ADVANCED ALGORITHMS LAB

ASSIGNMENT - 2

MERGE SORT

Name - AMAR PARAMESWARAN

Class - TYCS A

PRN - 19070122016

Advanced Algorithm Lab Assignment No: 2

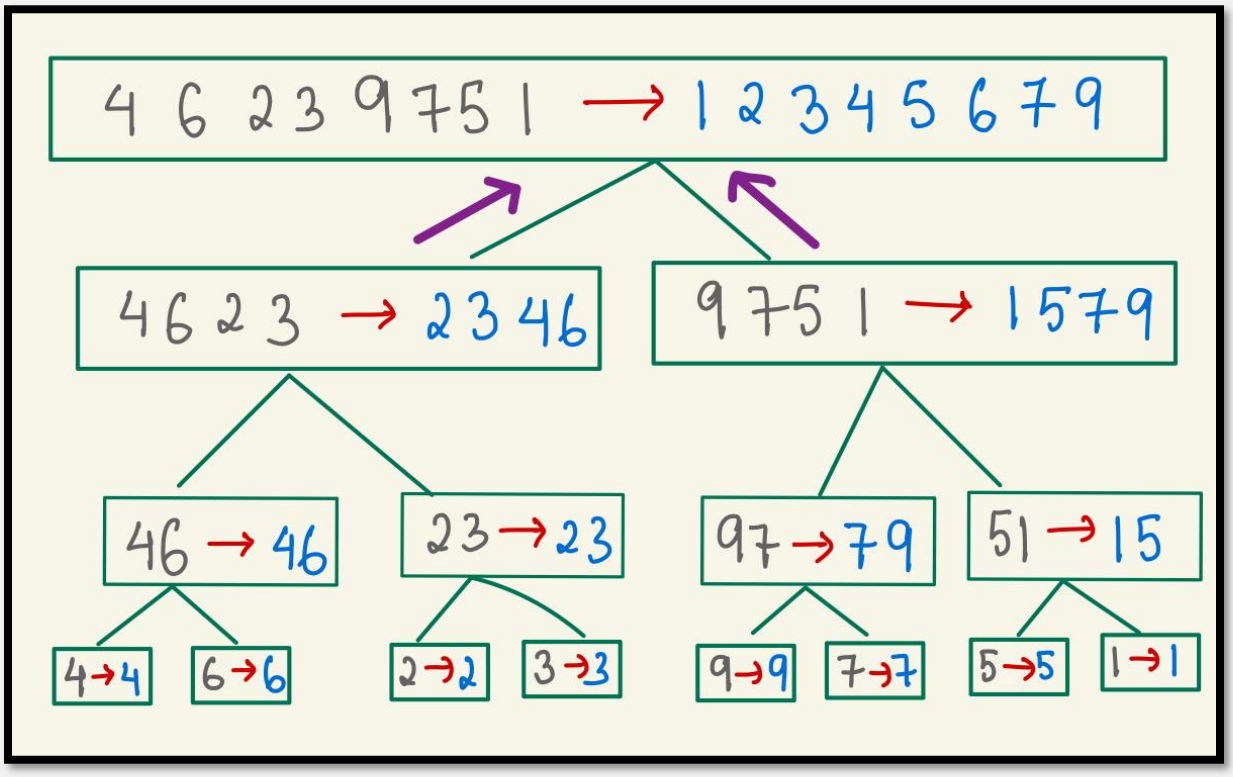
Name of the Student	PRN	Batch	TY CS/IT Division
Amar Parameswaran	19070122016	TY CS	A

TOPIC: MERGE SORT

Working Mechanism:

Merge Sort works on the principle of Divide and Conquer. It works by breaking down until we reach a single element and then by arranging them into a particular order, we merge the whole list.

Example:



Pseudo Code:

```
Pseudocode:
MergeSort(A, p, r):
    if p < r:
        q = (p+r)/2
        Mergesort(A, p , q)
        Mergesort(A, q+1, r)
        Merge(A, p, q, r)
```

Code Snippet:

```
void DisplayArr(int arr[MAX])   //Displays the array
{
   int i;
   for(i=0;i<n;i++)
      cout<<arr[i]<<" ";
}

void merge(int arr[MAX], int s, int mid, int e)
{
   //count-keeps track of passes
   static int count=1;
   //s-start , e-end , mid-middle
   int i = s;
   int j = mid+1;
   int k = s;

   int tempArr[MAX];   //temporary array stores sorted elements

   //Compares the elements in both halves and stores them in sorted order
   while (i<=mid && j<=e)
   {
      if (arr[i]<=arr[j])
      {
         tempArr[k] = arr[i];
         i++;
      }
      else
      {
         tempArr[k] = arr[j];
         j++;
      }
      k++;
```

```
    }

    //Stores remaining elements of the first half in the temporary array
    while (i<=mid)
    {
        tempArr[k] = arr[i];
        i++;
        k++;
    }

    //Stores remaining elements of the second half in the temporary array
    while (j<=e)
    {
        tempArr[k] = arr[j];
        j++;
        k++;
    }

    //Copies the elements of temporary array in original array
    for(i=s;i<=e;i++)
        arr[i]=tempArr[i];

    //Displays the passes
    cout<<"\nPass "<<count<<" => ";
    DisplayArr(arr);
    count++;
}

void MergeSort(int arr[MAX], int s, int e)
{
    if (s>=e) //Stopping condition
        return;
    int mid=s + (e-s)/2;

    // Sort first and second halves
    MergeSort(arr,s,mid); //first half
    MergeSort(arr,mid+1,e); //second half

    merge(arr,s,mid,e);       //Merge the 2 halves
}
```

Output:

Link for code: https://github.com/Amar1709/AdvancedAlgos_Practice.git

AMAR PARAMESWARAN
TYCSA
19070122016

MERGE SORT

Time Complexity And Space Complexity

⇒ Best Time Complexity = Average Time Complexity = Worst time complexity

Time Complexity,

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

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

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

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

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

$$= 2^3 T\left(\frac{n}{8}\right) + 3n$$

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

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

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

$\therefore T(n) = O(n\log_2 n)$

AMAR PARAMESWARAN
TYCSA
19070122016

⇒ **Space Complexity = O(n)**

Since the space occupied stays the same before and after the sort the space complexity of merge sort is O(n)

★ **REAL-LIFE APPLICATIONS**

→ Adding up change

This is a really easy example of merge sort. When we are sorting change, we tend to divide the change into small denominations, then total up each denomination before adding them together.

★ **OPTIMISATIONS**

→ We can optimize by arranging the array in sorted order before performing merge sort so that the merge phase is skipped.
It can be easily implemented by comparing the last element in the left subarray and the first element in the right subarray.
