
5000	4701	832	3921
10000	12018	1944	8860
50000	54800	6622	32207
100000	106287	12056	61333
1000000	1180554	113349	550122

Table 2: Run time of linear time selection

The run time comparison is shown in figure 1.

Combining the table and figure above, we can conclude the following points.

1. For each linear time selection algorithm, the run time increases as the size of the array increases.

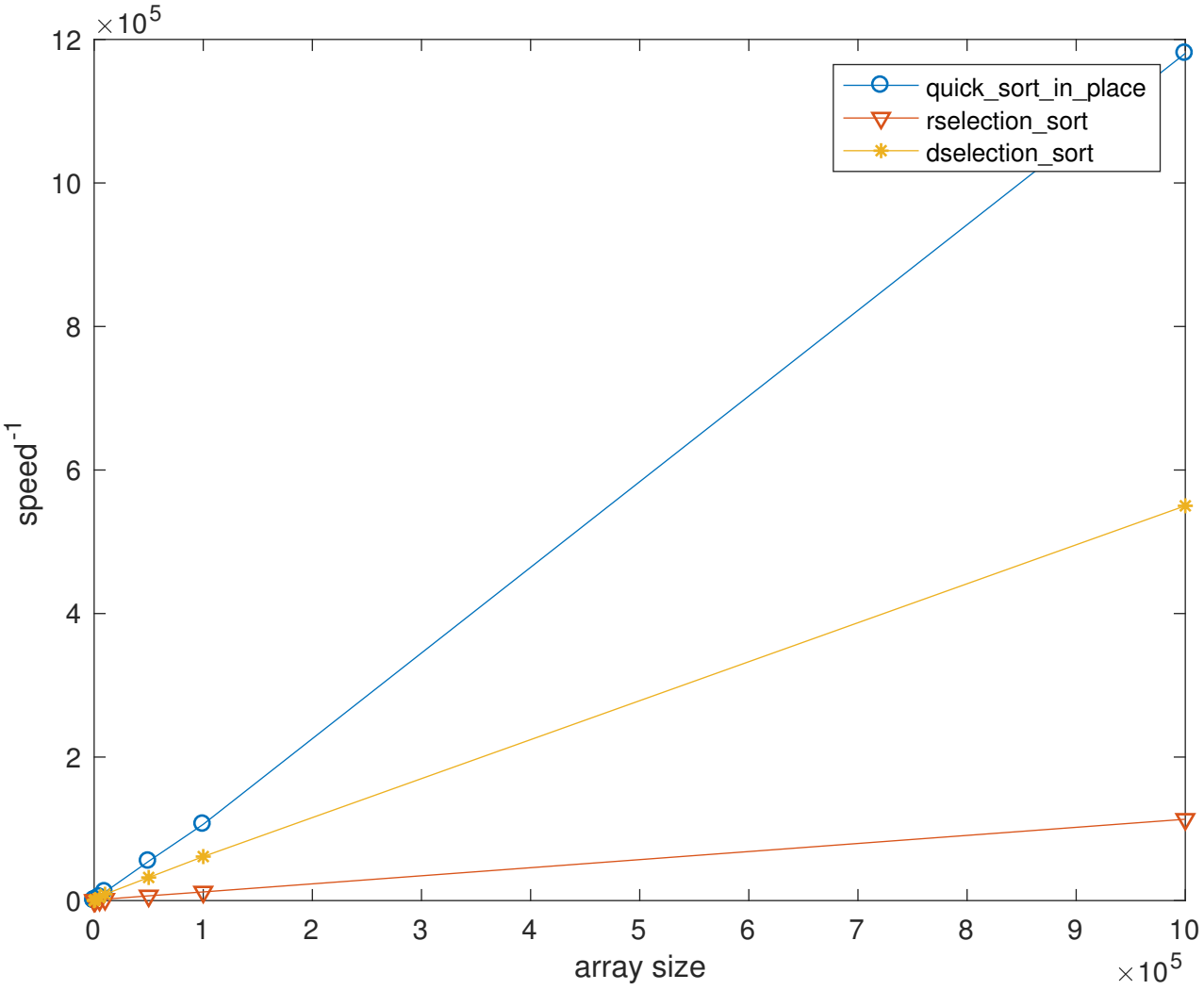

Figure 1: Run time comparison

2. For any given array size, Rselect always has a leading performance, while Dselect ranks second and quick sort third.

From the conclusion listed above, we can see that it fits the time complexity shown in table 1, which means the algorithms make sense.

