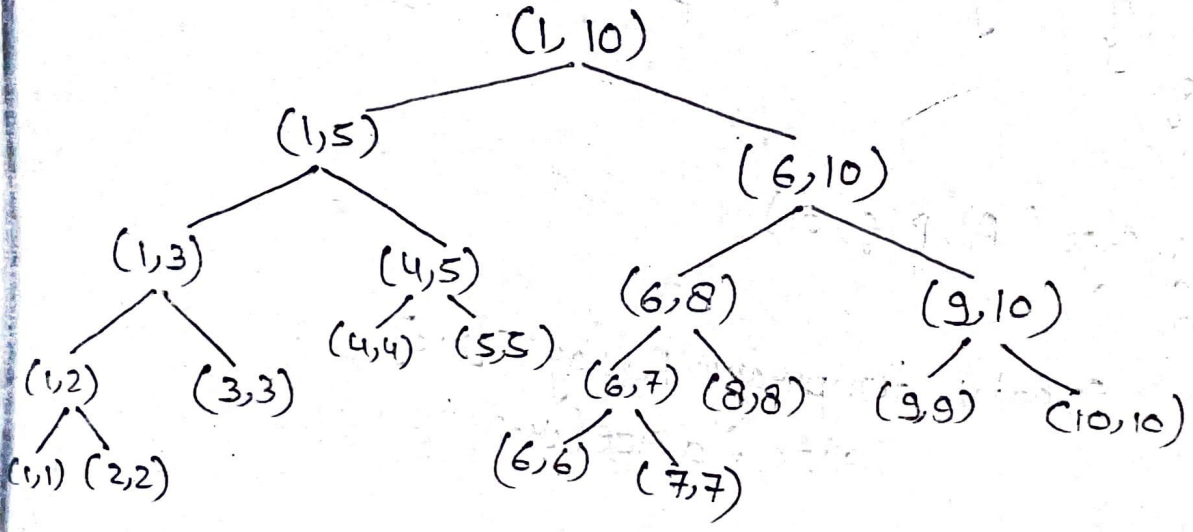
MERGE SORT →

Divide – Divide the input of array into two subarrays of equal size

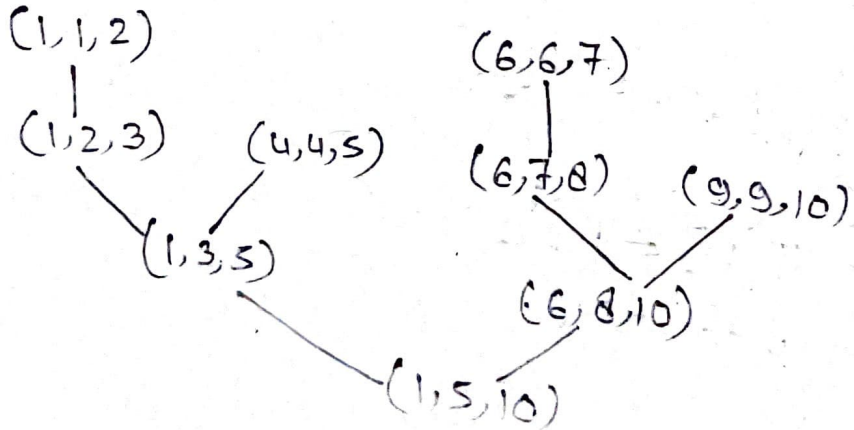
Conquer
~~Compare~~ – Recursively sort each sub array by applying DAC

Combine – merge sorted sol. of each sub array to get the final sorted list.

Partitioning of list using Merge sort.



