Merge Sort (Based on Divide & Conquer Rule)

Ex: arr = [10, 24, 76, 73]

Note:- Whenever a function return something its execution gets completed.

// mergeSort Code → Divide till arr.length ≤ 1

```
function mergeSort (arr) {
    if (arr.length ≤ 1) return arr;
                        [Base Case]

    let mid = Math.floor(arr.length/2),
        left = mergeSort(arr.slice(0, mid)),
        right = mergeSort(arr.slice(mid));

    return merge(left, right);

}
mergeSort([10, 24, 76, 73]);
```

// mergeArrays Code → Conquering

```
function merge (arr1, arr2) {
    let i = 0,
        j = 0,
        result = [];
    while (i < arr1.length && j < arr2.length) {
        if (arr1[i] < arr2[j]) {
            result.push(arr1[i]);
            i++;
        }
else{
    result.push(arr2[j]);
     j++;
    }
  }
  while (i< arr1.length) {
        result.push(arr1[i]);
        i++;
  }
  while (j< arr2.length) {
        result.push(arr2[j]);
          j++;
     }
     return result;
}
```

Pseudo Code:-

(1). MergeSort Code (Divide):-

Steps:

(i) Firstly, we find the 'mid' point of the array.

(ii) Then, recursively (Function under root function) execute the 'Left Parent Function' with argument (array having Left half of the Root/Main Array from 0 to mid Point).

(iii). Now, ~~divide the~~ find the 'mid' point of the 'Left Parent Array' till the length of the array to 1.

(iv) Then, Start storing the ~~1 length of~~ array to the left side of the 'Left Parent Array'.

(v) Then, Store array to the right side of the 'Left Parent Array'.

(vi) Then merging the left & right side of the 'Left Parent Array'.

(vii) Merging Function return the merge array to the 'Left Parent function' and keep it store till the 'Right Parent Function' executed.

(viii) Follow Step:-(ii) to (vii) for 'Right Parent Function'

(ix) Then, merges 'Left' & 'Right' Partial Array Function. and Return merge ~~Function~~ back to the 'Root' MergeSort Function.

② . Merging Array (Conquer):-

Steps:- Initialize:- i = j = 0, result = [ ];

(i) Start Looping Over both array from first to Last position.

- if Element of 1st arr is < Element of 2nd arr, push Element of 1st arr to an Empty arr.
- if Element of 2nd arr is < Element of 1st arr, push Element of 2nd arr to an Empty arr (result[ ])

(ii) If looping not done over 1st arr, then remaining element of 1st arr will not move to result [ ]. So, create a person loop for it and push all remaining elemt to result[ ]

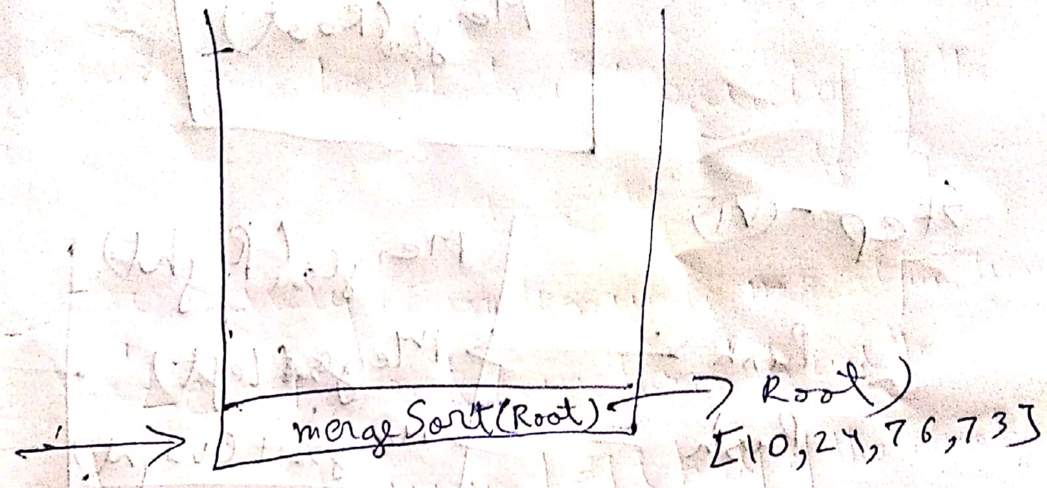
(iii) If looping not done over 2nd ,, ,,

~~So~~ Same as Step (ii) for 2nd array.

