Merge sort

1. Split array A[0..*n*-1] in two about equal halves and make copies of each half in arrays B and C
2. Sort arrays B and C recursively
3. Merge sorted arrays B and C into array A as follows:

- Repeat the following until no elements remain in one of the arrays:
  – compare the first elements in the remaining unprocessed portions of the arrays
  – copy the smaller of the two into A, while incrementing the index indicating the unprocessed portion of that array
- Once all elements in one of the arrays are processed, copy the remaining unprocessed elements from the other array into A.

Merge sort algorithm

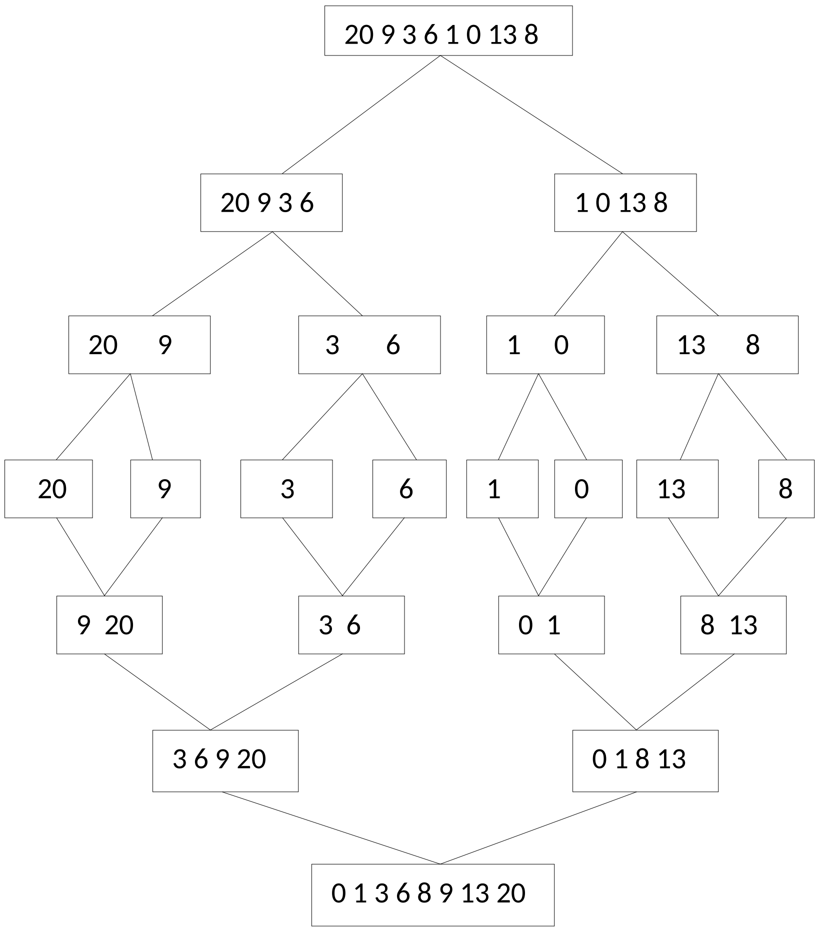
```
Merge_sort (arr , start_index, last_index )
{
if(start_index== last_index)
        Return(arr[start_index]);
else {
        Mid=floor((start_index+last_index)/2);
        Merge_sort(arr, start_index, mid);
        Merge_sort(arr, mid+1, last_index);
        Merge(arr, start_index, mid, last_index);
    }
}
```

Merge algorithm

```
Merge(arr, start_index, mid,last_index)
{  k=mid+1, p=1;
   While(start_index<=mid && k<=last_index)
{          if(arr[start_index]<arr[k]) {
                Brr[p]=arr[start-index];
                start_index++;
                P++;
        }
else
        {       Brr[p]=arr[k];
                k++;
                P++;
        }

}
for(; start_index<=mid;start_index++)
        {       Brr[p]=arr[start_index]
                  P++;
        }
for(; k<=last_index; k++)
        {       Brr[p]=arr[k];
                P++;
        }
for(k=1; k<=last_index; k++)
        {
                arr[k]=Brr[k];
        }
}
```

Example of merge sort

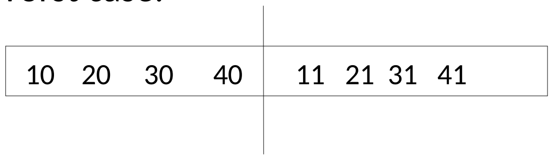


Merge procedure analysis

Input: two sorted sub array

Output: single sorted array

Worst case:



Min(10,11)
Min(20,11)
Min(21,20)
Min(30,21)
Min(30,31)
Min(40,31)
Min(40,41)

total no. of comparison=(4+4-1)=7 comparison.

worst case time complexity of merge procedure

m+n-1 = **O(m+n)** where m & n is the size of sub-array

When both the sub-array size is equal then time complexity

n/2+n/2-1=2n=**O(n)** (neglect constant)

Best case:

10 20 30 40	1 2 3 4

Min(10,1)
Min(10,2)
Min(10,3)
Min(10,4)

Best case time complexity is no. of comparison

- if 1 part of array has size m & 2 part of array has size n then time complexity O(min(m, n))
- If size of sub- arrays is n/2 then time complexity **(n)**

Note: if we don’t use second array for merge procedure then time complexity of merge procedure of two sorted sub-array will increase.

Merge sort can be both in-place or outplace but in outplace time complexity is less as compare to in-place because of usage of second array for merge procedure.

Worst case: **O(n)**
Best case: **(n)**
Average case: **(n)**

Time complexity :

Merge sort recurrence relation equation

$$T(n)=\begin{cases} O(1) & \text{if } n=1 \\ \underbrace{O(1)}_{\text{Divide cost}} + \underbrace{T(n/2) + T(n/2)}_{\text{2 sub-array}} + \underbrace{\square(n)}_{\text{conquer cost/ merge cost}} & \text{if } n>1 \end{cases}$$

$T(n)=2T(n/2) + n$ ….. neglecting constant time

Solved using recurrence relation solving technique. We get time complexity of merge sort **O(nlogn) for outplace sorting.**

Space complexity

For merge sort : $\underbrace{n}_{\text{No. of element in array}} + \underbrace{c\log_2 n}_{\text{stack size (no. of function calls, where c is no of variable size in each function)}} + \underbrace{n}_{\text{array size of b array}}$

space complexity of merge sort**=O(n)**

Note: If array size is small then merge sort is not recommended. Merge sort is used for large size array.