

Experiment Number: 4

Aim: - Comparative analysis of Quick Sort and Merge Sort.

Problem Statement: - Doing the comparative analysis of Quick Sort and Merge Sort on the basis of comparisons required for sorting list of large value of n .

Theory:-

Quick Sort

Quicksort, like merge sort, is a divide-and-conquer recursive algorithm. The basic divide-and-conquer process for sorting a sub-array $S[p...r]$ is summarized in the following three easy steps:

Divide:

Partition $S[p...r]$ into two different sub-arrays $S[p...q-1]$ and $S[q+1...r]$ such that each element of $S[p...q-1]$ is less than or equal to $S[q]$, which is, in turn, less than or equal to each element of $S[q+1...r]$. Compute the index q as part of this partitioning procedure

Conquer:

Sort the two sub-arrays $S[p...q-1]$ and $S[q+1...r]$ by recursive calls to quicksort.

Combine:

Since the sub-arrays are sorted in place, no work is needed to combining them. The entire array S is now sorted.

Algorithm: -

```
quicksort(a, low, high)
{
    if ( high > low )
    {
        pivot = partition( a, low, high );
        quicksort( a, low, pivot-1 );
        quicksort( a, pivot+1, high );
    }
}

partition (a, low, high)
{
    v=a[low]; i=a[m]; j=high;
    repeat
    {
        repeat
            i = i + 1;
        until (a[i] ≥ v);
        repeat
            j = j - 1;
        until (a[j] ≤ v);
        if( i < j ) then interchange (a, i, j);
    }
    until (i ≥ j)
    a[m]=a[j]; a[j]=v; return j;
}
```

interchange (a, i, j)

```
{  
  p=a[i]; a[i]=a[j]; a[j]=p;  
}
```

While calculating the complexity for Quick Sort we count only the number of element comparisons.

The recurrence equation of the Quick Sort can be given as,

$$T(n) = T(i) + T(n - i - 1) + n$$

Best case:

In best case of the Quick Sort the pivot element always partition the array into two equal size sub-arrays. Hence we can rewrite the recurrence equation as follows,

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

If we solve above recurrence equation using Master Method then we have,

$$T(n) = O(n \log \log n)$$

Worst Case:

In the worst case the partitioning algorithm partition the array in such a way that on one of the side of array there are zero elements while on the other side of pivot there are (n - 1) elements. Hence we can rewrite the recurrence equation as follows:

$$T(n) = T(n - 1) + n$$

We can write as,

$$T(n - 1) = T(n - 2) + (n - 1)$$

$$T(n - 2) = T(n - 3) + (n - 2)$$

$$T(n - 3) = T(n - 4) + (n - 3)$$

.....

$$T(2) = T(1) + 2$$

Adding all above equations we get.

$$T(n) = T(1) + (2 + 3 + 4 + \dots + n)$$

$$\therefore T(n) = 1 + \sum_{j=2}^n j$$

$$\therefore T(n) = 1 + \frac{n(n-1)}{2}$$

$$\therefore T(n) = O(n^2)$$

Average case:

The average value of T (i) is 1/n times the sum of T (0) through T (n-1). Hence we can rewrite the recurrence equation as follows:

$$T(n) = \frac{2}{n} \left(\sum_{j=0}^{n-1} T(j) \right) + n$$

Multiplying both the sides by 'n' we get

$$nT(n) = 2 \left(\sum_{j=0}^{n-1} T(j) \right) + n^2$$

To remove the summation we rewrite the equation for (n - 1)

$$(n - 1)T(n - 1) = 2 \left(\sum_{j=0}^{n-2} T(j) \right) + (n - 1)^2$$

Subtracting above equation from the previous equation of nT(n) we get,

$$nT(n) - (n - 1)T(n - 1) = 2T(n - 1) + 2n - 1$$

$$\therefore nT(n) = (n + 1)T(n - 1) + 2n$$

Dividing both the sides by n(n+1) we get,

$$\frac{T(n)}{(n+1)} = \frac{T(n-1)}{n} + \frac{2}{n+1}$$

We can write as,

$$\therefore \frac{T(n-1)}{(n)} = \frac{T(n-2)}{(n-1)} + \frac{2}{n}$$

$$\therefore \frac{T(n-2)}{(n-1)} = \frac{T(n-3)}{(n-2)} + \frac{2}{(n-1)}$$

$$\therefore \frac{T(2)}{(3)} = \frac{T(1)}{(2)} + \frac{2}{3}$$

Adding all above terms we get,

$$\therefore \frac{T(n)}{(n+1)} = \frac{T(1)}{(2)} + 2 \sum_{j=3}^{n+1} \frac{1}{j}$$

$$\therefore T(n) = (n + 1) \left(\frac{1}{2} + \log \log n \right)$$

$$\therefore T(n) = O(n \log n)$$

Merge Sort

The merge sort algorithm is based on the classical divide and conquer paradigm. It operates as follows:

DIVIDE:

Partition the n-element array into two sub-arrays of (n/2) elements each.