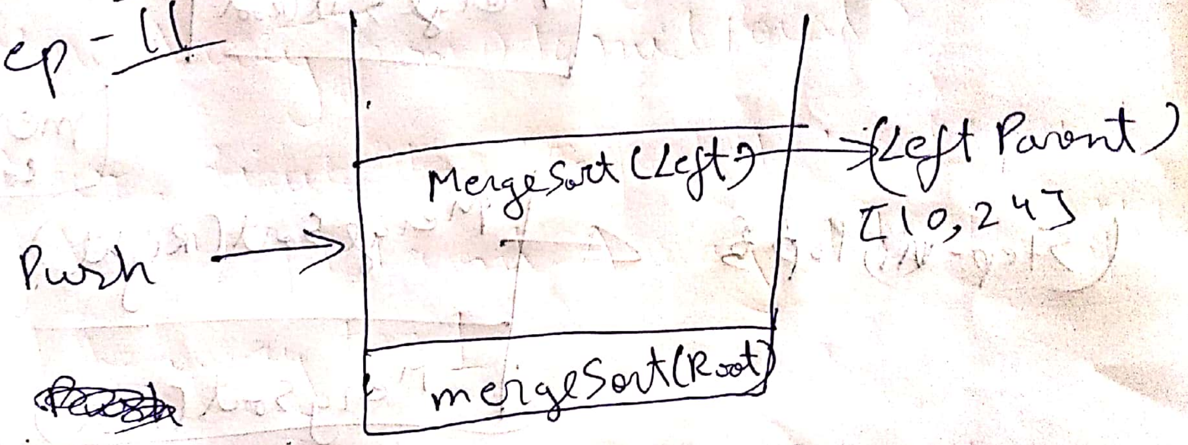
(iv) Return result [ ].

Note:
Merge Sort Code ; Stack Representation
for the Example of arr [10, 24, 76, 73]

