Merge Sort

Implementation:

```
#include<bits/stdc++.h>
using namespace std;

void Merge(int *arr, int *L, int left, int *R, int right)
{
   int i,j,k;
   i=0;
   j=0;
   k=0;
   while(i<left && j<right){
    if(L[i]<R[j]){
      arr[k] = L[i];
      i++;
    }else{
      arr[k] = R[j];
      j++;
    }
    k++;
   }
   while(i<left){
      arr[k++] = L[i++];
   }
   while(j<right){
      arr[k++] = R[j++];
   }
}

void mergeSort(int *arr, int n)
{
   int mid,i,*L,*R;
   if(n<2) return;

   mid = n/2;

   L = (int*)malloc(mid*sizeof(int));
   R = (int*)malloc((n-mid)*sizeof(int));

   for(i=0; i<mid; i++)
   {
      L[i] = arr[i];
   }
   for(i=mid; i<n; i++)
```

Md. Foysal Ahmed
ID: 191-15-12486

```
    {
       R[i-mid] = arr[i];
    }
    mergeSort(L,mid);
    mergeSort(R,n-mid);
    Merge(arr,L,mid,R,n-mid);
    free(L);
    free(R);

}

int main()
{
 int n,i;
 cin>>n;
 int arr[n];
 for(i=0; i<n; i++)
 {
    cin>>arr[i];
 }
 mergeSort(arr,n);
 for(i=0; i<n; i++)
 {
    cout<<arr[i]<<" ";
 }
 cout<<endl;
}
```

Analysis:

We start analyzing the Insertion Sort procedure with the time "cost" of each statement and the number of times each statement is executed.

pseudo code of merge sort:

MERGE_SORT(*A*)

Statements	Cost	Time

`n <- length(A)`
`mid <- n/2`
`left <- array sizeof(mid)`
`right <- array sizeof(n-mid)`
	c1	(1)

`for i <- 0 to mid-1`
`   left[i] <- A[i]`
`for i <- mid to n-1`
`   right[i-mid] <- A[i]`
	c2	n

`MERGE_SORT(left)`	T(n/2)	
`MEGRE_SORT(right)`	T(n/2)	
`MERGE(A,left,right)`	(c3*n+c4)	

Md. Foysal Ahmed
ID: 191-15-12486

After sorting each n/2 elements of the array the merging function will cost asymptotically $T(n) = c * n + c'$ which can be written as $an + b$ which is a linear function and it is written in line 11 as follows.

The running time of the algorithm is the sum of running times for each statement executed; a statement that takes c_i steps to execute and executes n times will contribute $c_i * n$ times to the total running time.

$$T(n) = \begin{cases} c, & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + (c2 + c3) * n + (c1 + c4) \end{cases}$$

This equation can be written as,

$$T(n) = 2T\left(\frac{n}{2}\right) + c'n + c'' \qquad \text{[where c' and c'' are some constant]}$$

When the value of n is grater then 1 then constant c'' become negligible (n>1)

$$T(n) = 2T\left(\frac{n}{2}\right) + c'n$$

We can find out T(n/2) to substitute T(n),

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + c'\left(\frac{n}{2}\right)$$

Substituting,

$$T(n) = 2\left\{2T\left(\frac{n}{4}\right) + c'\left(\frac{n}{2}\right)\right\} + c'n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2c'\left(\frac{n}{2}\right) + c'n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + c'n + c'n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2c'n$$

Again,

$$T\left(\frac{n}{4}\right) = 4T\left(\frac{n}{16}\right) + 2c'\left(\frac{n}{4}\right)$$

Substituting,

$$T(n) = 4\left\{4T\left(\frac{n}{16}\right) + 2c'\left(\frac{n}{4}\right)\right\} + 2c'n$$

$$T(n) = 16T\left(\frac{n}{16}\right) + 8c'\left(\frac{n}{4}\right) + 2c'n$$

$$T(n) = 16T\left(\frac{n}{16}\right) + 2c'n + 2c'n$$

$$T(n) = 16T\left(\frac{n}{16}\right) + 4c'n$$

.

.

.

.

$$T(n) = 2^k\,T\left(\frac{n}{2^k}\right) + kc'n$$

Md. Foysal Ahmed
ID: 191-15-12486

Assume,

$$\frac{n}{2^k} = 1$$
$$=> 2^k = n$$
$$=> k = \log_2 n$$

Therefore,

$$T(n) = 2^{\log_2 n}\, T(1) + \log_2 n * c'n$$
$$= n * c + c'n * \log_2 n \qquad [T(1) = c]$$

If the array is already sorted till the recursive call will be execute and the time function will remain as same.

<u>Time Complexity:</u>

Worst Case: The time function we get from the analysis is $T(n)$ = $n * c + c'n * \log_2 n$
So the complexity of merge sort in worst case is $O(n\ logn)$

Best Case: The time function we get from the analysis is $T(n)$ = = $n * c + c'n * \log_2 n$
So the complexity of merge sort in best case is $O(n \log n)$

Md. Foysal Ahmed
ID: 191-15-12486