DAA Lab-5

Name: Kshitij Kumar Sharma Roll No.: 1905514 Date: 07/08/2021

Q-5.1) Write a program to sort a given set of elements with the Quick sort.

1)Repeat the experiment for different values of n = 5000, 10000, 50000, 100000 and report the time (in seconds) required to sort the elements.

2)For each of aforementioned case, consider arrays as random, sorted, and reverse-sorted and observe running time variation for different types of input for quick sort. [Provide your observation regarding sensitivity of quick sort on the input in your lab record.]

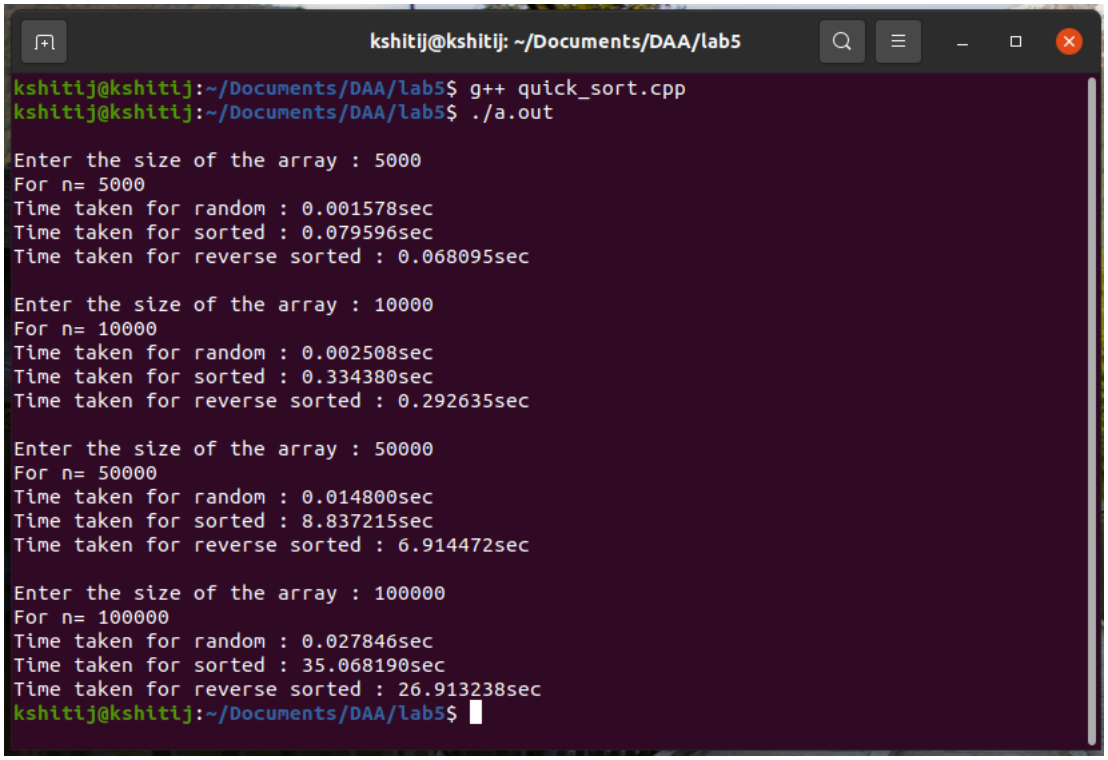
Program:

```
/*
Written by: Kshitij Kumar Sharma                    Roll No.: 1905514
Idea of the solution:
        At first I have implemented the quick_sort() and partition() functions and then generated
random, sorted and reverse sorted array to pass into the quick_sort() function and stored their
timings separately. The partition function divides the array into two halfs with respect to a value
called pivot which is the last element of the array, the left half contains all the values which are
smaller than that of pivot and the right half contains all the grater than the pivot value and it returns
the index of the pivot element. The quick_sort() function calls the partition() function, after getting
the index of the pivot element the quick_sort() function recursively calls itself for the left half and
right half of the pivot and it does so until single element is left in the left and right sub arrays.
*/
#include<bits/stdc++.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
using namespace std;

int partition(int a[],int p,int r)
        {
                int j,t,i=p-1;                          //latest index of elements less than pivot
                int pivot=a[r];                         //assigning last element as pivot

                for(j=p;j<=r-1;j++)
                        {
                                if(a[j]<=pivot)         //checking for element smaller than pivot
                                        {
                                                i++;
                                                t=a[i];         //swapping the element
                                                a[i]=a[j];
                                                a[j]=t;
                                        }
```

```
		}
		t=a[i+1];		//putting pivot element at its position
		a[i+1]=a[r];
		a[r]=t;
		return i+1;
	}

void quick_sort(int a[],int p,int r)
	{
		int q;
		if(p<r)
			{
				q=partition(a,p,r);		//calling partition() for getting pivot's index
				quick_sort(a,p,q-1);		//recursively sending left sub half
				quick_sort(a,q+1,r);		//recursively sending right sub half
			}
	}

int main()
	{
		int n,i,j,k;
		clock_t start, end;		//Variables for keeping start and end time
		double cpu_time_used;		//Variable for keeping cpu time used
		for(i=0;i<4;i++)
		{
		cout<<endl;
		cout<<"Enter the size of the array : ";
		cin>>n;
		int a[n];
		srand(time(0));
		for(j=0;j<n;j++)
			a[j]=rand()%1000000;		//generating random array
		cout<<"For n= "<<n<<endl;
		start=clock();		//keeping start time of the clock for random array
		quick_sort(a,0,n-1);		//sorting
		end=clock();		//keeping end time of the clock for random array

		cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
		printf("Time taken for random : %fsec \n",cpu_time_used);

		sort(a,a+n);

		start=clock();		//keeping start time of the clock for sorted array
		quick_sort(a,0,n-1);		//sorting
```

```
        end=clock();                    //keeping end time of the clock for sorted array

        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Time taken for sorted : %fsec \n",cpu_time_used);

        sort(a,a+n,greater<int>());

        start=clock();                  //keeping start time of the clock for reverse sorted array
        quick_sort(a,0,n-1);            //sorting
        end=clock();                    //keeping end time of the clock for reverse sorted array

        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Time taken for reverse sorted : %fsec \n",cpu_time_used);
        }
}
```

