**LAB # 05**

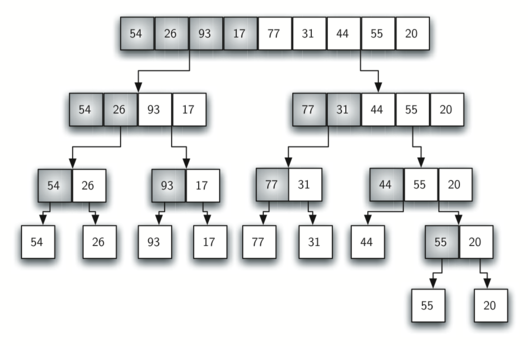
**Introduction to Sorting Algorithms continue**

## **Introduction**

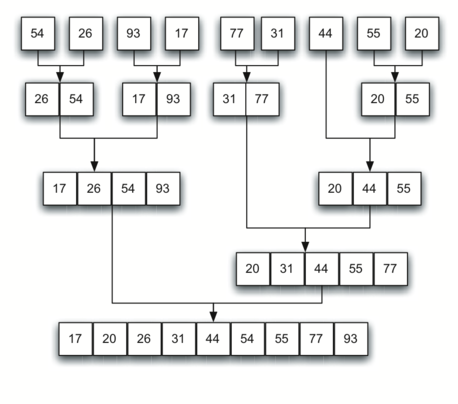
### Merge sort

Merge sort is a divide and conquer strategy to improve the performance of sorting algorithms. Merge sort is a recursive algorithm that continually splits a list in half. If the list is empty or has one item. If the list has more than one item, we split the list and recursively invoke a merge sort on both halves. Once the two halves are sorted, the fundamental operation, called a **merge**, is performed. Merging is the process of taking two smaller sorted lists and combining them together into a single, sorted, new list.

**Split Array**



**Merge Array**



##### MERGE SORT

MergeSort(OrignalArray) {

n <-length(OrignalArray)

int mid, LeftArray, RightArray;

if (n < 2) return; // base condition. If the array has less than two element, do nothing.

mid = n / 2; // find the mid index.

// create left and right subarrays

// mid elements (from index 0 till mid-1) should be part of left sub-array

// and (n-mid) elements (from mid to n-1) will be part of right sub-array

LeftArray = arrayOfSize(mid)

RightArray= arrayOfSize(n - mid)

for i = 0 to mid - 1

LeftArray[i] = OrignalArray[i]; // creating left subarray

for i = mid to i<n

R[i - mid] = OrignalArray[i]; // creating right subarray

MergeSort(LeftArray); // sorting the left subarray

MergeSort(RightArray); // sorting the right subarray

Merge(leftArray, rightArray, orignalArray); // Merging L and R into A as sorted list.

}

// Function to Merge Arrays L and R into A.

Merge(orignalArray, LeftArray, RightArray) {

int i, j, k;

// i - to mark the index of left aubarray (LeftArray)

// j - to mark the index of right sub-raay (RightArray)

// k - to mark the index of merged subarray (orignalArray)

elementInLeft = number of elements in LeftArray

elementInRight= number of elements in RightArray.

i = 0; j = 0; k = 0;

while (i<elementInLeft && j< elementInRight) {

if (LeftArray[i] < RightArray[j])

orignalArray[k] = LeftArray[i];

k ++

i++

else orignalArray[k] = RightArray[j];

k++

j++

}

while (i < elementInLeft)

orignalArray[k] = LeftArray[i];

k++

i++

while (i < elementInRight)

orignalArray[k] = RightArray[j];

k++

j++

##### }

## **Time Boxing**

|  |  |  |
| --- | --- | --- |
| Activity Name | Activity Time | Total Time |
| Login Systems + Setting up Visual Studio Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

## **Objectives/Outcomes**

This exercise delivers the idea/ concept of:

* Sorting a static list of values using different Algorithms. Performance issue of sorting algorithms.

## **Lab Tasks/Practical Work**

**Q1** Create a program that take an array of 10 inputs from the user and generate the sorted out put using Merge Sort Algorithms.