COURSE HANDOUT

Course Code	ACSC13
Course Name	Design and Analysis of Algorithms
Class / Semester	IV SEM
Section	A-SECTION
Name of the Department	CSE-CYBER SECURITY
Employee ID	IARE11023
Employee Name	Dr K RAJENDRA PRASAD
Topic Covered	Merge Sort
Course Outcome/s	Use the merge sort for sorting of elements and find the time analysis
Handout Number	19
Date	21 April, 2023

Content about topic covered: Merge sort

Merge sort

Given a sequence of n elements a[1], … , a[n]. The idea is to split in to two sets a[1], … ,a[n/2] and a[(n/2)+1], … , a[n]. Each set is individually sorted, and the resulting sorted sequences are merged to produce a single sorted sequence of n elements.

```
Algorithm MergeSort(low, high)
// a[low : high] is a global array to be sorted.
// Small(P) is true if there is only one element
// to sort. In this case the list is already sorted.
{
    if (low < high) then  // If there are more than one element
    {
        // Divide P into subproblems.
            // Find where to split the set.
                mid := ⌊(low + high)/2⌋;
        // Solve the subproblems.
            MergeSort(low, mid);
            MergeSort(mid + 1, high);
        // Combine the solutions.
            Merge(low, mid, high);
    }
}
```

```
Algorithm Merge(low, mid, high)
// a[low : high] is a global array containing two sorted
// subsets in a[low : mid] and in a[mid + 1 : high]. The goal
// is to merge these two sets into a single set residing
// in a[low : high]. b[ ] is an auxiliary global array.
{
    h := low; i := low; j := mid + 1;
    while ((h ≤ mid) and (j ≤ high)) do
    {
        if (a[h] ≤ a[j]) then
        {
            b[i] := a[h];h := h + 1;
        }
        else
        {
            b[i] := a[j]; j := j + 1;
        }
        i := i + 1;
    }
    if (h > mid) then
        for k := j to high do
        {
            b[i] := a[k]; i := i + 1;
        }
    else
        for k := h to mid do
        {
            b[i] := a[k]; i := i + 1;
        }
    for k := low to high do a[k] := b[k];
}
```

Example of quicksort:

Eg: a[l:10]= (310, 285, 179, 652, 351, 423, 861, 254, 450, 520)

(310| 285| 179| 652,351| 423,861,254, 450,520)

Elements a[l]and a[2] are merged to yield

(285,310| 179| 652,351| 423,861,254, 450,520)

Then a[3] is merged with a[l:2]

(179,285,310| 652,351| 423,861,254, 450,520)

Next, elements a[4] and a[5] are merged

(179,285,310| 351,652| 423,861,254, 450,520)

and then a[l:3] and a[4 :5] are merged

(179,285,310,351,652| 423,861,254, 450,520)

(179,285,310,351,652| 423 | 861| 254 | 450,520)

Elements a[6] and a[7] are merged. Then a[8] is merged with a[6:7]:

(179,285,310,351,652| 254, 423,861| 450,520)

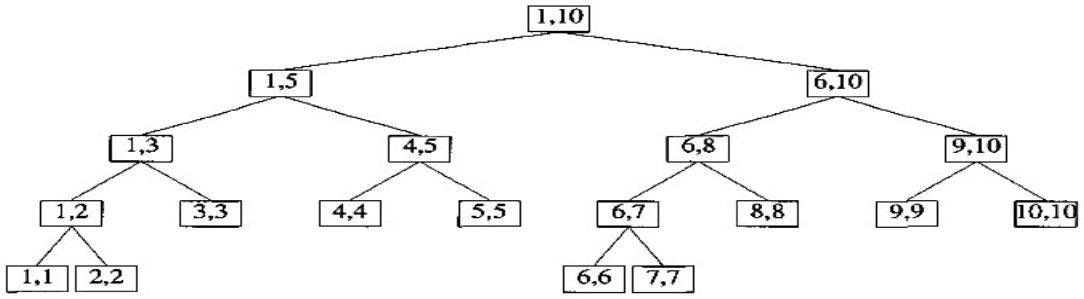
Next a[9]and a[10] are merged, and then a[6:8] and a[9:10] are merged

(179,285,310,351,652| 254, 423,450,520,861)

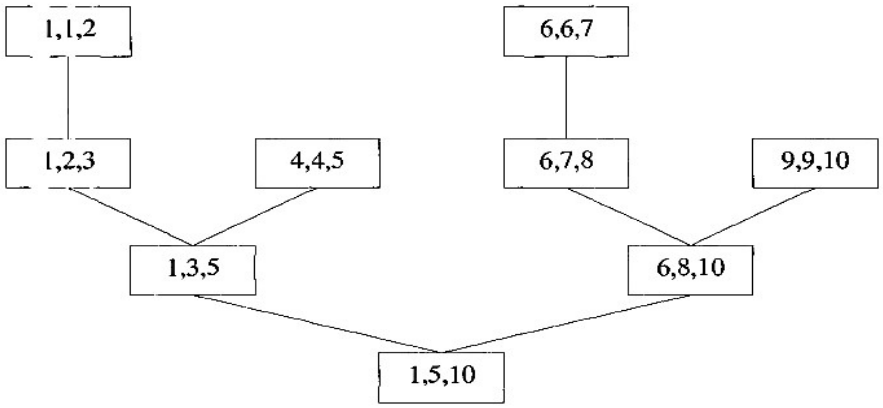
At this point there are two sorted sub arrays and the final merge produces the fully sorted result

(179, 254, 285,310, 351, 423, 450, 520, 652, 861).

Tree of calls of Merge_Sort(1,l0) are shown below:



Trees of calls of Merge are shown below:



If the time for the merging operation is proportional to ‘n’, then the computing time for merge sort is described by the recurrence relation

$$T(n) = \begin{cases} a & n = 1, \ \ a \ is \ constant \\ 2\,T\left(\frac{n}{2}\right) + cn & n > 1, \ \ c \ is \ constant \end{cases}$$

$$
\begin{aligned}
T(n) &= 2\,T\left(\frac{n}{2}\right) + cn \\
&= 2\left[2\,T\left(\frac{n}{4}\right) + \frac{cn}{2}\right] + cn \\
&= 4\,T\left(\frac{n}{4}\right) + 2cn \\
&= 4\left[2\,T\left(\frac{n}{8}\right) + \frac{cn}{4}\right] + 2cn \\
&= 8\,T\left(\frac{n}{8}\right) + 3cn
\end{aligned}
$$

When n is power of 2, $n = 2^k$

$$
\begin{aligned}
T(n) &= 2^k\,T(1) + k\,c\,n \\
&= a\,n + c\,n \log n
\end{aligned}
$$

Disadvantage of Merge sort:

Merge sort uses 2n locations. The additional n locations are needed because we could not merge two sorted sets in place. On each call of Merge the values from array b[] are copied back to array a[].

To avoid this advantage, we use QUICKSORT.∴ The time complexity T(n) = O(n log n).