DESIGN AND ANALYSIS OF ALGORITHMS – CSCI 612

NAME: AADITHYA GOPALAKRISHNA BHARADWAJ

ZID: Z1862641

ASSIGNEMENT NUMBER, SEMESTER: Assignment 1, SPRING 2019

DUE DATE: 01/29/2019

CONTENT

1.) PROBLEM STATEMENT:

To Implement Insertion sort and Selection sort algorithms and study the impact of the pattern of input data on time cost of the algorithms. Following are the further instruction on how the program should be constructed:

a.) The driver program takes up to three command line arguments in form of "[N=n] [S=R/A/D] G=I/S". The first argument specifies the size of the sorting task, i.e., the number of integers. You will use sizes of 100, 1000, 10000, 100000, 1000000, … etc. to test your program. The second argument specifies the pattern of data to be sorted: 'A' indicates ascending order; 'D' indicates descending order; 'R' indicates random data. The third argument specifies the algorithm invoked when the program executes. "[]" indicates an argument is optional. By default, N is 1000, S is 'R'. For example, invoking the following command./your-program N=10000 S=A G=I will generate 10000 ascending sorted integers to be initial data and use insertion sort algorithm to sort the data. And the command ./your-program G=S will generate 1000 random integers and use selection sort algorithm to sort the data.

b.) Preparation of your initial data: Generate the specified number of random integers if the second command line argument is "S=R" or there is no argument on this. You can use library routine(s) for this step. If the argument is "S=A", use a simple approach to produce the specified number of integers in ascending order, e.g., in sequence of "1, 2, 3, 4, 5, 6, …" Do similarly for the argument "S=D'.

c.) Implement the two sorting algorithms as separate routines. Invoke one of them based on the command line argument "G=I/S".

d.) When running your program, use the time command to record time spent on running the program.

2.) PROGRAMMING LANGUAGE USED AND COMMANDS USED TO COMPILE AND EXECUTE THE CODE:

a.) I used C++ Programming Language to implement the insertion and selection sort algorithm.

b.) Used the Command g++ <program name.cpp> to compile the program.

c.) Used the Command time ./a.out N= <No. of Arrya Elements> S= <Ascending/Descending> G=<Insertion/Selection technique> to execute the code.

3.) SYSTEM INFORMATION WHERE CODE WAS EXECUTED:

a.) The code was executed in Ubuntu 18.04. The System Specification is as follows:

1.) **Processor:** Intel core i5 7th Generation, Quad Core Processor

2.) **RAM:** 8 GB Ram

3.) **Hard Disk Size:** 1 TB

4.) SOURCE CODE:

```
#include <iostream>
#include <string.h>
#include <sstream>
#include <time.h>
using namespace std;
void printArray(int arr[], int n)
{
   int i;
   for (i=0; i<n; i++)
      cout<<arr[i]<<" ";
   cout<<"\n";
}
int insertionSort( int arr[], int n)  //Function to sort using Insertion Sort
{
   int counter=0;  // Just an addition to the program to calculate the number of comparisons it to
took to sort the array elements
   cout<<"Insertion Sort"<<endl;
   int i, key, j;
   for (i=1; i<n; i++)
   {
      key=arr[i];
      j=i-1;
      while(j>=0 && arr[j]>key)
      {
         arr[j+1]=arr[j];
         j=j-1;
         counter++; // Just an addition to the program to calculate the number of comparisons it to
took to sort the array elements
      }
      arr[j+1]=key;
   }
   return counter;
}
void swap(int *x, int *y)
{
   int temp=*x;
   *x=*y;
   *y=temp;
}
int selectionSort(int arr[], int n)  //Function to sort using Selection Sort
{
   double counter=0;
   cout<<"Selection Sort"<<endl;
   int i, j, min_idx;
   for (i=0;i<n-1;i++)
```

```
    {
        min_idx=i;
        for (j=i+1;j<n;j++)
        {
            counter++;
            if(arr[j]<arr[min_idx])
                min_idx=j;
        }
        swap(&arr[min_idx], &arr[i]);
    }
    return counter;
}
int main(int argc, char** argv)
{
        cout<<"Number of arguments"<<argc<<"\n";
        int Num=1000;
        char S='R';
        char G;
        string temp;
        for (int i=0; i<argc; ++i){
                if(argv[i][0]=='N')
                {
                        temp=argv[i];
                        string N=temp.substr(2);
                        //To convert this string to integer
                        stringstream temp1(N);
                        temp1>>Num;
                }
                if (argv[i][0]=='S')
                {
                        S=argv[i][2];
                }
                if (argv[i][0]=='G')
                {
                        G=argv[i][2];
                }
        }
//Initialize and populate our array
  int arr[Num]={};
  if(S=='A')                //for ascending order
  {
     for (int i=0; i<Num; i++)
     {
        arr[i]=i;
     }
  }
```

```
 if (S=='D')              //for descendinging order
 {
    int j=Num-1;
    for (int i=0; i<Num; i++)
    {
       arr[i]=j; j--;
    }
 }
 if (S=='R')              //for random numbers
 {
    for(int i=0; i<Num; i++)
    {
       arr[i]=rand()%100000000001;
    }
 }
 printArray(arr, Num);
 //Sorting
 int c=0;
 //clock_t t;             //For traditional method of calculating time.
 if (G=='I')   //If the sorting method is selected as Insertion
 {
    //t=clock();                  //For traditional method of calculating time.
    c=insertionSort(arr, Num);
    //t=clock()-t;        //For traditional method of calculating time.
 }
 if (G=='S')     // //If the sorting method is selected as Insertion
 {
    //t=clock();                  //For traditional method of calculating time.
    c=selectionSort(arr, Num);
    //t=clock()-t;        //For traditional method of calculating time.
 }
  //double time_taken=((double) t)/CLOCKS_PER_SEC; //For traditional method of calculating
time.
   printArray(arr, Num);
   cout<<"Number of comparisons="<<c<<endl;// Just an additional command to calculate the
number of comparisons it to took to sort the array elements
   //cout<<"Time Taken="<<time_taken<<endl;//For traditional method of calculating time.
}
```

