17-03-2023

Divide & conquer (class 1)

Date.....

* **Merge Sort** :- Merge sort is one of the most efficient sorting algorithms. It works on the principle of Divide & conquer.

↓

[FOOT DAALO, RAAJ KARO]

Merge sort repeatedly break down an array into several arrays, until each array consist of a single elemen-t & merging those arrays in a manner that results into a sorted array.

For example, let suppose we have given an unsorted array,

0	1	2	3	4	5	6	7
7	5	2	4	1	6	3	0

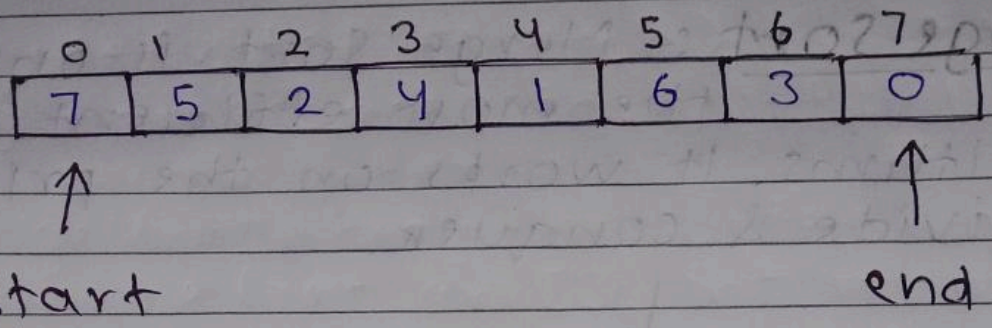
↳ First we calculate the middle of this array, like we do in Binary Search.

After calculating the middle, we have two arrays, first array with start index to mid index & another array with mid +1 index to end index. The same procedure will follow until we have individual element left.

Spiral

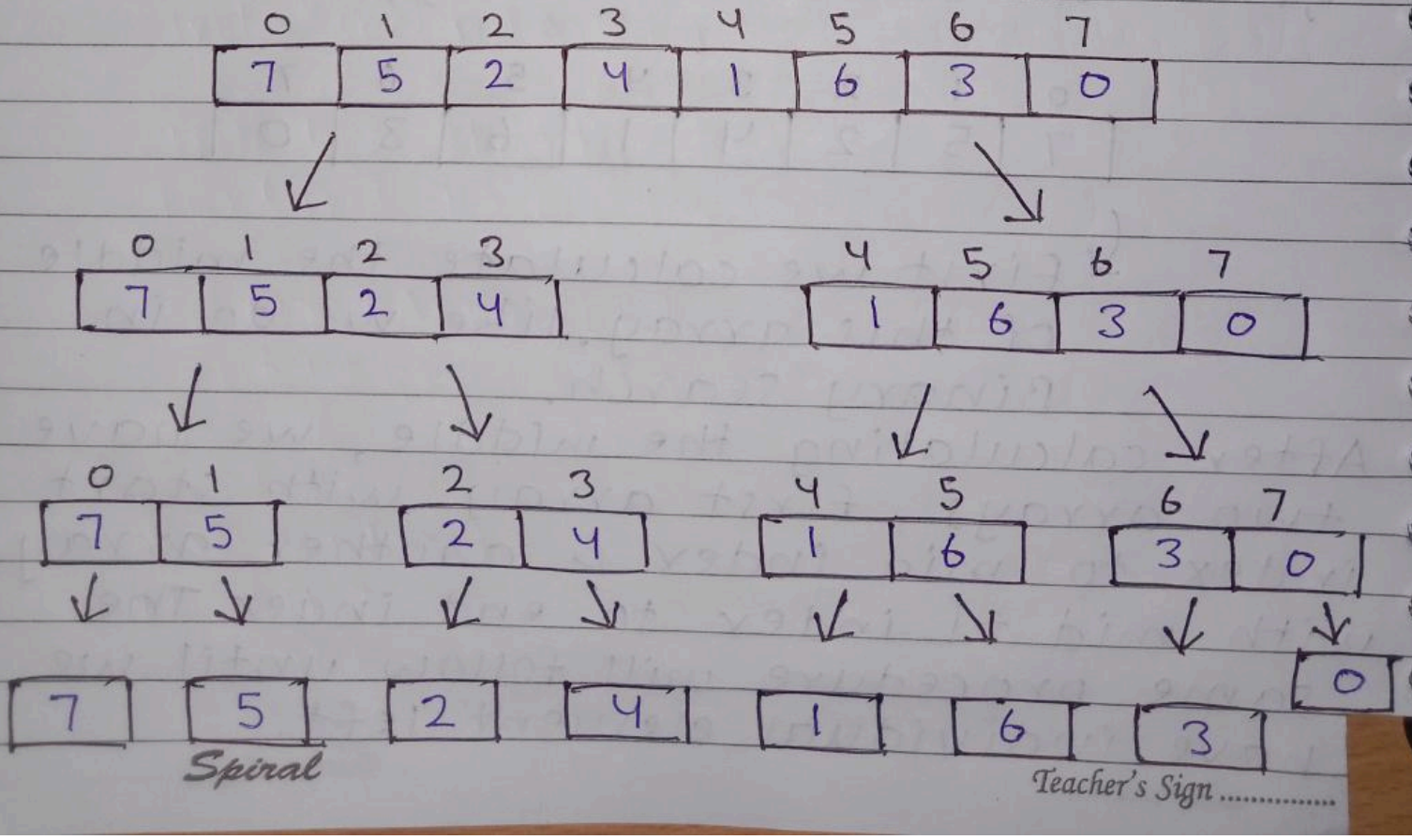
Teacher's Sign.....