Outputs:



```
kshitij@kshitij: ~/Documents/DAA/lab5
kshitij@kshitij:~/Documents/DAA/lab5$ g++ quick_sort.cpp
kshitij@kshitij:~/Documents/DAA/lab5$ ./a.out

Enter the size of the array : 5000
For n= 5000
Time taken for random : 0.001146sec
Time taken for sorted : 0.085650sec
Time taken for reverse sorted : 0.076272sec

Enter the size of the array : 10000
For n= 10000
Time taken for random : 0.002460sec
Time taken for sorted : 0.335991sec
Time taken for reverse sorted : 0.278444sec

Enter the size of the array : 50000
For n= 50000
Time taken for random : 0.013313sec
Time taken for sorted : 8.180964sec
Time taken for reverse sorted : 6.822692sec

Enter the size of the array : 100000
For n= 100000
Time taken for random : 0.028411sec
Time taken for sorted : 33.737986sec
Time taken for reverse sorted : 27.292968sec
kshitij@kshitij:~/Documents/DAA/lab5$
```

Q-5.2) Repeat implementation of 5.1 with a randomized version of quick sort in which partition selects the pivot element randomly. Compare the previous version with randomized quick sort.

Program:

```
/*
Written by: Kshitij Kumar Sharma                    Roll No.: 1905514
Idea of the solution:
        At first I have implemented the random_quick_sort(), random_partition() and partition() functions and then generated random, sorted and reverse sorted array to pass into the random_quick_sort() function and stored their timings separately. The partition function divides the array into two halfs with respect to a value called pivot which is the last element of the array, the left half contains all the values which are smaller than that of pivot and the right half contains all the grater than the pivot value and it returns the index of the pivot element. The random_partition() function generates a random index value, in between p and r and then swaps that index value with the last index value and then calls the partition() function. The random_quick_sort() function calls the random_partition() function, after getting the index of the pivot element the random_quick_sort() function recursively calls itself for the left half and right half of the pivot and it does so until single element is left in the left and right sub arrays.
*/
#include<bits/stdc++.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
using namespace std;

int partition(int a[],int p,int r)
        {
```

```
        int j,t,i=p-1;                          //latest index of elements less than pivot
        int pivot=a[r];                         //assigning last element as pivot

        for(j=p;j<=r-1;j++)
            {
                if(a[j]<=pivot)                 //checking for element smaller than pivot
                    {
                        i++;
                        t=a[i];                 //swapping the element
                        a[i]=a[j];
                        a[j]=t;
                    }
            }
            t=a[i+1];                           //putting pivot element at its position
            a[i+1]=a[r];
            a[r]=t;
            return i+1;
    }
int random_partition(int a[],int p,int r)       //for choosing random pivot
    {
        srand(time(0));
        int i=(rand()%(r-p+1))+p;               //generating random index between p and r
        int t=a[r];                             //swapping the last element with the random
        a[r]=a[i];                              // index value.
        a[i]=t;
        return partition(a,p,r);                //calling partition()
    }
void random_quick_sort(int a[],int p,int r)
    {
        int q;
        if(p<r)
            {
                q=random_partition(a,p,r);      //calling for pivot's index