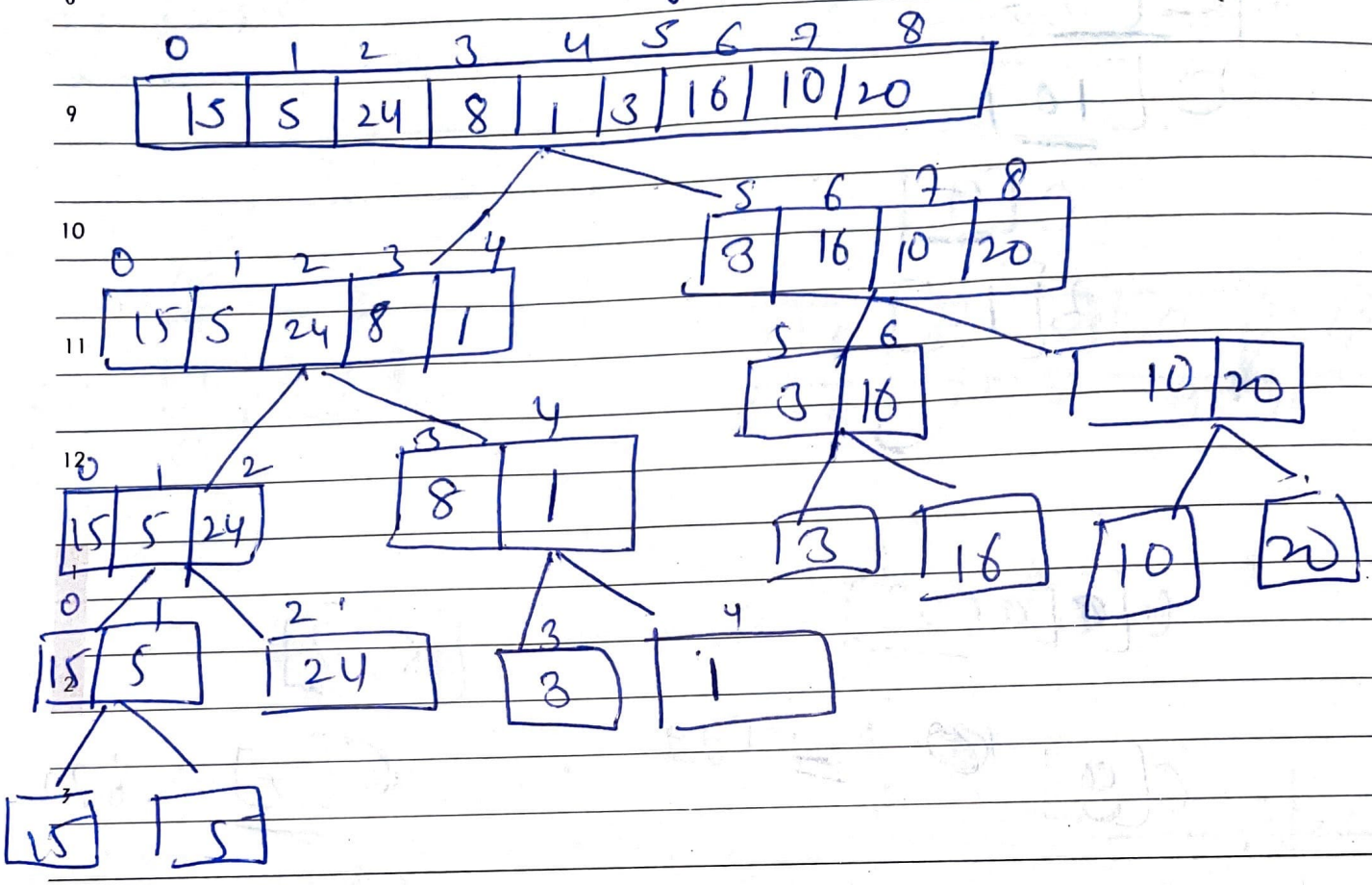
Tree calls of merger



APRIL '13
S M T W T F S
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

Appointments / Meetings

Merge Sort

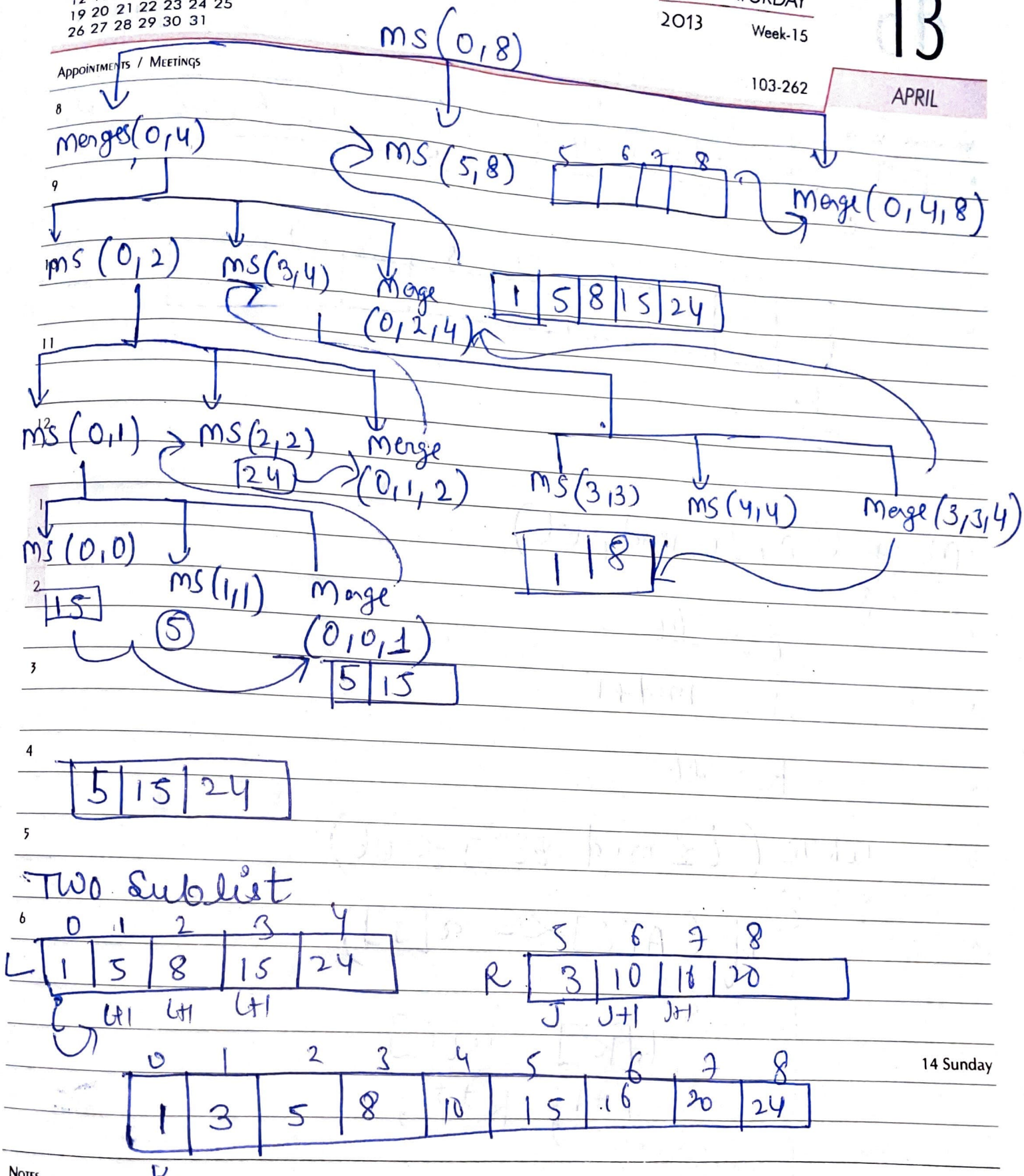


Untill each sub-list has one element

```
MergeSort (A, lb, ub)
{ if (lb < ub)
  { mid = (lb + ub)/2
    Merge
    MergeSort(A, lb, mid);
    MergeSort(A, mid+1, ub);
    Merge (A, lb, mid, ub)
  }
}
```

Notes

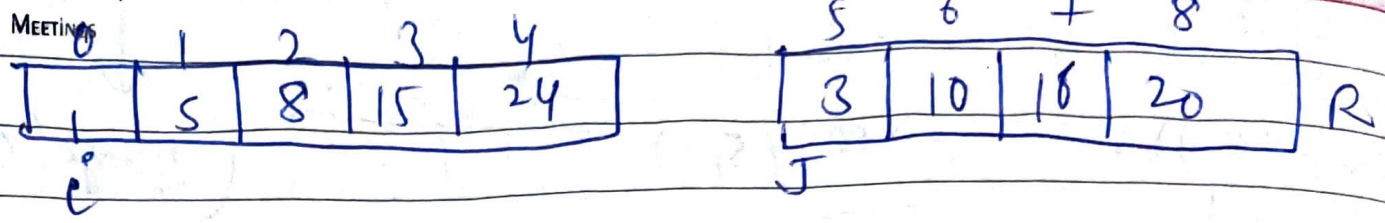
Appointments / Meetings

