DAA Lab-4

Name: Kshitij Kumar Sharma Roll No.: 1905514 Date: 30/07/2021

**Q-4.1)** Write a program to sort a given set of elements with the Merge sort.

1)Repeat the experiment for different values of n = 500, 1000,5000, 10000 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 merge sort. [Provide your observation regarding sensitivity of Merge sort on the input in your lab record.]

**Program:**

/\*

Written by: Kshitij Kumar Sharma Roll No.: 1905514

Idea of the solution:

I have first implemented the merge sort algorithm and then I created the code for random array generation and timing analysis. For implementing the merge sort algorithm I had implement merge\_sort() function and a merge() function. The merge\_sort() function breaks the array into smaller parts, let say half of its original size recursively until one-one element is left in each sub-array and then I called the merge() function to merge the single element arrays into an array and so on, to form a sorted array.

\*/

#include<bits/stdc++.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

void merge(int arr[], int l, int m, int r); //prototype for merge function

void merge\_sort(int arr[], int l, int r) //merge\_sort() function defination

{ //l=left index r=right index of the subarray

if (l<r) //checking left index less than right index

{

int m=l+(r-l)/2; //calculating the mid point

merge\_sort(arr,l,m); //recursively calling merge\_sort() for the 1st half

merge\_sort(arr,m+1,r); // recursively calling merge\_sort() for the 1st half

merge(arr,l,m,r); //calling merge() after completion of breaking of array

}

}

void merge(int arr[], int l, int m, int r) //definition of merge() function

{ //l=left index r=right index m=middle index

int i,j,k,n1,n2;

n1=m-l+1; //calculating the size of 1st sub-array

n2=r-m; // calculating the size of 2nd sub-array

int L[n1],R[n2]; //creating the two sub-arrays

for (i=0;i<n1;i++) //initialising 1st sub-array

L[i]=arr[l+i];

for (j=0;j<n2;j++) //initialising 2nd sub-array

R[j]=arr[m+1+j];

i=0;

j=0;

k=l;

while(i<n1&&j<n2) //loop for merging into an array in sorted order

{

if (L[i]<=R[j]) //checking for the larger element

{

arr[k]=L[i]; //inserting it to its position

i++;

}

else

{

arr[k]=R[j];

j++;

}

k++;

}

while(i<n1) //inserting any left-out element from 1st sub-array

{

arr[k]=L[i];

i++;

k++;

}

while(j<n2) //inserting any left-out element from 2nd sub-array

{

arr[k]=R[j];

j++;

k++;

}

}

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 the random array

cout<<"For n= "<<n<<endl;

start=clock(); //keeping start time of the clock for random array merge\_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

merge\_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

merge\_sort(a,0,n-1); //sorting

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

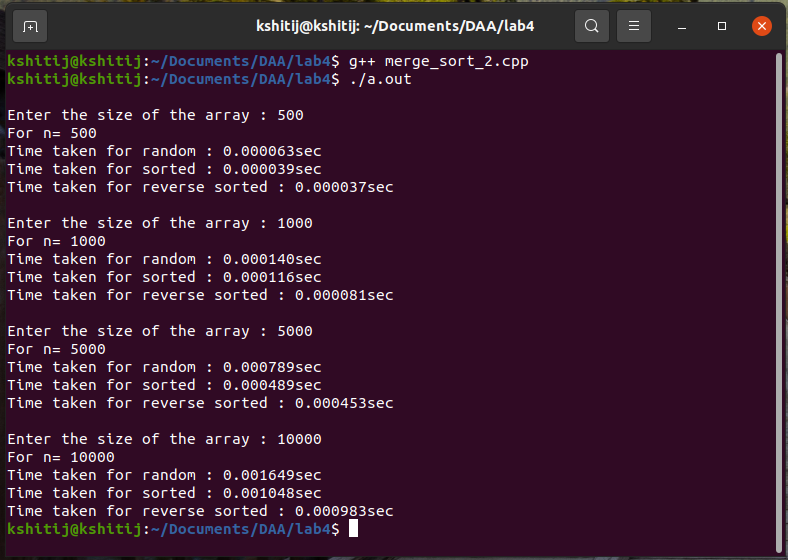
cpu\_time\_used=((double)(end-start))/CLOCKS\_PER\_SEC;

