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.

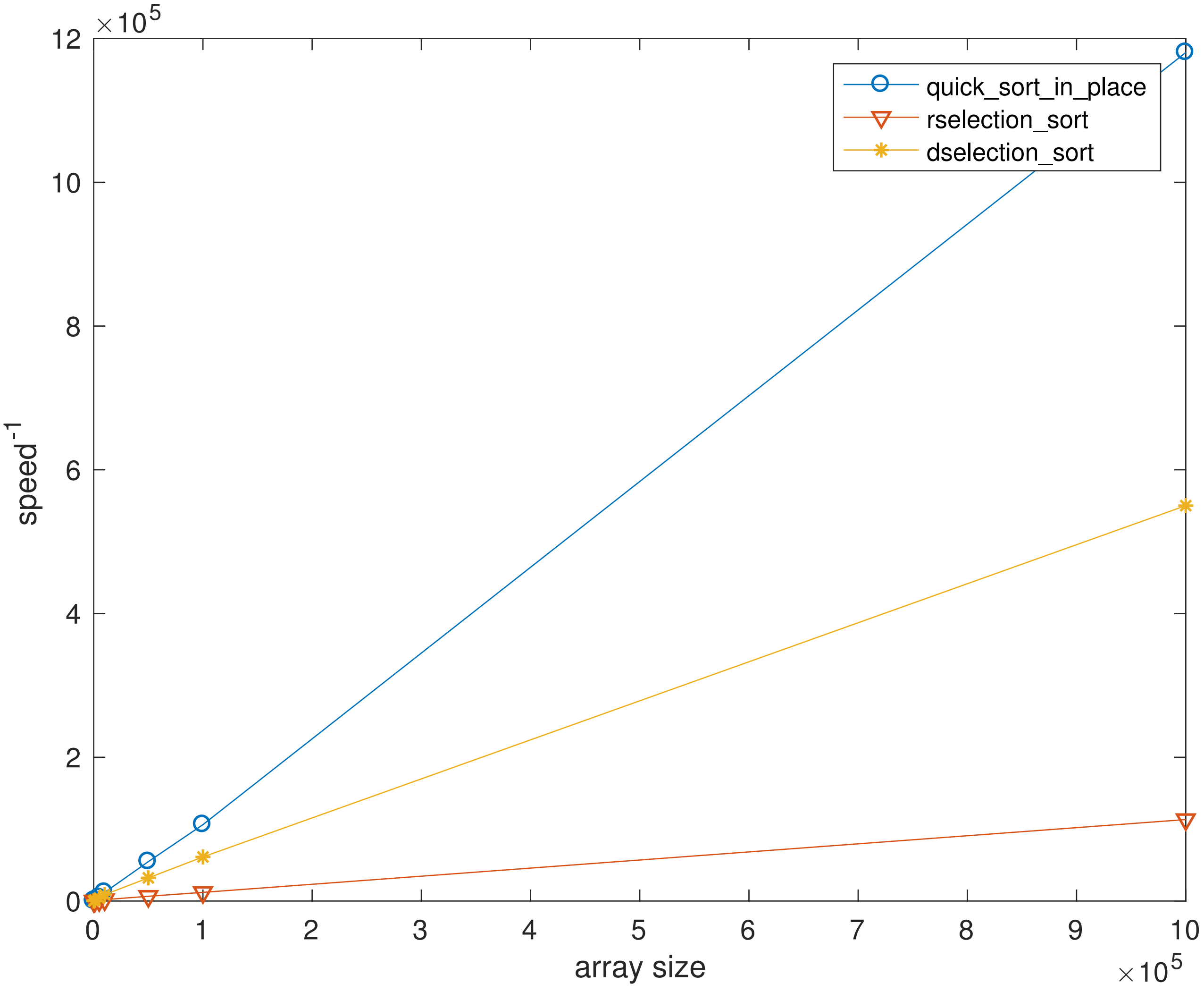

Figure 1: Run time comparison

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

```
//
//  main.cpp
//  project2
//
//  Created by            on 2017/9/27.
//  Copyright    2017              . All rights reserved.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstdlib>
#include <climits>
#include <ctime>
#include <cassert>

using namespace std;

int Rselect(int *arr,int n,int i);

int Dselect(int *arr,int n,int i);

int partition(int *arr,int left,int right,int size,int pivotat);