CONQUER:

Sort the two sub-arrays recursively using merge sort.

COMBINE:

Merge the two sorted subsequences of size n/2 each to produce the sorted sequence consisting of n elements.

Algorithm:

```
merge_sort(a,low,high)
{
    if(high > low)
    {
        mid= (low + high)/2 ;
        merge_sort(a, low, mid);
        merge_sort(a, mid+1, high);
        merge(a, low, mid, high);
    }
}

merge (a, low, mid, high)
{
    n1 = mid-low+1;
    n2 = high-mid;
    for i = 1 to n1
        L[i] = a[low+i-1];
    for j = 1 to n2
        R[j] = a[mid+j];
    L[n1+1]= R[n2+1] = infinity;
    i = j = 1;
    for k = low to high
    {
        do if L[i] ≤ R[j]
            then a[k] = L[i]; i = i + 1;
        else
            a[k] = R[j]; j = j+1;
    }
}
```

If there is only one element in the array then the complexity of merge sort is constant and it is given by $T(1) = 1$.

If there more than 1 element in the array then the array of size 'n' is divided into sub-arrays of size 'n/2' each, and then it is sorted recursively calling merge_sort. At the last to combine two sub-arrays of size 'n/2', the complexity is linear and is given as 'n'. Hence we can write the recurrence equation for merge sort when $n > 1$ as follows:

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Dividing on both sides by 'n' we get,

$$\frac{T(n)}{n} = \left(\frac{T\left(\frac{n}{2}\right)}{\left(\frac{n}{2}\right)} \right) + 1$$

As n is in the powers of 2 i.e. $n = 2^k$ where $k=1,2,3, \dots$

We can write,

$$\frac{T\left(\frac{n}{2}\right)}{\left(\frac{n}{2}\right)} = \left(\frac{T\left(\frac{n}{4}\right)}{\left(\frac{n}{4}\right)} \right) + 1$$

$$\frac{T\left(\frac{n}{4}\right)}{\left(\frac{n}{4}\right)} = \left(\frac{T\left(\frac{n}{8}\right)}{\left(\frac{n}{8}\right)} \right) + 1$$

$$\frac{T\left(\frac{n}{8}\right)}{\left(\frac{n}{8}\right)} = \left(\frac{T\left(\frac{n}{16}\right)}{\left(\frac{n}{16}\right)} \right) + 1$$

$$\frac{T(2)}{2} = \frac{T(1)}{1} + 1$$

Adding all above equations we get,

$$\frac{T(n)}{n} = \frac{T(1)}{1} + \log \log n$$

$$\therefore T(n) = n(1 + \log \log n)$$

$$\therefore T(n) = O(n \log \log n)$$

EXPERIMENT N0-4

AIM: Comparative analysis of Quick sort and Merge sort

QUICK SORT:

CODE:

```
#include <stdio.h>

int partition(int arr[], int low, int high)
{
    int temp;
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}
```

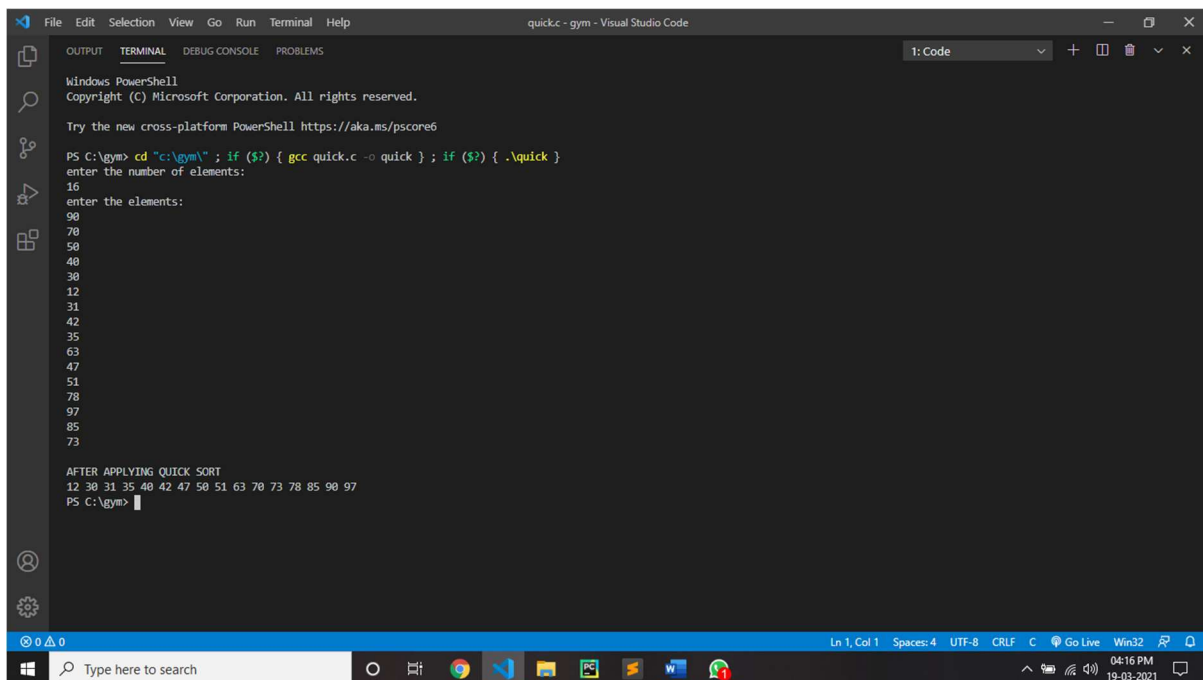
```

void quick_sort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quick_sort(arr, low, pi - 1);
        quick_sort(arr, pi + 1, high);
    }
}

int main()
{
    int n, i;
    printf("enter the number of elements:\n");
    scanf("%d", &n);
    int arr[n];
    printf("enter the elements:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    quick_sort(arr, 0, n - 1);
    printf("\nAFTER APPLYING QUICK SORT\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
}

