Name → Sajal Sourav
ENS-NO → IS114062

Date : __/__/____

Merge sort

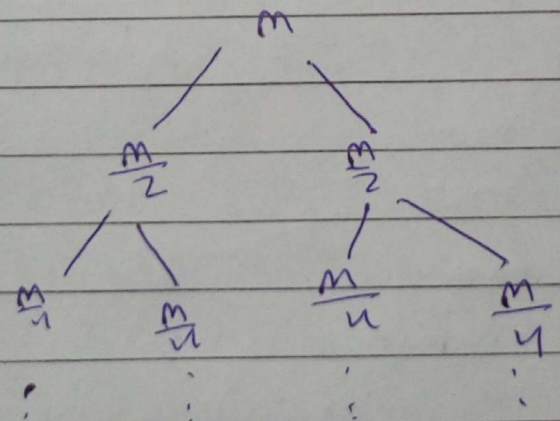
```
void mergesort (int *a, int low, int high)
{
  int mid;
  if(low < high)
  {
      mid = (low+high)/2
      mergesort(a, low, mid);
      mergesort(a, mid+1, high);
      merge(a, low, high, mid);
  }
  return;
}
void merge (int *a, int low, int high, int mid)
{
     int i, j, k, c[50];
     i = low
     k = low
     j = mid+1
  while(i<=mid && j<=high)
  {  if(a[i]<a[j])
     { c[k] = a[i]
       k++; i++;
     }
     else
     { c[k] = a[j]
       k++; j++;
     }
  }
  while(i<=mid)
  { c[k]=a[i]; k++; i++;
  }
```

UPCP

while (j<high)
{ c[k] = a[j]
k++ ; j++ ;
}
for(int i=low ; i<k ; i++)
{ a[i] = c[i] ; }

Conceptually, a merge sort works as follows
1) Divide the unsorted list into n sublists
2) Repeated merge sublist to produce new sorted sublist until there is only 1 sublist remaining.

So, in merge sort the tree goes like this



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

At each level we are merging some smaller segment to form bigger one. merging sorted array to form sorted array takes O(n). Since there are $\log_2^N$ levels the total time complexity - $O(N \log^N)$

Proof

Assume that it gives sorted array for $\frac{N}{2}$ elements and if merging is correct then by this merging of $\frac{N}{2} + \frac{N}{2}$ elements

gives sorted array of N elements.
Now ~~[illegible]~~ in the last level when no of elements = 1 it returns sorted array so whole array will become sorted.

Quick sort

```
void quick sort (int arr[], int l, int r)
{
    int i = l ; int j = r
    int tmp;
    int pivot = arr [(l+r)/2]

    while (i <= j) {

    while (arr[i] < pivot) i++

    while (arr[i] > pivot) j--;

      if (i <= j) {  swap(arr[i], arr[j])
                     i++ ;
                     j-- ;
                  }
    if(l < j) quicksort (arr, left, i);

    if (i < r) quick sort (arr, i, right);

}
```

Date : __/__/____

In quick sort we chooses a pivot element and partion the array about the pivot point. then we recursively send the left side of pivot element for sorting and then the right side.

Since after partioning the ~~partion~~ pivot elements comes to it's right place in sorted array and the the left side and right side also goes for partioning

Running Time Analysis

~~T(n) = T(k)~~

Let K be the position of pivot element

$\therefore T(n) = T(K-1) + T(n-K) + O(n)$

this comes to $n \log n$ for average K

For worst case $K = 1$

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

this is $O(n^2)$

$\therefore$ when array is already sorted in decreasing order. quick sort take $O(n^2)$

Memory Complexcity = $O(n)$ since we used only one array