It means,



mid = 3 ⟵ mid = (0 + 7)/2

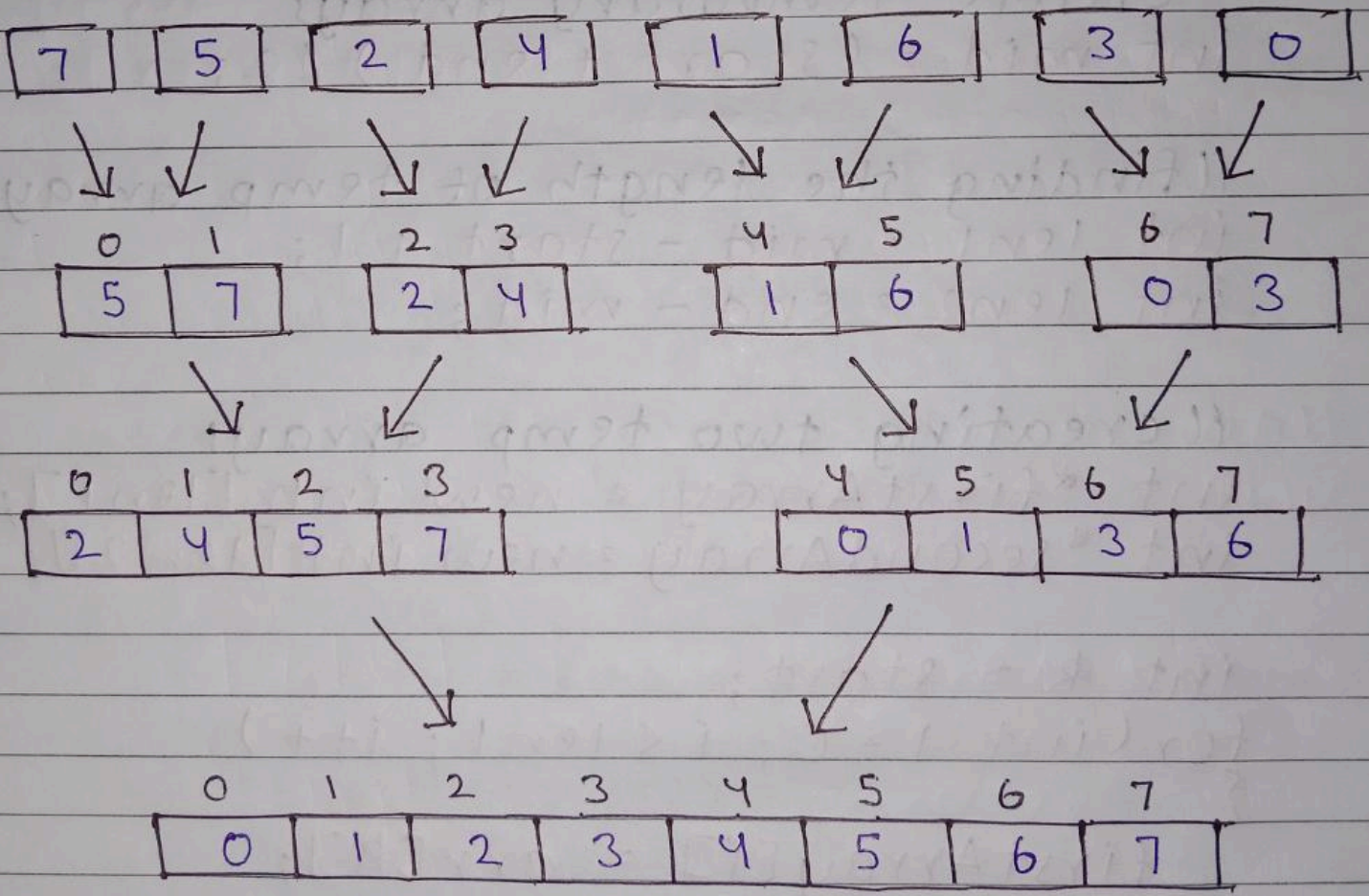
3 is our mid element. Now, we divide this array into 2 parts. First part is from 0 to 3, and second part is from 3 + 1 to 7, i.e., 4 to 7. Same strategy will follow until we have only single element left.