```

OUTPUT:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\gym> cd "C:\gym\" ; if ($?) { gcc quick.c -o quick } ; if ($?) { .\quick }
enter the number of elements:
16
enter the elements:
90
70
50
40
30
12
31
42
35
63
47
51
78
97
85
73

AFTER APPLYING QUICK SORT
12 30 31 35 40 42 47 50 51 63 70 73 78 85 90 97
PS C:\gym>
```

MERGE SORT:

CODE:

```
#include <stdio.h>
```

```
int merge(int arr[], int start, int mid, int end)
```

```
{
```

```
    int i, j, k;
```

```
    int num1 = mid - start + 1;
```

```
    int num2 = end - mid;
```

```
    int arr1[num1], arr2[num2];
```

```
    for (i = 0; i < num1; i++)
```

```
        arr1[i] = arr[start + i];
```

```
    for (j = 0; j < num2; j++)
```

```
        arr2[j] = arr[mid + 1 + j];
```



```
i = 0;
j = 0;
k = start;
while (i < num1 && j < num2)
{
    if (arr1[i] <= arr2[j])
    {
        arr[k] = arr1[i];
        i++;
    }
    else
    {
        arr[k] = arr2[j];
        j++;
    }
    k++;
}
```

```
while (i < num1)
{
    arr[k] = arr1[i];
    i++;
    k++;
}
while (j < num2)
{
    arr[k] = arr2[j];
```

```
        j++;
        k++;
    }
}
```

```
int divide(int arr[], int start, int end)
{
    if (start < end)
    {
        int mid;
        mid = (start + end) / 2;
        divide(arr, start, mid);
        divide(arr, mid + 1, end);
        merge(arr, start, mid, end);
    }
}
```

```
int main()
{
    int n, i;
    printf("enter the number of elements:\n");
    scanf("%d", &n);
    int arr[n];
    printf("enter the elements:\n");
    for (i = 0; i < n; i++)
    {
```

```

scanf("%d", &arr[i]);
}
divide(arr, 0, n - 1);
printf("\nAFTER APPLYING MERGE SORT\n");
for (int i = 0; i < n; i++)
{
    printf("%d ", arr[i]);
}
}

```

OUTPUT:

```

File Edit Selection View Go Run Terminal Help
merge.c - gym - Visual Studio Code

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS
1: Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\gym> cd "c:\gym\" ; if ($?) { gcc merge.c -o merge } ; if ($?) { .\merge }
enter the number of elements:
16
enter the elements:
76
43
54
65
76
87
98
12
32
24
55
64
72
84
98
70

AFTER APPLYING MERGE SORT
12 24 32 43 54 55 64 65 70 72 76 76 84 87 98 98
PS C:\gym>

```

CONCLUSION:

By performing above practical we can conclude that Merge sort is more efficient and works faster than quick sort in case of larger array size , Quick sort is more efficient and works faster than merge sort in case of smaller array size. Quick sort is not stable. Merge sort is stable. For quick sort the best case for time complexity is $O(n \log n)$ and the average case is $O(n \log n)$ and the worst case is $O(n^2)$.For insertion sort the best case for time complexity is $O(n \log n)$ and the average case is $O(n \log n)$ and the worst case is $O(n \log n)$.Space complexity of merge sort is $O(n)$ and for space complexity quick sort is $O(n \log n)$