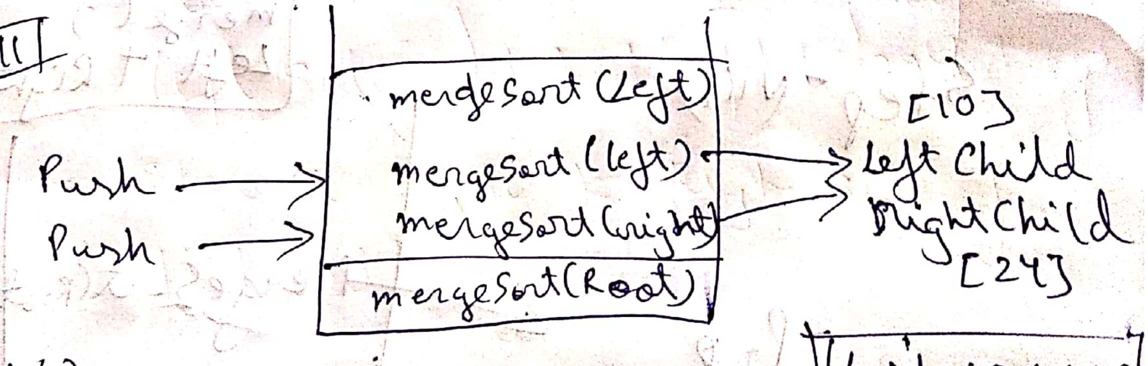
Step - I



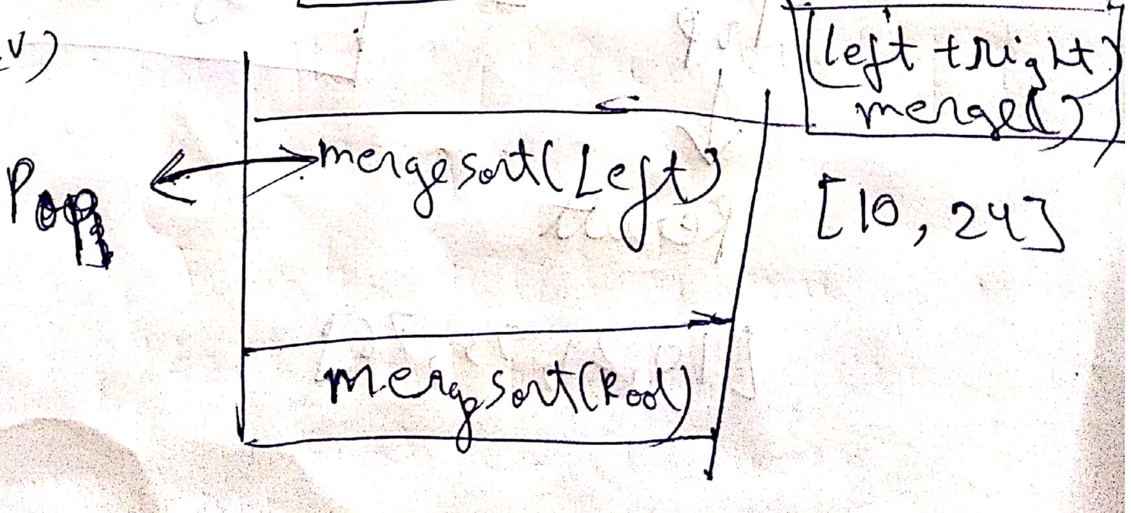
Step - II



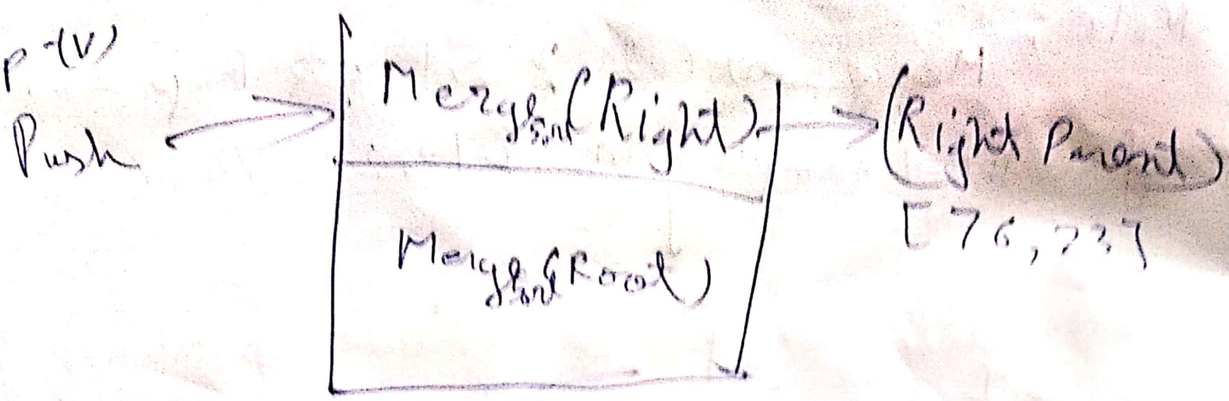
Step - III



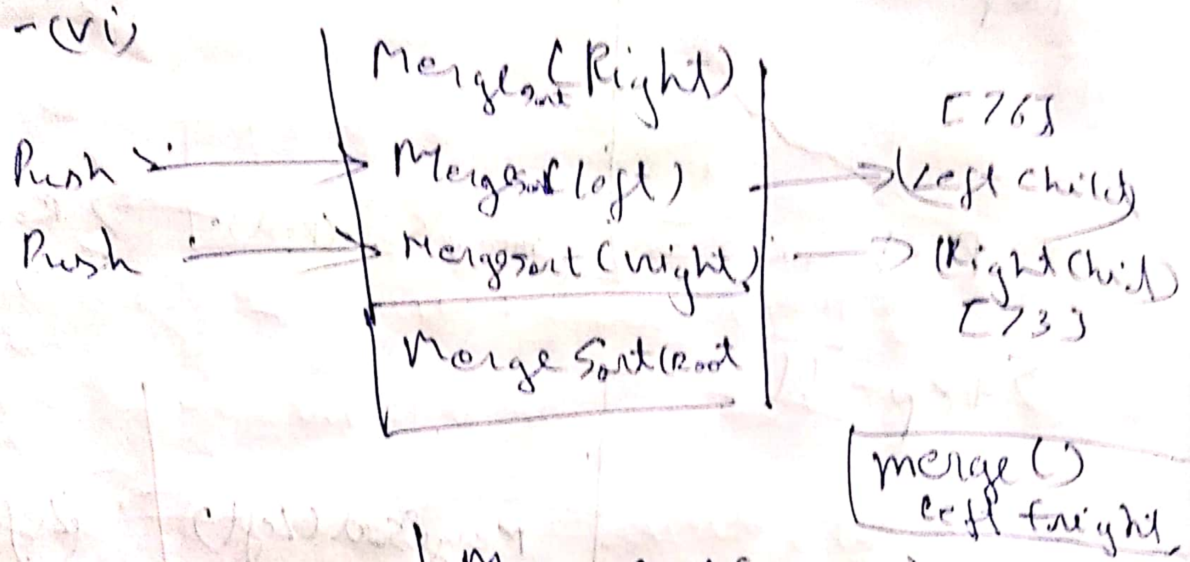
Step - (IV)



