Tutorial Link https://codequotient.com/tutorials/Divide &amp; Conquer - Merge Sort/5a12ea7a46765b2b63e3475a

TUTORIAL

Divide & Conquer - Merge Sort

Chapter

Divide and Conquer: - It is an algorithm design paradigm. A divide and conquer algorithm works by repeatedly breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.Some examples of this paradigm are Binary Search, merge Sort, Quick Sort etc.

Merge Sort

Merge Sort: - It is a sorting technique based on Divide and Conquer paradigm. It is based on the idea of breaking down a list into several sub-lists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list. It uses divide-and-conquer as below:

- Divide the array in two parts by finding the middle position between left and right.
- Conquer by recursively sorting the subarrays in each of the two subproblems created by the divide step.
- Combine by merging the two sorted subarrays back into the single sorted subarray array.

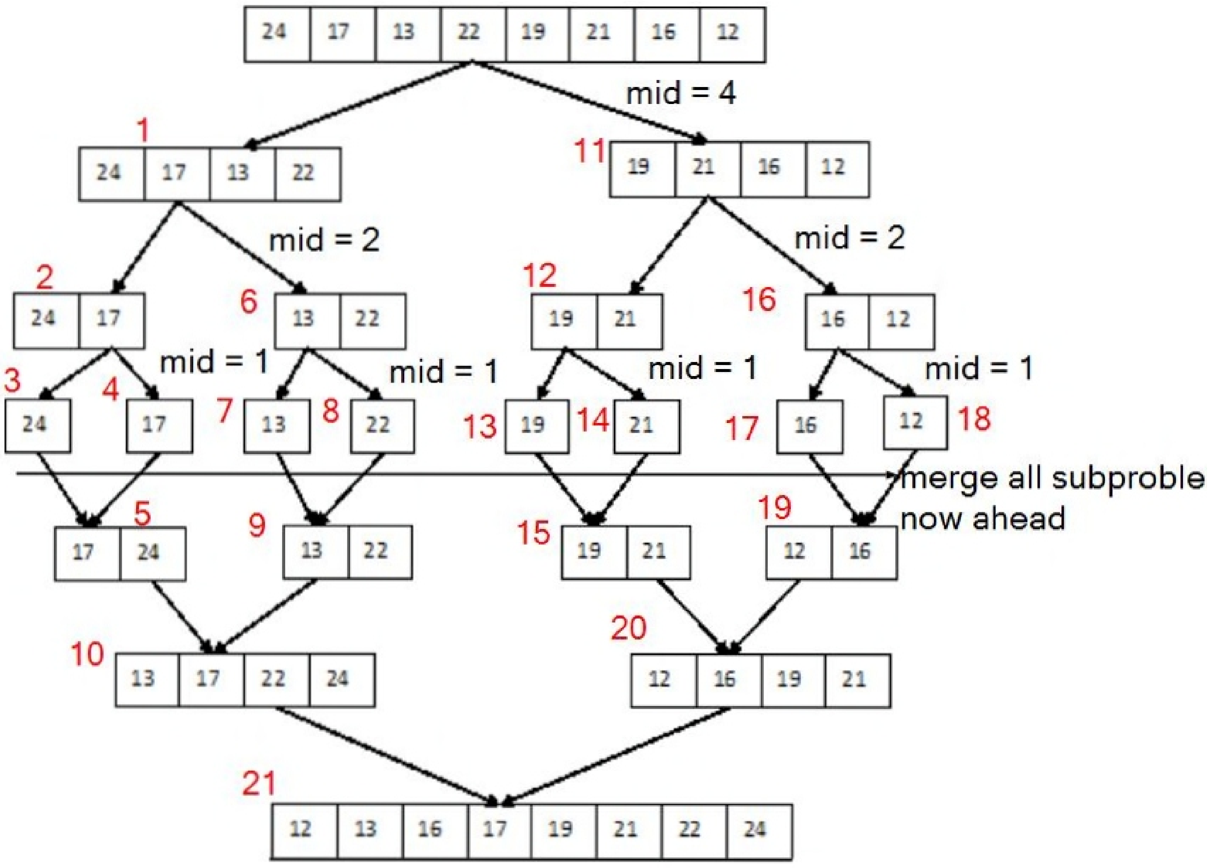
We need a base case. The base case is a subarray containing only one element, that is, when left = right, since a subarray with just one element is already sorted.