Teacher's Sign .....

Date.....

Now, we have single elements left. After this step we do the next thing, i.e., "merging", we compare the 2-2 pairs, if the left one is smaller than the right one, we leave it as it is, but if the right one is smaller than the left one then we swap the elements & the same procedure will follow until we find the proper sorted array.



Final Sorted Array.

code :-

```
#include<bits/stdc++.h>
using namespace std;

// merge the arrays in sorted way
void merge(int arr[], int n, int
                    start, int end)
{
    // calculating mid, so that we
      create temporary arrays
    int mid = (start + end) / 2;

    // finding the length of temp arrays
    int len1 = mid - start + 1;
    int len2 = end - mid;

    // creating two temp arrays
    int *firstArray = new int [len1];
    int *secondArray = new int [len2];

    int k = start;
    for(int i=0; i<len1; i++)
    {
        firstArray[i] = arr[k];
        k++;
    }

    k = mid + 1;
    for(int i=0; i<len2; i++)
    {
        secondArray[i] = arr[k];
    } k++;
```

Teacher's Sign .....

```
//merge two arrays
int leftIndex = 0;
int rightIndex = 0;
int mainIndex = start;

//checking both temp arrays & whichever element is smaller, we push the element in main array
while(leftIndex < len1 &&
                  rightIndex < len2)
{
  if(firstArray[leftIndex] <
          secondArray[rightIndex])
  {
     arr[mainIndex] = firstArray
                        [leftIndex];
     mainIndex++;
     leftIndex++;
  }
  else
  {
     arr[mainIndex] = secondArray
                        [rightIndex];
     mainIndex++;
     rightIndex++;
  }
}

//if one of the temp array is finished, then we pushed the other array elements in main array.
```

```
//merge two arrays
int leftIndex = 0;
int rightIndex = 0;
int mainIndex = start;

//checking both temp arrays & whichever element is smaller, we push the element in main array
while(leftIndex < len1 &&
                rightIndex < len2)
{
  if(firstArray[leftIndex] <
          secondArray[rightIndex])
  {
     arr[mainIndex] = firstArray
                        [leftIndex];
     mainIndex++;
     leftIndex++;
  }
  else
  {
     arr[mainIndex] = secondArray
                        [rightIndex];
     mainIndex++;
     rightIndex++;
  }
}

//if one of the temp array is finished, then we pushed the other array elements in main array.
```

```
    while(leftIndex < len1)
    {
        arr[mainIndex] = firstArray
                         [leftIndex];
        mainIndex++;
        leftIndex++;
    }

    while(rightIndex < len2)
    {
        arr[mainIndex] = secondArray
                         [rightIndex];
        mainIndex++;
        rightIndex++;
    }
}

//dividing the array;
void mergeSort(int arr[], int n,
               int start, int end)
{
    if(start >= end)
    {
        return;
    }
    int mid = (start + end)/2;

    // left array
    mergeSort(arr, n, start, mid);

    //right array
    mergeSort(arr, n, mid+1, end);
```