```

```
void selection_sort(int *arr,int n);
//////////////////////////////
//////////////////////////////
int main(int argc, const char * argv[])
{
    int choice;
    cin>>choice;
    int n;
    cin>>n;
    if(n==0)
    {
        return 0;
    }
    int i;
    cin>>i;
    if(i>=n||i<0)
    {
        return 0;
    }
    int a[n];
    for(int t=0;t<n;++t)
    {
        cin>>a[t];
    }
    int *arr=a;
    int result=0;
    switch(choice)
    {
```

```cpp
        case 0:
            result=Rselect(arr,n,i);
            break;
        case 1:
            result=Dselect(arr,n,i);
        default:
            break;
    }
    cout<<"The order-"<<i<<" item is "<<result<<endl;
//    srand(time(0));
//    int times=100;
//    while(times--)
//    {
//        int size=100;
//        int index=rand()%size;
//        int *arr=new int [size];
//        int brr[size];
//        for(int i=0;i<size;++i)
//        {
//            arr[i]=rand()%size;
//        }
//        for(int i=0;i<size;++i)
//        {
//            brr[i]=arr[i];
//        }
//        sort(arr,arr+size);
//        int p=arr[index];
//        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);
    }
}
```

```
//////////////////////////////
//////////////////////////////
int Dselect(int *arr,int n,int i)
{
    if(n<=1)
    {
        return *arr;
    }
    int group_num=n/5;
    int last=n%5;
    if(last==0)
    {
        last=5;
    }
    else
    {
        group_num++;
    }
    int *C=new int[group_num];
    int *temp=new int[5];
    int size=0;
    for(int p=0;p<group_num;++p)
    {
        if(p==group_num-1)
        {
            size=last;
        }
        else
        {
            size=5;
        }
        for(int q=0;q<size;++q)
        {
            temp[q]=arr[q+5*p];
        }
        selection_sort(temp,size);
        C[p]=temp[size/2];
    }
    int pivot=Dselect(C,n/5,n/10);
    delete [] C;
    delete [] temp;
    int pivotat=0;
    for(int p=0;p<n;++p)
    {
        if(arr[p]==pivot)
        {
            pivotat=p;
        }
    }
    pivotat=partition(arr,0,n-1,n,pivotat);
    if(pivotat==i)
    {
        return arr[pivotat];
    }
    else if(pivotat>i)
    {
        return Dselect(arr,pivotat,i);
    }
    else
    {
        return Dselect(arr+pivotat+1,n-pivotat-1,i-pivotat-1);
    }
}
//////////////////////////////
//////////////////////////////
int partition(int *arr,int left,int right,int size,int pivotat)
{
    swap(arr[0],arr[pivotat]);
    int i=1;
    int j=size-1;
    while(true)
    {
        while(i<size-1&&arr[i]<arr[0])
        {
            ++i;
        }
        while(j>0&&arr[j]>=arr[0])
        {
            --j;
        }
        if(i<j)
        {
            swap(arr[i],arr[j]);
        }
        else
        {
            break;
        }
    }
    swap(arr[0],arr[j]);
    return j;
}
//////////////////////////////
//////////////////////////////
void selection_sort(int *arr,int n)
{
    for(int i=0;i<n-1;++i)
    {
        int index=i;
        for(int j=i+1;j<n;++j)
        {
            if(arr[j]<arr[index])
            {
                index=j;
            }
        }
        if(index!=i)
        {
            int tmp=arr[index];
            arr[index]=arr[i];
            arr[i]=tmp;
        }
    }
}
```

