Name : Sonal Dilip Gholap

UID : 2021600020

Branch : CSE-AIML

Batch : B

Experiment : 2

Aim : Experiment on finding the running time of Merge Sort and Quick Sort algorithm (time complexity analysis)

- **Objectives :**

1. To understand the algorithms of merge sort and quick sort.
2. To derive the best case and worst-case time complexities of merge sort and quick sort.
3. To sort the given arrays using both merge sort and quick sort and present output along with a few comparisons and exchanges.
4. To compare the time complexities of merge sort and quick sort.

- **Algorithm for Merge Sort**

```
MergeSort (arr, left, right)
    if left > right then
        return
    mid = (left + right)/2
    MergeSort (arr, left, mid)
    MergeSort (arr, mid+1, right)
    Merge(arr, left, mid,right)
    End
```

- **<u>Algorithm for Quick Sort</u>**

```
QuickSort (arr, start, end)
    if (start > end) then
        return
    p = Partition (arr, start, end)
    QuickSort (arr, start, p - 1)
    QuickSort (arr, p + 1, end)



Partition (arr, start, end)
    pivot = arr[end]
    i = start - 1
    for j = start to end - 1 do
        if (arr[j] < pivot) then
            i++
            swap arr[i] with arr[j]

    swap arr[i+1] with arr[end]
    return (i+1)
```

- **<u>Derivation of Time Complexity</u>**

1. Merge Sort

* Merge Sort

Merge Sort is a recursive sorting algorithm

The recurrence relation of merge sort is

$$T(n) = \begin{cases} a & n = 1 \\ 2T\left(\frac{n}{2}\right) + kn & n > 1 \end{cases}$$

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

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$$

From the above eqns,

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

$$= 8T\left(\frac{n}{8}\right) + 3n$$

In general,

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

Put $\frac{n}{2^k} = 1$ $\therefore$ $n = 2^k$

$k = \log_2 n$

Date

Page No.

$T(n) = nT(1) + n\log_2 n$

$= an + n\log n$

$T(n) = n\log n$

The best, worst and average case time complexity is $O(n\log n)$.

2. Quick Sort

Date

Page No.

* Quick Sort

For worst case partitioning -

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

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

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

In general,

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

$\therefore$ Worst case time complexity is $O(n^2)$

For best case partitioning,

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

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

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

In General, $T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$

Date
Page No.

Put $\frac{n}{2^k} = 1$

$k = \log_2 n$

$T(n) = n(T(1)) + n\log_2 n$

$T(n) = n \log n$

$\therefore$ Best case time complexity $= O(n \log n)$.

Code :

```
#include <iostream>
using namespace std;
#include <vector>

int swap1 = 0;
int swap2 = 0;
int swap3 = 0;
int count=0;

void swap(int *a,int *b){
int t = *a;
*a = *b;
*b = t;
}

int partition(int arr[],int low,int high,int& swapn){

int pivot = arr[high];
int i = low - 1;

for(int j=low;j<high;j++){
if(arr[j]<pivot){
i++;
swap(&arr[i],&arr[j]);
if(i!=j){
```

```
    swapn++;
}


}
}

swap(&arr[i+1],&arr[high]);
if((i+1) != high){
    swapn++;
}
return (i+1);
}

void quicksort(int arr[],int low,int high,int& swapn){

if(low<high){
int pi = partition(arr,low,high,swapn);

quicksort(arr,low,pi-1,swapn);
quicksort(arr,pi+1,high,swapn);

}

}

void printarr(int a[],int n){
printf("\nSorted Array: ");
for(int i=0;i<n;i++){
printf("%d ",a[i]);
}
}

void printvec(vector<int> nums,int n){
printf("Sorted Array: ");
for(int i=0;i<n;i++){
printf("%d ",nums[i]);
}
}

void merge(vector<int>& nums,long int p,long int q,long int r,int& comp){
        long int n1 = q-p+1;
        long int n2 = r-q;

        int left[n1];
        int right[n2];

        for(int i=0;i<n1;i++){
            left[i] = (nums[p+i]);
        }
```

```
    for(int i=0;i<n2;i++){
        right[i] = (nums[q+i+1]);
    }
    //left.push_back(INT_MAX);
    //right.push_back(INT_MAX);

    int i=0,j=0,k=p;
    while(i<n1 && j<n2){
        if(left[i]<=right[j]){
            //comp++;
            if(left[i]==right[j]){
                comp--;
            }
            nums[k]=left[i];
            i++;
        }
        else{
            //comp++;
            nums[k]=right[j];
            j++;
        }
        k++;
        comp++;
    }

    while(i<n1){
        nums[k]=left[i];
        //comp++;
        i++;
        k++;
    }

    while(j<n2){
        nums[k]=right[j];
        //comp++;
        j++;
        k++;
    }

    //return nums;
}


void mergeSort(vector<int>& nums,long int p,long int r,int& comp){

    if(p<r){
        long int q = p + (r-p)/2;
        mergeSort(nums,p,q,comp);
        mergeSort(nums,q+1,r,comp);
        merge(nums,p,q,r,comp);
```

```
        }

            //return nums;


    }


int main(){

int comp1 = 0;
int comp2 = 0;
int comp3 = 0;
//int swap1 = 0;

int uid = 20;
int best[10],worst[10];

for(int i=0;i<10;i++){
    best[i] = (uid+((uid+1)*i));
    worst[i] = (uid+((uid+1)*(9-i)));
}

int random[10] = {2,12,76,5,29,7,88,102,35,234};

cout<<"Enter choice:"<<endl<<"1. Merge Sort"<<endl<<"2. Quick Sort"<<endl;
int choice;
cin>>choice;

if(choice==1){
    cout<<"Best Case: ";
    for(int i=0;i<10;i++){
        cout<<best[i]<<" ";
    }
    vector<int> nums(begin(best), end(best));
    mergeSort(nums,0,9,comp1);
    cout<<endl;
    printvec(nums,10);
    cout<<endl<<"Number of Comparisons : "<<comp1<<endl;

    cout<<endl<<endl;

    cout<<"Worst Case: ";
    for(int i=0;i<10;i++){
        cout<<worst[i]<<" ";
    }
    vector<int> nums1(begin(worst), end(worst));
    mergeSort(nums1,0,9,comp2);
    cout<<endl;
    printvec(nums1,10);
```

```
    cout<<endl<<"Number of Comparisons : "<<comp2<<endl;

    cout<<endl<<endl;

    cout<<"Random Case: ";
    for(int i=0;i<10;i++){
        cout<<random[i]<<" ";
    }
    vector<int> nums2(begin(random), end(random));
    mergeSort(nums2,0,9,comp3);
    cout<<endl;
    printvec(nums2,10);
    cout<<endl<<"Number of Comparisons : "<<comp3<<endl;
    cout<<endl<<endl;
}

else if(choice==2){
    cout<<"Best Case: ";
    for(int i=0;i<10;i++){
        cout<<best[i]<<" ";
    }
    quicksort(best,0,9,swap1);
    printarr(best,10);
    cout<<endl<<"Number of Swaps : "<<swap1<<endl;

    cout<<endl<<endl;

    cout<<"Worst Case: ";
    for(int i=0;i<10;i++){
        cout<<worst[i]<<" ";
    }
    quicksort(worst,0,9,swap2);
    printarr(worst,10);
    cout<<endl<<"Number of Swaps : "<<swap2<<endl;

    cout<<endl<<endl;

    cout<<"Random Case: ";
    for(int i=0;i<10;i++){
        cout<<random[i]<<" ";
    }
    quicksort(random,0,9,swap3);
    printarr(random,10);
    cout<<endl<<"Number of Swaps : "<<swap3<<endl;
    cout<<endl<<endl;
}
return 0;
}
```