                random_quick_sort(a,p,q-1);     //recursively sending left sub half
                random_quick_sort(a,q+1,r);     //recursively sending right sub half
            }
    }

int main()
    {
        int n,i,j,k;
        clock_t start, end;                     //Variables for keeping start and end time
        double cpu_time_used;                   //Variable for keeping cpu time used
        for(i=0;i<4;i++)
```

```
        {
        cout<<endl;
        cout<<"Enter the size of the array : ";
        cin>>n;
        int a[n];
        srand(time(0));
        for(j=0;j<n;j++)
                a[j]=rand()%1000000;            //generating random array
        cout<<"For n= "<<n<<endl;
        start=clock();                          //keeping start time of the clock for random array
        random_quick_sort(a,0,n-1);             //sorting
        end=clock();                            //keeping end time of the clock for random array

        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Time taken for random : %fsec \n",cpu_time_used);

        sort(a,a+n);

        start=clock();                          //keeping start time of the clock for sorted array
        random_quick_sort(a,0,n-1);             //sorting
        end=clock();                            //keeping end time of the clock for sorted array

        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Time taken for sorted : %fsec \n",cpu_time_used);

        sort(a,a+n,greater<int>());

        start=clock();                      //keeping start time of the clock for reverse sorted array
        random_quick_sort(a,0,n-1);         //sorting
        end=clock();                        //keeping end time of the clock for reverse sorted array

        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Time taken for reverse sorted : %fsec \n",cpu_time_used);
        }
}
```

Outputs:

```
kshitij@kshitij: ~/Documents/DAA/lab5
kshitij@kshitij:~/Documents/DAA/lab5$ g++ quick_sort_random.cpp
kshitij@kshitij:~/Documents/DAA/lab5$ ./a.out

Enter the size of the array : 5000
For n= 5000
Time taken for random : 0.011356sec
Time taken for sorted : 0.009888sec
Time taken for reverse sorted : 0.008764sec

Enter the size of the array : 10000
For n= 10000
Time taken for random : 0.020988sec
Time taken for sorted : 0.021303sec
Time taken for reverse sorted : 0.020152sec

Enter the size of the array : 50000
For n= 50000
Time taken for random : 0.090858sec
Time taken for sorted : 0.093256sec
Time taken for reverse sorted : 0.082902sec

Enter the size of the array : 100000
For n= 100000
Time taken for random : 0.186738sec
Time taken for sorted : 0.182639sec
Time taken for reverse sorted : 0.184604sec
kshitij@kshitij:~/Documents/DAA/lab5$
```

```
kshitij@kshitij: ~/Documents/DAA/lab5
kshitij@kshitij:~/Documents/DAA/lab5$ g++ quick_sort_random.cpp
kshitij@kshitij:~/Documents/DAA/lab5$ ./a.out

Enter the size of the array : 5000
For n= 5000
Time taken for random : 0.011145sec
Time taken for sorted : 0.009745sec
Time taken for reverse sorted : 0.009142sec

Enter the size of the array : 10000
For n= 10000
Time taken for random : 0.019986sec
Time taken for sorted : 0.017332sec
Time taken for reverse sorted : 0.018464sec

Enter the size of the array : 50000
For n= 50000
Time taken for random : 0.092772sec
Time taken for sorted : 0.088392sec
Time taken for reverse sorted : 0.086558sec

Enter the size of the array : 100000
For n= 100000
Time taken for random : 0.185500sec
Time taken for sorted : 0.170805sec
Time taken for reverse sorted : 0.170893sec
kshitij@kshitij:~/Documents/DAA/lab5$
```