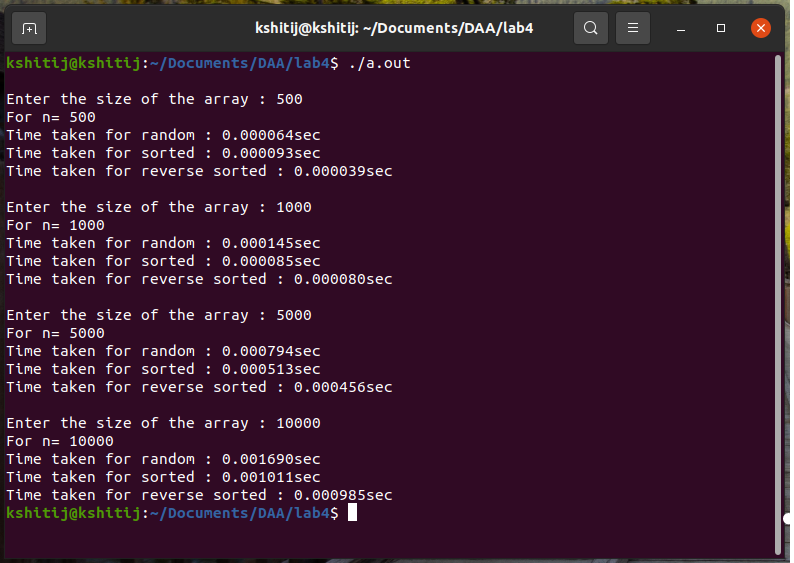
printf("Time taken for reverse sorted : %fsec \n",cpu\_time\_used);

}

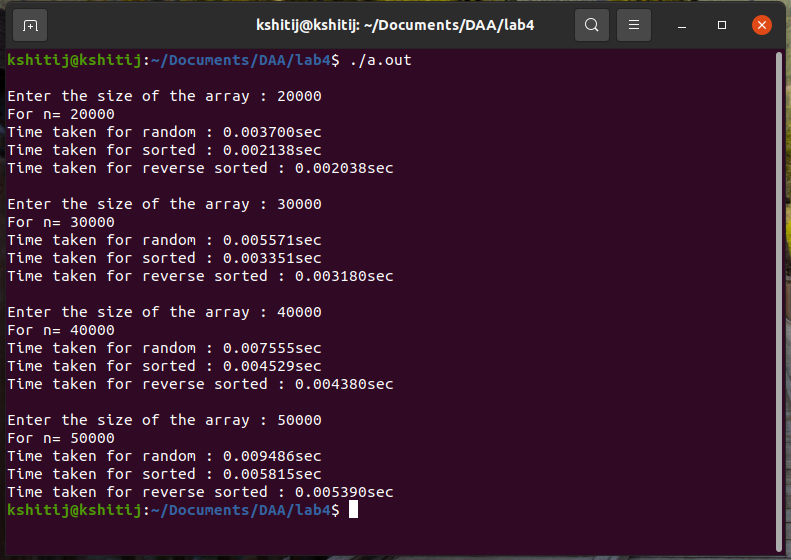
}

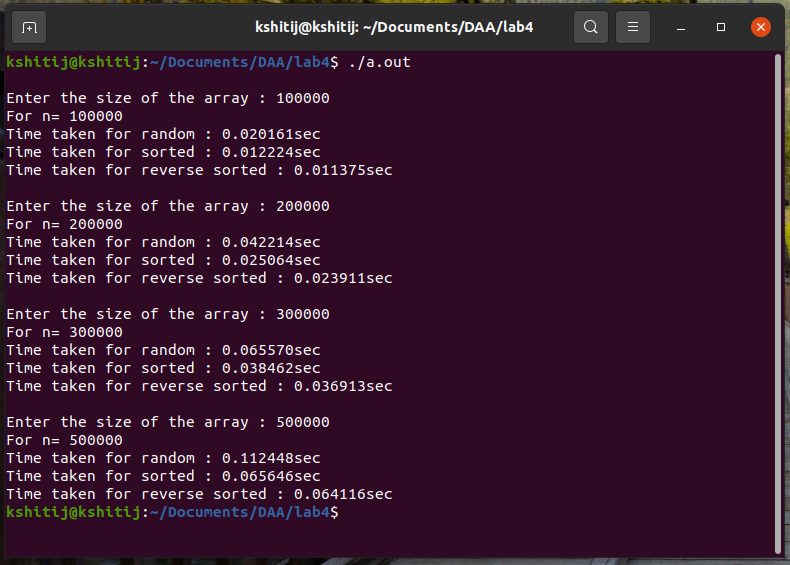
**Outputs as per asked in question:**

****

****

**Extra outputs for analysis:**

****

****

**Conclusions after analysis of outputs:**

1. The time taken for sorting a reverse sorted array is always less than the sorted element, random element array.
2. There is not much significant increase observed in time increase on increasing the array size.
3. It is faster than insertion sort algorithm.