Let's see an example.

```
Let us start with array holding [24, 17, 13, 22, 19, 21, 16, 12],
so that the first subarray is actually the full array, array[0..7] (left=0 and right=7).

This subarray has more than two elements, and so it's not a base case.
In the divide step, we compute middle=3 and divide the array in two parts as array1 [0 to 3]
and array2 [4 to 7].

Now array1 [24, 17, 13, 22] is divided using same approach.
It will be divided in two array from middle as array11 [0 to 1] and array12 [2 to 3].

Now array11[24, 17] will be divided in two arrays array111[0 to 0] and array112[1 to 1].
These two arrays are containing 1 element each 24 & 17 respectively, so they are already
sorted.
They will be merged so that array11[0 to 1] will contain items sorted as array11[17, 24] after
merging step.

The algorithm go back to do same steps on array12[13, 22] array then on array2[19, 21, 16, 12]
array.
```

```
And at last, whole array will be sorted in this manner.
```

Following figure shows the step by step procedure for merge sort on this array: -



The numbers in red color are showing the sequence of steps the merge sort procedure will take to sort this array. The algorithm for merge sort is as below: -

```
If left < right
   Find the middle point to divide the array into two halves: m = (l+r)/2
   Call mergeSort for first half
   Call mergeSort for second half
   Merge the two halves sorted in step c and d:
End if
```

Following is the implementation of Merge sort: -

Javascript

```
function print_array(arr){
    console.log(arr.join('\t'));
}

function merge(array,left,mid,right){
    let i, j, k;
    let n1 = mid - left + 1;
    let n2 =  right - mid;

    let Left = new Array(n1), Right = new Array(n2);

    /* Copy data to temp arrays L[] and R[] */

    for (i = 0; i < n1; i++)
        Left[i] = array[left + i];
    for (j = 0; j < n2; j++)
        Right[j] = array[mid + 1+ j];

    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = left; // Initial index of merged subarray
    while (i < n1 && j < n2){
        if (Left[i] <= Right[j])
            array[k++] = Left[i++];
        else
            array[k++] = Right[j++];
    }

    /* Copy the remaining elements of L[], if there are any */
    while (i < n1)
        array[k++] = Left[i++];

    /* Copy the remaining elements of R[], if there are any */
    while (j < n2)
        array[k++] = Right[j++];
}

function mergeSort(array,left,right){
    let i;
    if (left < right){
        let mid = Math.floor(left + (right - left) / 2);
        mergeSort(array, left, mid);
        mergeSort(array, mid+1, right);
        merge(array, left, mid, right);
    }
}

function main(){
    let array = [24, 17, 13, 22, 19, 21, 16, 12];
    let array_size = array.length;
    console.log('Given array is :')
    print_array(array,0,array_size-1);
    mergeSort(array, 0, array_size - 1);
    console.log('Sorted array is :');
    print_array(array,0,array_size-1);
}


main();
```

C

```c
#include<stdio.h>
void print_array(int array[],int left,int right){
    int i;
    for (i=left; i <= right; i++)
        printf("%d\t", array[i]);
    printf("\n");
}

void merge(int array[], int left, int mid, int right){
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 =  right - mid;

    int Left[n1], Right[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        Left[i] = array[left + i];
    for (j = 0; j < n2; j++)
        Right[j] = array[mid + 1+ j];

    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = left; // Initial index of merged subarray
    while (i < n1 && j < n2){
        if (Left[i] <= Right[j])
            array[k++] = Left[i++];
        else
            array[k++] = Right[j++];
    }

    /* Copy the remaining elements of L[], if there are any */
    while (i < n1)
        array[k++] = Left[i++];

    /* Copy the remaining elements of R[], if there are any */
    while (j < n2)
        array[k++] = Right[j++];
}

void mergeSort(int array[], int left, int right){
    int i;
    if (left < right){
        int mid = left + (right - left) / 2;
        mergeSort(array, left, mid);
        mergeSort(array, mid+1, right);
        merge(array, left, mid, right);
    }
}


int main(){
    int array[] = {24, 17, 13, 22, 19, 21, 16, 12};
    int array_size = sizeof(array)/sizeof(array[0]);
    int i;

    printf("Given array is :\n");
    print_array(array,0,array_size-1);
    mergeSort(array, 0, array_size - 1);
    printf("\nSorted array is :\n");
    print_array(array,0,array_size-1);
    return 0;
}
```

Java

```
import java.util.Scanner;

```

```
class Main{
    static void print_array(int array[],int left,int right){
        int i;
        for (i=left; i <= right; i++)
            System.out.print(array[i]+"\t");
        System.out.println();
    }

    static void merge(int array[], int left, int mid, int right){
        int i, j, k;
        int n1 = mid - left + 1;
        int n2 =  right - mid;

        int Left[] = new int[n1], Right[] = new int[n2];

        /* Copy data to temp arrays L[] and R[] */
        for (i = 0; i < n1; i++)
            Left[i] = array[left + i];
        for (j = 0; j < n2; j++)
            Right[j] = array[mid + 1+ j];

        i = 0; // Initial index of first subarray
        j = 0; // Initial index of second subarray
        k = left; // Initial index of merged subarray
        while (i < n1 && j < n2){
            if (Left[i] <= Right[j])
                array[k++] = Left[i++];
            else
                array[k++] = Right[j++];
        }

        /* Copy the remaining elements of L[], if there are any */
        while (i < n1)
            array[k++] = Left[i++];

        /* Copy the remaining elements of R[], if there are any */
        while (j < n2)
            array[k++] = Right[j++];
    }

    static void mergeSort(int array[], int left, int right){
        int i;
        if (left < right){
            int mid = left + (right - left) / 2;
            mergeSort(array, left, mid);
            mergeSort(array, mid+1, right);
            merge(array, left, mid, right);
        }
    }

    public static void main(String[] args){
        int array[] = {24, 17, 13, 22, 19, 21, 16, 12};
        int array_size = array.length;
        System.out.println("Given array is :");
        print_array(array,0,array_size-1);
        mergeSort(array, 0, array_size - 1);
        System.out.println("\nSorted array is :");

        print_array(array,0,array_size-1);
    }
}
```

