**Brief Report of Merge Sort**

1. **Divide array into two halves. O(1)**

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int\*L = NULL; // Pointer to int, initialize to nothing.

int\*R = NULL;

L = new int[n1]; // Allocate n ints and save ptr in array.

R = new int[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] >= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

1. **Recursively sort each half. T(n/2) + T(n/2)=2T(n/2)**

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l + (r - l) / 2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

3. Merge two halves to make sorted whole. **O(n)**

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int\*L = NULL; // Pointer to int, initialize to nothing.

int\*R = NULL;

L = new int[n1]; // Allocate n ints and save ptr in array.

R = new int[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] >= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

**Time Complexity of the worst case**

As divide use O(1), recursively sort use 2T(n/2), merge use O(n), thus worst case of time complexity of merge sort in descending order as original array is {1,2,3,4,5,6,7} to {7,6,5,4,3,2,1} is as below:

T(n)<=2T(n/2)+O(n)

a=2, b=2, d=1

* Logba=d=1
* O(nlogn)
* Thus, the worst case of time complexity of merge sort in descending order is O(nlogn)