

# 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:

```
kshiti@kshiti: ~/Documents/DAA/lab4
kshiti@kshiti:~/Documents/DAA/lab4$ g++ merge_sort_2.cpp
kshiti@kshiti:~/Documents/DAA/lab4$ ./a.out

Enter the size of the array : 500
For n= 500
Time taken for random : 0.000063sec
Time taken for sorted : 0.000039sec
Time taken for reverse sorted : 0.000037sec

Enter the size of the array : 1000
For n= 1000
Time taken for random : 0.000140sec
Time taken for sorted : 0.000116sec
Time taken for reverse sorted : 0.000081sec

Enter the size of the array : 5000
For n= 5000
Time taken for random : 0.000789sec
Time taken for sorted : 0.000489sec
Time taken for reverse sorted : 0.000453sec

Enter the size of the array : 10000
For n= 10000
Time taken for random : 0.001649sec
Time taken for sorted : 0.001048sec
Time taken for reverse sorted : 0.000983sec
kshiti@kshiti:~/Documents/DAA/lab4$
```

```
kshiti@kshiti: ~/Documents/DAA/lab4
kshiti@kshiti:~/Documents/DAA/lab4$ ./a.out

Enter the size of the array : 500
For n= 500
Time taken for random : 0.000064sec
Time taken for sorted : 0.000093sec
Time taken for reverse sorted : 0.000039sec

Enter the size of the array : 1000
For n= 1000
Time taken for random : 0.000145sec
Time taken for sorted : 0.000085sec
Time taken for reverse sorted : 0.000080sec

Enter the size of the array : 5000
For n= 5000
Time taken for random : 0.000794sec
Time taken for sorted : 0.000513sec
Time taken for reverse sorted : 0.000456sec

Enter the size of the array : 10000
For n= 10000
Time taken for random : 0.001690sec
Time taken for sorted : 0.001011sec
Time taken for reverse sorted : 0.000985sec
kshiti@kshiti:~/Documents/DAA/lab4$
```

## Extra outputs for analysis:

```
kshitij@kshitij: ~/Documents/DAA/lab4
kshitij@kshitij:~/Documents/DAA/lab4$ ./a.out
Enter the size of the array : 20000
For n= 20000
Time taken for random : 0.003700sec
Time taken for sorted : 0.002138sec
Time taken for reverse sorted : 0.002038sec

Enter the size of the array : 30000
For n= 30000
Time taken for random : 0.005571sec
Time taken for sorted : 0.003351sec
Time taken for reverse sorted : 0.003180sec

Enter the size of the array : 40000
For n= 40000
Time taken for random : 0.007555sec
Time taken for sorted : 0.004529sec
Time taken for reverse sorted : 0.004380sec

Enter the size of the array : 50000
For n= 50000
Time taken for random : 0.009486sec
Time taken for sorted : 0.005815sec
Time taken for reverse sorted : 0.005390sec
kshitij@kshitij:~/Documents/DAA/lab4$
```

```
kshitij@kshitij: ~/Documents/DAA/lab4
kshitij@kshitij:~/Documents/DAA/lab4$ ./a.out
Enter the size of the array : 100000
For n= 100000
Time taken for random : 0.020161sec
Time taken for sorted : 0.012224sec
Time taken for reverse sorted : 0.011375sec

Enter the size of the array : 200000
For n= 200000
Time taken for random : 0.042214sec
Time taken for sorted : 0.025064sec
Time taken for reverse sorted : 0.023911sec

Enter the size of the array : 300000
For n= 300000
Time taken for random : 0.065570sec
Time taken for sorted : 0.038462sec
Time taken for reverse sorted : 0.036913sec

Enter the size of the array : 500000
For n= 500000
Time taken for random : 0.112448sec
Time taken for sorted : 0.065646sec
Time taken for reverse sorted : 0.064116sec
kshitij@kshitij:~/Documents/DAA/lab4$
```

## 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.