Python 3

```
def print_array(arr):
    print('\t'.join(str(ele) for ele in arr))

def merge(arr, l, m, r):
        n1 = m - l + 1
        n2 = r- m
        # create temp arrays
        L = [0] * (n1)
        R = [0] * (n2)
        # Copy data to temp arrays L[] and R[]
        for i in range(0 , n1):
                L[i] = arr[l + i]

        for j in range(0 , n2):
                R[j] = arr[m + 1 + j]

        # Merge the temp arrays back into arr[l..r]
        i = 0    # Initial index of first subarray
        j = 0    # Initial index of second subarray
        k = l    # Initial index of merged subarray
        while i < n1 and j < n2 :
                if L[i] <= R[j]:
                        arr[k] = L[i]
                        i += 1
                else:
                        arr[k] = R[j]
                        j += 1
                k += 1

        # Copy the remaining elements of L[], if there
        # are any
        while i < n1:
                arr[k] = L[i]
                i += 1
                k += 1

        # Copy the remaining elements of R[], if there
        # are any
        while j < n2:
                arr[k] = R[j]
                j += 1
                k += 1

def mergeSort(arr,l,r):
        if l < r:
                m = (l+(r-1))//2

                mergeSort(arr, l, m)
                mergeSort(arr, m+1, r)
                merge(arr, l, m, r)


```

```python
if __name__ == '__main__':
    arr = [24, 17, 13, 22, 19, 21, 16, 12]
    n = len(arr)
    print('Given Array is :')
    print_array(arr)
    mergeSort(arr,0,n-1)
    print('\nSorted Array is :')
    print_array(arr)
```

C++

```cpp
#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;
void print_array(int array[],int left,int right){
    int i;
    for (i=left; i <= right; i++)
        cout<<array[i]<<'\t';
    cout<<endl;
}

void merge(int array[], int left, int mid, int right){
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 =  right - mid;

    int Left[n1], Right[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        Left[i] = array[left + i];
    for (j = 0; j < n2; j++)
        Right[j] = array[mid + 1+ j];

    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = left; // Initial index of merged subarray
    while (i < n1 && j < n2){
        if (Left[i] <= Right[j])
            array[k++] = Left[i++];
        else
            array[k++] = Right[j++];
    }

    /* Copy the remaining elements of L[], if there are any */
    while (i < n1)
        array[k++] = Left[i++];

    /* Copy the remaining elements of R[], if there are any */
    while (j < n2)
        array[k++] = Right[j++];
}

void mergeSort(int array[], int left, int right){
    int i;
    if (left < right){
        int mid = left + (right - left) / 2;
        mergeSort(array, left, mid);
        mergeSort(array, mid+1, right);
        merge(array, left, mid, right);
    }
}


int main(){
    int array[] = {24, 17, 13, 22, 19, 21, 16, 12};
    int array_size = sizeof(array)/sizeof(array[0]);
    int i;

    cout<<"Given array is :\n";
    print_array(array,0,array_size-1);
    mergeSort(array, 0, array_size - 1);
    cout<<"\nSorted array is :\n";
    print_array(array,0,array_size-1);
    return 0;
}

```

Output: -

```
Given array is :
24   17   13   22   19   21   16   12

Sorted array is :
12   13   16   17   19   21   22   24
```

The output above shows the divide and merge step in each iteration. The algorithm always divide the original array into two halves. The recurrence equation for merge sort is as below: -

```
T(n) = 2.T(n/2) + n
```

Every time 2 problems of size n/2 are done so 2.T(n/2) and at each step n elements to be merged so n is added at end.

We can solve this equation using any recurrence solving method and complexity of this algorithm is **O(n.Log n)**.

Properties of Merge sort

Worst and Average Case Time Complexity: O(n Log n).

Best Case Time Complexity: O(n Log n). No specific best case for Merge sort.

Auxiliary Space: O(n), as we have to divide the array into two halves and repeat the procedure.

Sorting In Place: Yes

Stable: Yes

It is popularly used when sorting data is huge in size as it can be performed as an external sort easily. Also when data is not in form of array for example, to sort two linked list of size m,n respectively merge sort is popularly used.