Output :

1. Merge Sort

```
PS C:\C++ Learning Course>  & 'c:\Users\Sonal Dilip Gholap\.vscode\extensions\ms-vscode.cpptools-1.14.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-5rxu4a01.ksn' '--stdout=Microsoft-MIEngine-Out-hgnpivdz.gfz' '--stderr=Microsoft-MIEngine-Error-jo2v1knk.kb2' '--pid=Microsoft-MIEngine-Pid-zdvjumd0.tnr' '--dbgExe=C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin\gdb.exe' '--interpreter=mi'
Enter choice:
1. Merge Sort
2. Quick Sort
1
Best Case: 20 41 62 83 104 125 146 167 188 209
Sorted Array: 20 41 62 83 104 125 146 167 188 209
Number of Comparisons : 19


Worst Case: 209 188 167 146 125 104 83 62 41 20
Sorted Array: 20 41 62 83 104 125 146 167 188 209
Number of Comparisons : 15


Random Case: 2 12 76 5 29 7 88 102 35 234
Sorted Array: 2 5 7 12 29 35 76 88 102 234
Number of Comparisons : 23


PS C:\C++ Learning Course>
```

2. Quick Sort

```
PS C:\C++ Learning Course>  & 'c:\Users\Sonal Dilip Gholap\.vscode\extensions\ms-vscode.cpptoo
ls-1.14.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In
-15wsp1re.wh1' '--stdout=Microsoft-MIEngine-Out-p45x3bqb.pcg' '--stderr=Microsoft-MIEngine-Err
or-ceg4ghlh.xtf' '--pid=Microsoft-MIEngine-Pid-kee0pgsp.0sh' '--dbgExe=C:\Program Files (x86)\
mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin\gdb.exe' '--interpreter=mi'
Enter choice:
1. Merge Sort
2. Quick Sort
2
Best Case: 20 41 62 83 104 125 146 167 188 209
Sorted Array: 20 41 62 83 104 125 146 167 188 209
Number of Swaps : 54


Worst Case: 209 188 167 146 125 104 83 62 41 20
Sorted Array: 20 41 62 83 104 125 146 167 188 209
Number of Swaps : 29


Random Case: 2 12 76 5 29 7 88 102 35 234
Sorted Array: 2 5 7 12 29 35 76 88 102 234
Number of Swaps : 24


PS C:\C++ Learning Course>
```

Conclusion :

1. In this experiment, we learnt how to perform merge sort and quick sort algorithms and analyze them using their time complexities.
2. From our observations, we come to a conclusion that merge sort is more efficient than quick sort as quick sort always has a worst case time complexity of **$O(n^2)$** while merge sort has best, average and worst case time complexity of **$O(nlogn)$**.