This inspires me that in future learning, when dealing with selection, I should use Rselect as often as possible since it has the best performance.

3 Appendix

3.1 Linear time selection algorithms

```
1  //
2  //  main.cpp
3  //  project2
4  //
5  //  Created by           on 2017/9/27.
6  //  Copyright    2017             . All rights reserved.
7  //
8
9  #include <iostream>
10 #include <fstream>
11 #include <sstream>
12 #include <string>
13 #include <cstdlib>
14 #include <climits>
15 #include <ctime>
16 #include <cassert>
17
18 using namespace std;
19
20 int Rselect(int *arr,int n,int i);
21
22 int Dselect(int *arr,int n,int i);
23
24 int partition(int *arr,int left,int right,int size,int pivotat);
25
```

```
26 void selection_sort(int *arr,int n);
27 ////////////////////////////
28 ////////////////////////////
29 int main(int argc, const char * argv[])
30 {
31     int choice;
32     cin>>choice;
33     int n;
34     cin>>n;
35     if(n==0)
36     {
37         return 0;
38     }
39     int i;
40     cin>>i;
41     if(i>=n||i<0)
42     {
43         return 0;
44     }
45     int a[n];
46     for(int t=0;t<n;++t)
47     {
48         cin>>a[t];
49     }
50     int *arr=a;
51     int result=0;
52     switch(choice)
53     {
```

```
54          case 0:
55              result=Rselect(arr,n,i);
56              break;
57          case 1:
58              result=Dselect(arr,n,i);
59          default:
60              break;
61      }
62      cout<<"The order-"<<i<<" item is "<<result<<endl;
63  //      srand(time(0));
64  //      int times=100;
65  //      while(times--)
66  //      {
67  //          int size=100;
68  //          int index=rand()%size;
69  //          int *arr=new int [size];
70  //          int brr[size];
71  //          for(int i=0;i<size;++i)
72  //          {
73  //              arr[i]=rand()%size;
74  //          }
75  //          for(int i=0;i<size;++i)
76  //          {
77  //              brr[i]=arr[i];
78  //          }
79  //          sort(arr,arr+size);
80  //          int p=arr[index];
81  //          int q=Rselect(brr,size,index);
```

```
//         cout<<p-q<<endl;
//         delete [] arr;
//     }
    return 0;
}
////////////////////////////
////////////////////////////
int Rselect(int *arr,int n,int i)
{
    if(n==1)
    {
        return *arr;
    }
    int pivotat=rand()%n;
    pivotat=partition(arr,0,n-1,n,pivotat);
    if(pivotat==i)
    {
        return arr[pivotat];
    }
    else if(pivotat>i)
    {
        return Rselect(arr,pivotat,i);
    }
    else
    {
        return Rselect(arr+pivotat+1,n-pivotat-1,i-pivotat-1);
    }
}
```

```
110 ////////////////////////////
111 ////////////////////////////
112 int Dselect(int *arr,int n,int i)
113 {
114     if(n<=1)
115     {
116         return *arr;
117     }
118     int group_num=n/5;
119     int last=n%5;
120     if(last==0)
121     {
122         last=5;
123     }
124     else
125     {
126         group_num++;
127     }
128     int *C=new int[group_num];
129     int *temp=new int[5];
130     int size=0;
131     for(int p=0;p<group_num;++p)
132     {
133         if(p==group_num-1)
134         {
135             size=last;
136         }
137         else
```

```
138         {
139             size=5;
140         }
141         for(int q=0;q<size;++q)
142         {
143             temp[q]=arr[q+5*p];
144         }
145         selection_sort(temp,size);
146         C[p]=temp[size/2];
147     }
148     int pivot=Dselect(C,n/5,n/10);
149     delete [] C;
150     delete [] temp;
151     int pivotat=0;
152     for(int p=0;p<n;++p)
153     {
154         if(arr[p]==pivot)
155         {
156             pivotat=p;
157         }
158     }
159     pivotat=partition(arr,0,n-1,n,pivotat);
160     if(pivotat==i)
161     {
162         return arr[pivotat];
163     }
164     else if(pivotat>i)
165     {
```

```
166         return Dselect(arr,pivotat,i);
167     }
168     else
169     {
170         return Dselect(arr+pivotat+1,n-pivotat-1,i-pivotat-1);
171     }
172 }
173 ////////////////////////////
174 ////////////////////////////
175 int partition(int *arr,int left,int right,int size,int pivotat)
176 {
177     swap(arr[0],arr[pivotat]);
178     int i=1;
179     int j=size-1;
180     while(true)
181     {
182         while(i<size-1&&arr[i]<arr[0])
183         {
184             ++i;
185         }
186         while(j>0&&arr[j]>=arr[0])
187         {
188             --j;
189         }
190         if(i<j)
191         {
192             swap(arr[i],arr[j]);
193         }
```

```
194         else
195         {
196             break;
197         }
198     }
199     swap(arr[0],arr[j]);
200     return j;
201 }
202 ////////////////////////////
203 ////////////////////////////
204 void selection_sort(int *arr,int n)
205 {
206     for(int i=0;i<n-1;++i)
207     {
208         int index=i;
209         for(int j=i+1;j<n;++j)
210         {
211             if(arr[j]<arr[index])
212             {
213                 index=j;
214             }
215         }
216         if(index!=i)
217         {
218             int tmp=arr[index];
219             arr[index]=arr[i];
220             arr[i]=tmp;
221         }
```

```
222     }
223 }
```