3.2 Run-time calculation

```
//
//  main.cpp
//  run_time_study
//
//  Created by           on 2017/10/9.
//  Copyright    2017              . All rights reserved.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstdlib>
#include <climits>
#include <ctime>
#include <cassert>

using namespace std;

int Rselect(int *arr,int n,int i);

int Dselect(int *arr,int n,int i);

int partition(int *arr,int left,int right,int size,int pivotat);
```

```

void selection_sort(int *arr,int n);

void quick_sort_in_place(int *arr,int left,int right,int size);
//////////////////////////////
//////////////////////////////
int main(int argc, const char * argv[])
{
    int lines=1000000;
    srand(time(0));
    int t=10;
    long temp0=0;
    long temp1=0;
    long temp2=0;
    long time0=0;
    long time1=0;
    long time2=0;
    clock_t start,finish;
    int arr[lines];
    for(int i=0;i<lines;++i)
    {
        arr[i]=rand();
    }
    int brr[lines];
    for(int j=0;j<lines;++j)
    {
        brr[j]=arr[j];
    }
```

```
//////////////////////////////
while(t--){
int index=rand()%lines;
start=clock();
for(int a=0;a<lines;++a)
{
    arr[a]=brr[a];
}
quick_sort_in_place(arr,0,lines-1,lines);
int p=arr[index];
finish=clock();
long t0=finish-start;
temp0+=t0;
//////////////////////////////
start=clock();
for(int a=0;a<lines;++a)
{
    arr[a]=brr[a];
}
Rselect(arr,lines,index);
finish=clock();
long t1=finish-start;
temp1+=t1;
//////////////////////////////
start=clock();
for(int a=0;a<lines;++a)
{
    arr[a]=brr[a];
    }
    Dselect(arr,lines,index);
    finish=clock();
    long t2=finish-start;
    temp2+=t2;
    //////////////////////////////
    time0+=temp0/1;
    time1+=temp1/1;
    time2+=temp2/1;
    }
    cout<<time0/10<<endl;
    cout<<time1/10<<endl;
    cout<<time2/10<<endl;
    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);
    }
}
//////////////////////////////
//////////////////////////////
int Dselect(int *arr,int n,int i)
{
    if(n<=1)
    {
        return *arr;
    }
    int group_num=n/5;
    int last=n%5;
    if(last==0)
    {
        last=5;
    }
    else
    {
        group_num++;
    }
    int *C=new int[group_num];
    int *temp=new int[5];
    int size=0;
    for(int p=0;p<group_num;++p)
    {
        if(p==group_num-1)
        {
            size=last;
        }
        else
        {
            size=5;
        }
        for(int q=0;q<size;++q)
        {
            temp[q]=arr[q+5*p];
        }
        selection_sort(temp,size);
        C[p]=temp[size/2];
    }
    int pivot=Dselect(C,n/5,n/10);
    delete [] C;
    delete [] temp;
    int pivotat=0;
    for(int p=0;p<n;++p)
    {
        if(arr[p]==pivot)
        {
```

```cpp
            pivotat=p;
        }
    }
    pivotat=partition(arr,0,n-1,n,pivotat);
    if(pivotat==i)
    {
        return arr[pivotat];
    }
    else if(pivotat>i)
    {
        return Dselect(arr,pivotat,i);
    }
    else
    {
        return Dselect(arr+pivotat+1,n-pivotat-1,i-pivotat-1);
    }
}
//////////////////////////////
//////////////////////////////
int partition(int *arr,int left,int right,int size,int pivotat)
{
    swap(arr[0],arr[pivotat]);
    int i=1;
    int j=size-1;
    while(true)
    {
        while(i<size-1&&arr[i]<arr[0])
        {
            ++i;
        }
        while(j>0&&arr[j]>=arr[0])
        {
            --j;
        }
        if(i<j)
        {
            swap(arr[i],arr[j]);
        }
        else
        {
            break;
        }
    }
    swap(arr[0],arr[j]);
    return j;
}
//////////////////////////////
//////////////////////////////
void selection_sort(int *arr,int n)
{
    for(int i=0;i<n-1;++i)
    {
        int index=i;
        for(int j=i+1;j<n;++j)
        {
            if(arr[j]<arr[index])
            {
                index=j;
            }
        }
        if(index!=i)
        {
            int tmp=arr[index];
            arr[index]=arr[i];
            arr[i]=tmp;
        }
    }
}

void quick_sort_in_place(int *arr,int left,int right,int size)
{
    if(size==0)
    {
        return;
    }
    int pivotat=rand()%size;
    if(left>=right)
    {
        return;
    }
    pivotat=partition(arr,left,right,size,pivotat);
    quick_sort_in_place(arr,left,pivotat-1,pivotat-left);
    quick_sort_in_place(arr+pivotat+1,0,right-pivotat-1,right-pivotat);
}
```

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');
```