**1. Problem Discussion:**

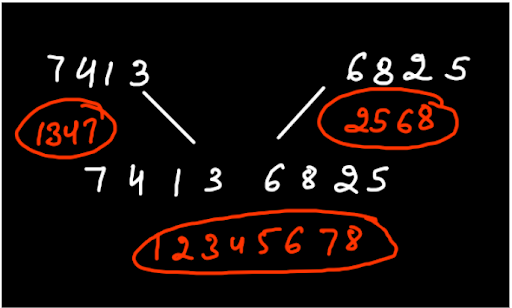
You are given an array(arr) of n integers and have to use the merge sort to sort the array in increasing order. You've previously completed "Merge two Sorted Arrays" in the preceding question.

[**Merge two Sorted Arrays**](https://www.youtube.com/embed/WMxNhIBA92I)

That technique will be used in Merge Sort, so watch that video now if you haven't yet.

**2. Approach:**

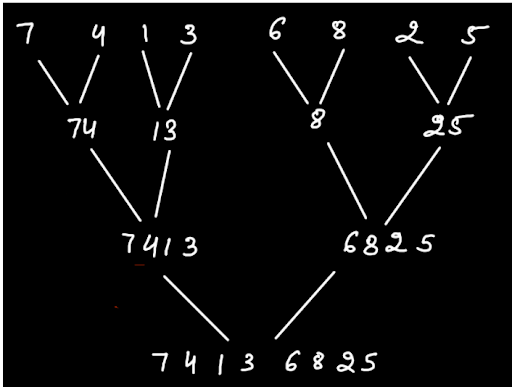
Consider the following array: [7, 4, 1, 3, 6, 8, 2, 5]. As seen in Figure 1, we partition this array into two halves.

****

7,4,1,3 and 2,5,6,8 are the first and second parts, respectively. Do you recall that we always believe in recursive questions, reader? So, if we divide the array into two halves and preserve faith, we know that each part of the array will return as a sorted array. In Figure 1, we believe that recursively calling the function on 7,4,1,3 will return 1,3,4,7, and similarly calling it on 6,8,2,5 would return 2,5,6,8. We now know how to combine two sorted arrays. As a result, we combine the sorted arrays from the two portions (1,3,4,7 and 2,5,6,8) to form a new merged and sorted array (1,2,3,4,5,6,7,8).

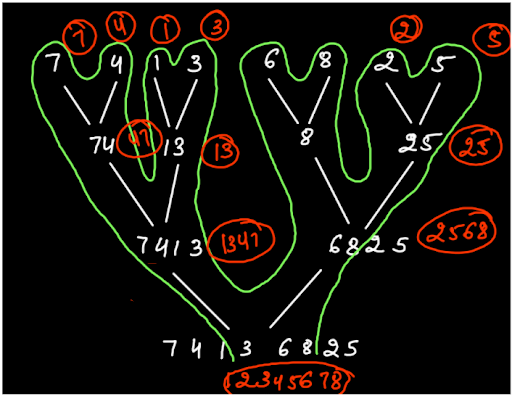
Recursion Tree

You'll observe that each array is divided into two arrays until the last array has only one element remaining and can't be divided any more.

****

Euler Path

Let's look at how we go through the recursion tree now that we've seen it. The Euler path can be seen as a green highlight. From the unsorted array 7,4,1,3,6,8,2,5, we progress up the levels until we reach the last level, which includes a single element array 7. When we get there, a new array with one element named '7' is added to the previous array 7,4. The path now returns to the previous level, and the stack's last level is popped. As the Euler route progresses up a level, the next level is pushed back into the stack. A new array with the element '4' is returned once more. Now that we've reached the fourth post area, the two sorted arrays 7 and 4 are merged, and a new array with 4,7 is returned.

****

This Euler path continues to sort two arrays, merge them, and return them to the previous level. Recursively, when we reach the first level, we get two sorted arrays, 1,3,4,7 and 2,5,6,8, which are merged, and when the stack fills up, we get the final answer, 1,2,3,4,5,6,7,8. Here is the code:

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static int[] mergeSort(int[] arr, int lo, int hi) {

if(lo == hi){

int[] bans = new int[1];

bans[0] = arr[lo];

return bans;

}

int mid = (lo + hi) / 2;

int[] lsa = mergeSort(arr,lo,mid);

int[] rsa = mergeSort(arr,mid + 1,hi);

int[] ans = mergeTwoSortedArrays(lsa,rsa);

return ans;

}

//used for merging two sorted arrays

public static int[] mergeTwoSortedArrays(int[] a, int[] b){

System.out.println("Merging these two arrays ");

System.out.print("left array -> ");

print(a);

System.out.print("right array -> ");

print(b);

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

int[] ans = new int[a.length + b.length];

while(i < a.length && j < b.length){

if(a[i] <= b[j]){

ans[k] = a[i];

i++;

k++;

}else{

ans[k] = b[j];

j++;

k++;

}

}

while(i < a.length){

ans[k] = a[i];

k++;

i++;

}

while(j < b.length){

ans[k] = b[j];

k++;

j++;

}

return ans;

}

public static void print(int[] arr) {

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

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

}

System.out.println();

}

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = scn.nextInt();

}

int[] sa = mergeSort(arr,0,arr.length - 1);

System.out.print("Sorted Array -> ");

print(sa);

}

}

**3. Code Discussion:**

1. We call mergeSort with the specified array to be sorted, as well as the parameters lo and hi. The index 0 at the start of the array is lo, and the index arr.length-1 at the end of the array is hi. 2. Base Case: If lo and hi intersect at any point, an array of size 1 is created and the element at lo (or hi) is stored. After that, the array is returned. 3. Finally, we identify the array's midpoint to divide our unsorted array into two halves. 4. Then we recursively run the function mergeSort on the first array, which then returns a sorted array (By our Faith) (By our Faith). 5. Similarly, we execute mergeSort on the second array in a recursive manner, which returns a sorted array (By our Faith). 6. When we have two sorted arrays, we merge them. 7. We return this merged sorted array as the answer now.

**4. Analysis:**

Time Complexity: O(NlogN)

Because merge sort divides the array into two parts and merges these halves in linear time, the time complexity is of order nlogn.

Space Complexity: O(N)

We are using 1D arrays, so space complexity will be linear.