3.2 Run-time calculation

```
1  //
2  //  main.cpp
3  //  run_time_study
4  //
5  //  Created by           on 2017/10/9.
6  //  Copyright    2017             . All rights reserved.
7  //
8
9  #include <iostream>
10 #include <fstream>
11 #include <sstream>
12 #include <string>
13 #include <cstdlib>
14 #include <climits>
15 #include <ctime>
16 #include <cassert>
17
18 using namespace std;
19
20 int Rselect(int *arr,int n,int i);
21
22 int Dselect(int *arr,int n,int i);
23
24 int partition(int *arr,int left,int right,int size,int pivotat);
```

```
25
26 void selection_sort(int *arr,int n);
27
28 void quick_sort_in_place(int *arr,int left,int right,int size);
29 ////////////////////////////
30 ////////////////////////////
31 int main(int argc, const char * argv[])
32 {
33     int lines=1000000;
34     srand(time(0));
35     int t=10;
36     long temp0=0;
37     long temp1=0;
38     long temp2=0;
39     long time0=0;
40     long time1=0;
41     long time2=0;
42     clock_t start,finish;
43     int arr[lines];
44     for(int i=0;i<lines;++i)
45     {
46         arr[i]=rand();
47     }
48     int brr[lines];
49     for(int j=0;j<lines;++j)
50     {
51         brr[j]=arr[j];
52     }
```

```
53      ////////////////////////////
54      while(t--){
55      int index=rand()%lines;
56      start=clock();
57      for(int a=0;a<lines;++a)
58      {
59          arr[a]=brr[a];
60      }
61      quick_sort_in_place(arr,0,lines-1,lines);
62      int p=arr[index];
63      finish=clock();
64      long t0=finish-start;
65      temp0+=t0;
66      ////////////////////////////
67      start=clock();
68      for(int a=0;a<lines;++a)
69      {
70          arr[a]=brr[a];
71      }
72      Rselect(arr,lines,index);
73      finish=clock();
74      long t1=finish-start;
75      temp1+=t1;
76      ////////////////////////////
77      start=clock();
78      for(int a=0;a<lines;++a)
79      {
80          arr[a]=brr[a];
```

```
81      }
82      Dselect(arr,lines,index);
83      finish=clock();
84      long t2=finish-start;
85      temp2+=t2;
86      ////////////////////////////
87      time0+=temp0/1;
88      time1+=temp1/1;
89      time2+=temp2/1;
90      }
91      cout<<time0/10<<endl;
92      cout<<time1/10<<endl;
93      cout<<time2/10<<endl;
94      return 0;
95  }
96  ////////////////////////////
97  ////////////////////////////
98  int Rselect(int *arr,int n,int i)
99  {
100     if(n==1)
101     {
102         return *arr;
103     }
104     int pivotat=rand()%n;
105     pivotat=partition(arr,0,n-1,n,pivotat);
106     if(pivotat==i)
107     {
108         return arr[pivotat];
```

```
109     }
110     else if(pivotat>i)
111     {
112         return Rselect(arr,pivotat,i);
113     }
114     else
115     {
116         return Rselect(arr+pivotat+1,n-pivotat-1,i-pivotat-1);
117     }
118 }
119 ////////////////////////////
120 ////////////////////////////
121 int Dselect(int *arr,int n,int i)
122 {
123     if(n<=1)
124     {
125         return *arr;
126     }
127     int group_num=n/5;
128     int last=n%5;
129     if(last==0)
130     {
131         last=5;
132     }