5.) FINDINGS:

S.NO	No. OF ELEMENTS IN THE ARRAY	TECHNIQUE	ARRAY ORDER	REAL TIME	USER TIME	SYSTEM TIME
1	1000	INSERTION	ASCENDING	0m 0.0042s	0m 0.0041s	0m 0s
2	10000	INSERTION	ASCENDING	0m 0.0285s	0m 0.0113s	0m 0.002s
3	100000	INSERTION	ASCENDING	0m 0.2018s	0m 0.0395s	0m 0.044s
4	1000000	INSERTION	ASCENDING	0m 1.4813s	0m 1.2221s	0m 0.072s
5	1100000	INSERTION	ASCENDING	0m 1.6247s	0m 1.5015s	0m 0.081s
6	1000	INSERTION	DESCENDING	0m 0.0125s	0m 0.008s	0m 0.004s
7	10000	INSERTION	DESCENDING	0m 0.0174s	0m 0.0166s	0m 0.008s
8	100000	INSERTION	DESCENDING	0m 14.259s	0m 14.098s	0m 0.014s
9	300000	INSERTION	DESCENDING	2m 5.865s	2m 5.483s	0m 0.028s
10	600000	INSERTION	DESCENDING	3m 3.325s	3m 45.25s	0m 1.256s
11	1000	INSERTION	RANDOM	0m 0.0125s	0m 0.0055s	0m 0.004s
12	10000	INSERTION	RANDOM	0m 0.165s	0m 0.1195s	0m 0.008s
13	100000	INSERTION	RANDOM	0m 7.675s	0m 7.395s	0m 0.031s
14	300000	INSERTION	RANDOM	1m 6.265s	1m 5.436s	0m 0.0465s
15	600000	INSERTION	RANDOM	4m 18.375s	4m 17.14s	0m 2.08s
16	1000	SELECTION	ASCENDING	0m 0.0125s	0m 0.008s	0m 0.004s
17	10000	SELECTION	ASCENDING	0m 0.187s	0m 0.176s	0m 0.008s
18	100000	SELECTION	ASCENDING	0m 15.75s	0m 15.58s	0m 0.215s
19	300000	SELECTION	ASCENDING	2m 17.64s	2m 17.13s	0m 0.76s
20	600000	SELECTION	ASCENDING	5m 32.42s	5m 12.46s	0m 1.25s
21	1000	SELECTION	DESCENDING	0m 0.0115s	0m 0.008s	0m 0.003s
22	10000	SELECTION	DESCENDING	0m 0.2155s	0m 0.201s	0m 0.021s
23	100000	SELECTION	DESCENDING	0m 15.321s	0m 15.11s	0m 0.045s
24	300000	SELECTION	DESCENDING	2m 15.544s	2m 13.978s	0m 0.065s
25	600000	SELECTION	DESCENDING	7m 14.771s	7m 13.25s	0m 2.15s
26	1000	SELECTION	RANDOM	0m 0.0135s	0m 0.009s	0m 0.005s
27	10000	SELECTION	RANDOM	0m 0.251s	0m 0.0233s	0m 0.012s
28	100000	SELECTION	RANDOM	0m 15.404s	0m 15.133s	0m 0.026s
29	300000	SELECTION	RANDOM	2m 18.666s	2m 17.765s	0m 0.081s
30	600000	SELECTION	RANDOM	8m 23.429s	8m 21.12s	0m 2.45s

Table 1: Time taken for sorting

The time complexity for **insertion sort** is:

- **Worst Case:** $O(n^2)$
- **Best Case:** $\Omega(n)$

We **cannot say insertion time cost is $\Theta(n^2)$** but **we can say worst case insertion sort time cost is $\Theta(n^2)$**

The time complexity for **selection sort is same for best case and worst case** and it is $O(n^2)$, $\Theta(n^2)$ and $\Omega(n^2)$.

For small array (less that 20-30 elements), both insertion and selection sort are typically faster than the $O(n*\log n)$. But usually **insertion sort will perform less comparisons than the selection sort when the array elements increases.** This we can prove with the number of comparison code I have written in the program where the insertion sort for same array took less comparison compared to selection sort and also, we can conclude from Table 1 the time taken by insertion sort is less than that of selection sort.

Insertion sort is also more stable than selection sort. Insertion sort is also In-Place as in int only requires a constant amount of O(1) of additional memory space and it can sort a list as it receives it.

From Table 1 we also inference that if the array elements are in **Ascending order then the time taken to sort be it Insertion Sort or Selection Sort is faster compared to that of Descending and Random values** and from the array elements, I compared with the Random values took more time to sort

We can also safely conclude that Real Time value is greater than the User Time value for any array elements and that the System takes the least amount of time to sort the elements.

Real Time > User Time > System Time

Note:

I have attached the Source Code along with the PDF file of this report to Blackboard. I couldn't login to hopper.cs.niu.edu as the password I tried to enter said is invalid, hence executed in my personal system.