Date.....

```
    //sorting part
    merge(arr, n, start, end);
}

int main()
{
    int arr[] = {5, 9, 8, 0, 4, 2, 1};
    int n = 7;

    cout << "array before sorting ";
    for(int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }

    mergeSort(arr, n, start, end);

    cout << "array after sorting ";
    for(int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }

    return 0;
}
```

Time complexity of merge sort:-
we need to find the recursive relation of the code,

$$T(n) = k + T(n/2) + T(n/2) + O(n)$$

- k: Base case
- $T(n/2)$: Left Part
- $T(n/2)$: Right Part
- $O(n)$: Merge

$$T(n) = k + 2T(n/2) + n * k$$

↳ This is constant so we can ignore this 'k'

$$T(n) = 2T(n/2) + n * k$$

$$T(n/2) = T(n/4) + n/2 * k \quad \} \times 2$$

$$T(n/4) = T(n/8) + n/4 * k \quad \} \times 4$$

$$T(n/8) = T(n/16) + n/8 * k \quad \} \times 8$$

⋮

$T(1) = k$ → constant time to sort array of size 1.

Date.....

$T(n) = (a-1) \times n \times k + k$

$T(n) = (\log n - 1) \times n \times k + k$

$T(n) = k \times \log n \times n - nk$

$T(n) = nk\log n$

↳ ignore, because of constant.

$T(n) = n\log n$.

↳ Time complexity of merge sort is $O(n\log n)$

$n/2^a = 1$

$n = 2^a$

$a = \log n$

↳ used in above equa.

Ques Merge two sorted arrays, using temporary array.

```
#include <bits/stdc++.h>
using namespace std;
void mergeArrays(int arr1[], int n1,
                 int arr2[], int n2)
{
    int newLen = n1 + n2;
    int *temp = new int[newLen];
    int firstIndex = 0;
    int secondIndex = 0;
    int mainIndex = 0;
    while(firstIndex < n1 && secondIn
                -dex < n2)
    {
        if(arr1[firstIndex] < arr2[secon
                -dIndex])
        {
            temp[mainIndex] = arr1[first
                -Index];
            mainIndex++;
            firstIndex++;
        }
        else
        {
            temp[mainIndex] = arr2[secon
                -dIndex];
            mainIndex++;
            secondIndex++;
        }
}
```


Teacher's Sign .....

```
while (firstIndex < n1)
{
    temp[mainIndex] = arr1[first
                          -Index];
    mainIndex++;
    firstIndex++;
}
while(secondIndex < n2)
{
    temp[mainIndex] = arr2[second
                          -Index];
    mainIndex++;
    secondIndex++;
}
cout << "\nAfter merging both the
                arrays : ";
for(int i = 0; i < newLen; i++)
{
    cout << temp[i] << " ";
}
}
int main()
{
    int arr1[] = {3, 4, 9, 12, 14, 18, 21};
    int n1 = 7;
    int arr2[] = {1, 5, 11, 13, 16, 20};
    int n2 = 6;
    cout << "\nFirst array : ";
    for(int i = 0; i < n1; i++)
    {
        cout << arr1[i] << " ";
    }
```

```
    cout << "In Second Array:";
    for(int i=0; i<n2; i++)
    {
        cout << arr2[i] << "  ";
    }
    mergeArrays(arr1, n1, arr2, n2);
    return 0;
}
```