133     else
134     {
135         group_num++;
136     }
```

```
137     int *C=new int[group_num];
138     int *temp=new int[5];
139     int size=0;
140     for(int p=0;p<group_num;++p)
141     {
142         if(p==group_num-1)
143         {
144             size=last;
145         }
146         else
147         {
148             size=5;
149         }
150         for(int q=0;q<size;++q)
151         {
152             temp[q]=arr[q+5*p];
153         }
154         selection_sort(temp,size);
155         C[p]=temp[size/2];
156     }
157     int pivot=Dselect(C,n/5,n/10);
158     delete [] C;
159     delete [] temp;
160     int pivotat=0;
161     for(int p=0;p<n;++p)
162     {
163         if(arr[p]==pivot)
164         {
```

```
165             pivotat=p;
166         }
167     }
168     pivotat=partition(arr,0,n-1,n,pivotat);
169     if(pivotat==i)
170     {
171         return arr[pivotat];
172     }
173     else if(pivotat>i)
174     {
175         return Dselect(arr,pivotat,i);
176     }
177     else
178     {
179         return Dselect(arr+pivotat+1,n-pivotat-1,i-pivotat-1);
180     }
181 }
182 ////////////////////////////
183 ////////////////////////////
184 int partition(int *arr,int left,int right,int size,int pivotat)
185 {
186     swap(arr[0],arr[pivotat]);
187     int i=1;
188     int j=size-1;
189     while(true)
190     {
191         while(i<size-1&&arr[i]<arr[0])
192         {
```

```
193             ++i;
194         }
195         while(j>0&&arr[j]>=arr[0])
196         {
197             --j;
198         }
199         if(i<j)
200         {
201             swap(arr[i],arr[j]);
202         }
203         else
204         {
205             break;
206         }
207     }
208     swap(arr[0],arr[j]);
209     return j;
210 }
211 ////////////////////////////
212 ////////////////////////////
213 void selection_sort(int *arr,int n)
214 {
215     for(int i=0;i<n-1;++i)
216     {
217         int index=i;
218         for(int j=i+1;j<n;++j)
219         {
220             if(arr[j]<arr[index])
```

```
221             {
222                 index=j;
223             }
224         }
225         if(index!=i)
226         {
227             int tmp=arr[index];
228             arr[index]=arr[i];
229             arr[i]=tmp;
230         }
231     }
232 }
233
234 void quick_sort_in_place(int *arr,int left,int right,int size)
235 {
236     if(size==0)
237     {
238         return;
239     }
240     int pivotat=rand()%size;
241     if(left>=right)
242     {
243         return;
244     }
245     pivotat=partition(arr,left,right,size,pivotat);
246     quick_sort_in_place(arr,left,pivotat-1,pivotat-left);
247     quick_sort_in_place(arr+pivotat+1,0,right-pivotat-1,right-pivotat);
248 }
```

3.3 Visualization

```
clear all;clc;
t0=[7 57 983 4701 12018 54800 106287 1180554];
t1=[3 15 191 832 1944 6622 12056 113349];
t2=[9 69 1089 3921 8860 32207 61333 550122];
size=[10 100 1000 5000 10000 50000 100000 1000000];
plot(size,t0,'o-',size,t1,'v-',size,t2,'*-');
xlabel('array size');
ylabel('speed^{-1}');
legend('quick\_sort\_in\_place','rselection\_sort','dselection\_sort');
```