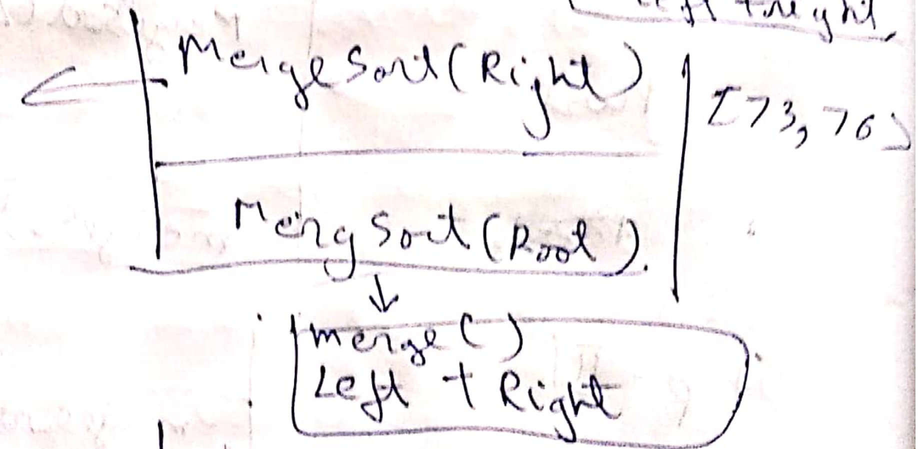
Step-(v)



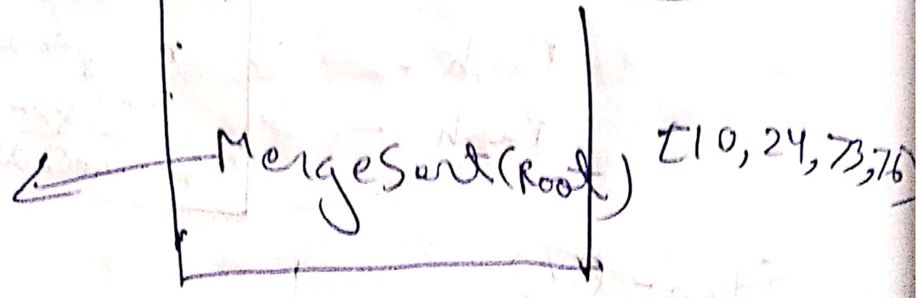
Step-(vi)



(Step-vii) Pop



Step-viii)



Pop
↓
Result

[10, 24, 73, 76]

Ex: arr = [10, 24, 76, 73]

// mergeSort()

~~for~~ function mergesort()
    if (arr.length ≤ 1) return arr;
    $\boxed{5}$, $\boxed{8}$, $\boxed{20}$, $\boxed{23}$
    [10] [24] [76] [73]

    mid = 2 $\boxed{1}$, 1 $\boxed{3}$, 1 $\boxed{18}$ (calling)

    left = ([10, 24]) $\boxed{2}$, ([10]) $\boxed{4}$, [10] $\boxed{6}$; ([76]) $\boxed{19}$, [76] $\boxed{21}$

[10, 24] $\boxed{16}$

    right = ([24]) $\boxed{7}$, [24] $\boxed{9}$; ([76, 73]) $\boxed{17}$, ([73]) $\boxed{22}$, [73] $\boxed{24}$

[73, 76] $\boxed{31}$

    return merge([10], [24])) $\boxed{10}$, ([76, 73]) $\boxed{25}$
    function calling
    merge([10, 24], [73, 76]) $\boxed{32}$

// merge()

function merge(arr1, arr2)
    result = [10 $\boxed{12}$, 24 $\boxed{14}$], [73 $\boxed{27}$, 76 $\boxed{29}$]
    if () {
        result.push(10);
    $\boxed{11}$
    else result.push(73)
    $\boxed{26}$
    while {
        result.push(24);
    $\boxed{13}$
    return [10, 24] $\boxed{15}$, [73, 76] $\boxed{20}$
    while {
        result.push(76);
    $\boxed{28}$