**Merging**

Merging is a process of combining two sorted list into a single sorted list

List A List B

List C

2 5

8 9

15 12

18 17

List A = i=0 List B = j=0 List C = k=0

If(A(i)<B(j) --🡪 C[k] = A[i] i++ k++

Else -🡪 c[k] = B[j] j++ k++

Algorithm Merge(List A, List B, Size List A m, Size List B n){

I=0, j=0, k=0

While(i<m && j<n){

If(A[i]<B[j>) C[k++] = A[i++]

Else C[k++] = B[j++]

}}

For(;j<n;j++)

C[k++]=B[j]

For(;i<m;i++)

C[k++] = A[i]

public class MergeTwoSortedArrays {

public static void main(String[] args) {

int[] arr1 = {2,8,15,18};

int arr1\_length = arr1.length;

int[] arr2 = {5,9,12,17};

int arr2\_length = arr2.length;

int[] arr3 = new int[arr1\_length + arr2\_length];

mergeArrays(arr1,arr2,arr1\_length,arr2\_length,arr3);

/\*iterate the resultant array \*/

for(int x : arr3) {

System.out.print(x+" \t");

}

}

static void mergeArrays(int[] arr1, int[] arr2, int arr1\_length, int arr2\_length, int[] arr3)

{

int i=0,j=0,k=0;

/\* Traverse both array \*/

while(i<arr1\_length && j<arr2\_length)

{

/\*

Check if current element of first array is smaller than current element

of second array. If yes, store first array element and increment first array

index. Otherwise do same with second array

\*/

if(arr1[i]<arr2[j])

arr3[k++] = arr1[i++];

else

arr3[k++] = arr2[j++];

}

/\*Store remaining elements of first array \*/

for(;i<arr1\_length;i++) {

arr3[k++] = arr1[i];

}

/\*Store remaining elements of second array \*/

for(;j<arr2\_length;j++) {

arr3[k++] = arr2[j];

}

}

}

**2 Way Merge Sort**

List A – 9 3 7 5 6 4 8 2

First Pass 3 9 5 7 4 6 2 8

Second Pass 3 5 7 9 2 4 6 8

Third Pass 2 3 4 5 6 7 8 9

No of Passes required = logn

**Divide and Conquer**

* Related to a strategy for solving a problem
* Strategy is an approach or design for solving a computational problem
* Other strategies are also there, like 3D method, dynamic programming, backtracking, branch and bound etc
* If a problem is large, then divide into sub-problems , then get the solution for each sub-problems- and combine the solutions of the sub-problems to get the solution for the main problem | if sub-problem is large , then again break that into sub-sub-problems and do divide and conquer

P (problem – size=n)

P1 P2 P3 P4 ........ Pk

S1 S2 S3 S4 ........ Sk

S (Solution)

One important point of Divide and Conquer is – whatever the problem is- the sub problem should be same as that of main problem – as an example if the main problem is to sort, then sub-problem also should be to sort

DAC(Problem p){

If(small(p))

Solution(p)

}

Else{

Divide P into P1, P2, P3, Pk

Apply DAC(P1); DAC(p2), DAC(Pk)

Combine (DAC(P1); DAC(p2), DAC(Pk))

1. Binary Search
2. Finding Maximum and Minimum
3. Merge Sort
4. Quick Sort
5. Etc

Dive and Conquer strategy is recursive

**Merge Sort**

The merge sort algorithm is based on divide and conquers strategy.

It basically divides input array in 2 halves, calls itself for the two halves and then merges the two sorted halves in the end

**Problem – to sort the elements**

A = 9 3 7 5 6 4 8 2

Algorithm MergeSort(low, high){

If(low < high){

mid = (low + high)/2;

MergeSort(low, mid)

MergeSort(mid+1, high)

Merge(low, mid, high)

}

}

**Module of Edureka-4 /34**

Low = 1 High = 8

Mid = (1+8)/2 = 4

Low = 1 High = 4

Mid = (1+4)/2 = 2

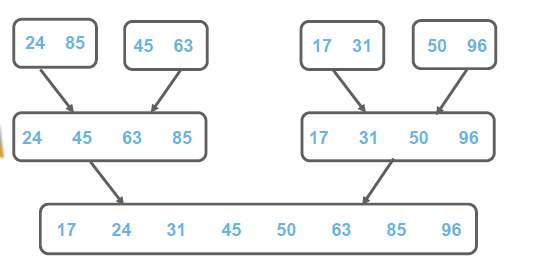
Low = 1 High = 2

Mid = (1+2)/2 = 1

Low = 1 High = 1



Divide



Combine and Conquer

Consider an array A of n number of elements. The algorithm processes the elements in 3 steps:

> If A Contains 0 or 1 elements then it is already sorted, otherwise, Divide A into two sub-array of equal number of elements.

> Conquer means sort the two sub-arrays recursively using the merge sort.

> Combine the sub-arrays to form a single final sorted array maintaining the ordering of the array.

Sort the following array by using merge sort.

A = {10, 5, 2, 23, 45, 21, 7}

public class MergeSort

{

void merge(int arr[], int beg, int mid, int end)

{

int l = mid - beg + 1;

int r = end - mid;

int leftArray[] = new int [l];

int rightArray[] = new int [r];

for (int i=0; i<l; ++i)

leftArray[i] = arr[beg + i];

for (int j=0; j<r; ++j)

rightArray[j] = arr[mid + 1+ j];

int i = 0, j = 0;

int k = beg;

while (i<l&&j<r)

{

if(leftArray[i] <= rightArray[j])

{

arr[k] = leftArray[i];

i++;

}

else

{

arr[k] = rightArray[j];

j++;

}

k++;

}//while

while (i<l)

{

arr[k] = leftArray[i];

i++;

k++;

}

while (j<r)

{

arr[k] = rightArray[j];

j++;

k++;

}

}

void sort(int arr[], int beg, int end) {

if (beg < end) {

int mid = (beg + end) / 2;

sort(arr, beg, mid);

sort(arr, mid + 1, end);

merge(arr, beg, mid, end);

}

}

public static void main(String args[])

{

int arr[] = {90,23,101,45,65,23,67,89,34,23};

MergeSort msObj = new MergeSort();

msObj.sort(arr, 0, arr.length-1);

System.out.println("Sorted Array: ");

for(int i =0; i<arr.length;i++)

{

System.out.print(arr[i]+" ");

}

}

}

**Pros & Cons of Merge Sort**

Pros:

1. Suitable for Large Size List
2. Suitable for LinkedList – we can merge twi LinkedLists without creating a new LinkedList
3. Supports External Sorting
4. Stable

Cons:

1. Extra Space
2. No small problem
3. Recursive – uses stack