UM–SJTU Joint Institute

Data Structures and Algorithms

(VE281)

Homework 2

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

Contents

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 run the program 10 times 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 sorting should meet the identical array. Otherwise, you will see that insertion sort will have a leading performance no matter how large the size is.

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	1	1	2
100	17	4	16
1000	154	24	152
5000	923	158	749
10000	2053	355	1498
50000	9853	1243	6046
100000	19602	2004	11323
1000000	206184	25104	100056

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.

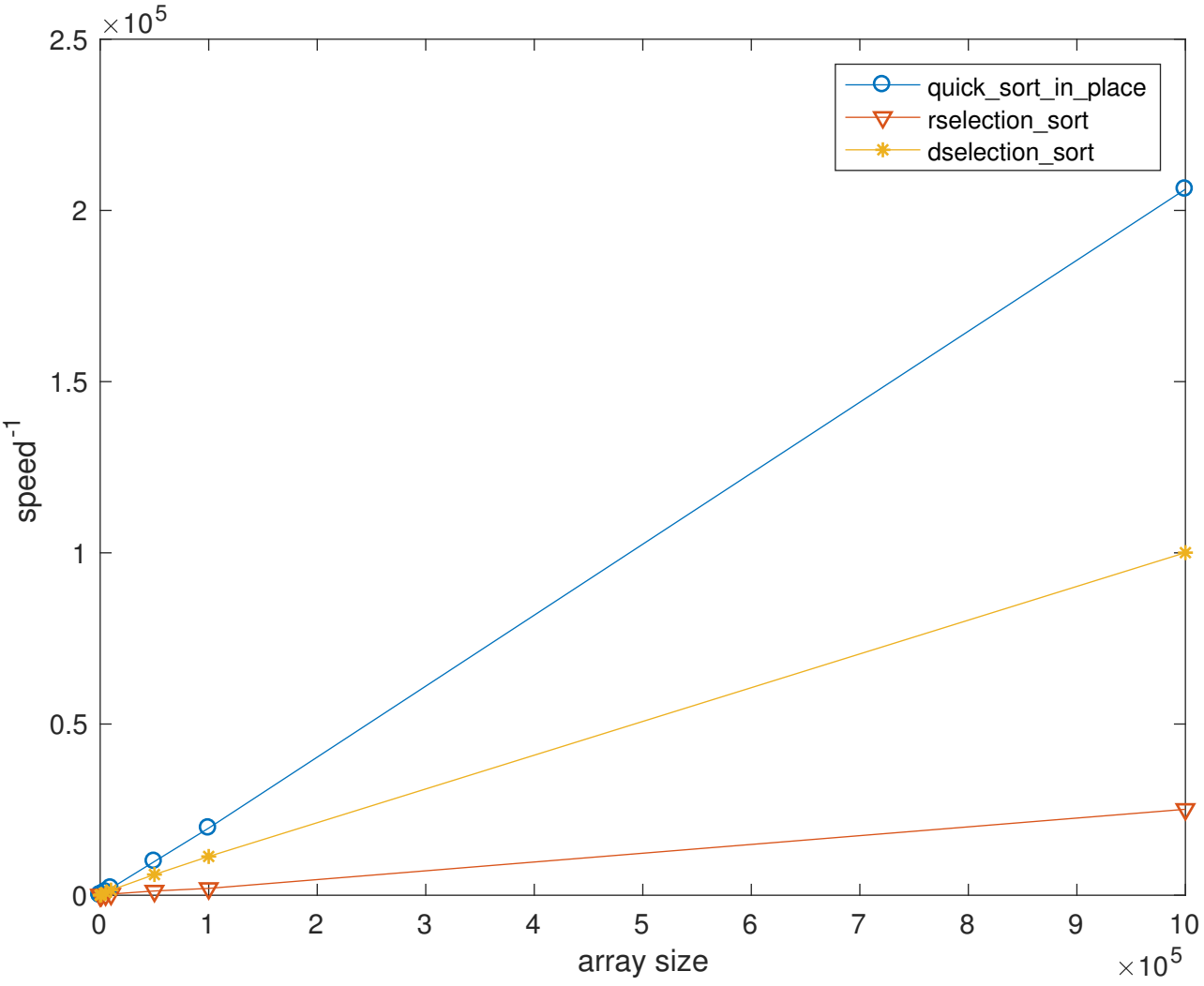

Figure 1: Run time comparison

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

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 index=rand()%lines;
36     long temp0=0;
37     long temp1=0;
38     long temp2=0;
39     clock_t start,finish;
40     int arr[lines];
41     for(int i=0;i<lines;++i)
42     {
43         arr[i]=rand();
44     }
45     int brr[lines];
46     for(int j=0;j<lines;++j)
47     {
48         brr[j]=arr[j];
49     }
50     int k=0;
51     while(k<10)
52     {
```

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

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

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

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

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

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

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

3.3 Visualization

```
clear all;clc;
t0=[1 17 154 923 2053 9853 19602 206184];
t1=[1 4 24 158 355 1243 2004 25104];
t2=[2 16 152 749 1498 6046 11323 100056];
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');
```