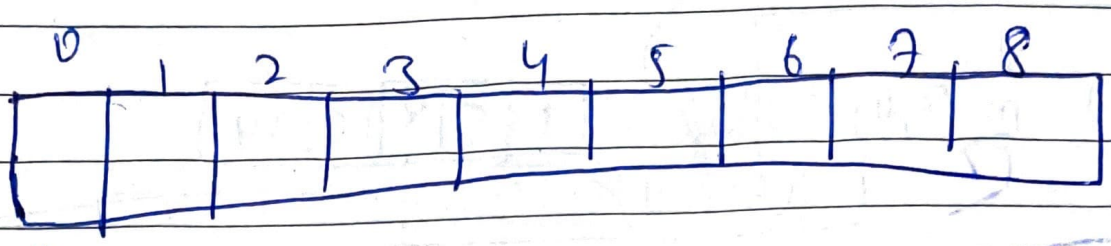


Two Sublist

14 Sunday

Notes






```
Merge ( A, lb, mid, ub)
{
    i = lb
    J = mid+1
    K = lb
while ( i <= mid && J <= ub)
{
if ( A[i] <= a[J])
    {
        b[K] = a[i]
        i++; k++;
    }
else
    {
        b[K] = a[J]
        J++; k++
} }
```

Notes

```
if (i > mid)
{
    while (j <= ub)
    {
        b[k] = a[j]
        j++; k++
    }
}
else {
    while (i <= mid)
    {
        b[k] = a[i]
        i++; k++;
    }
}

for (k=lb; k<=ub; k++)
{
    a[k] = b[k]
}
```

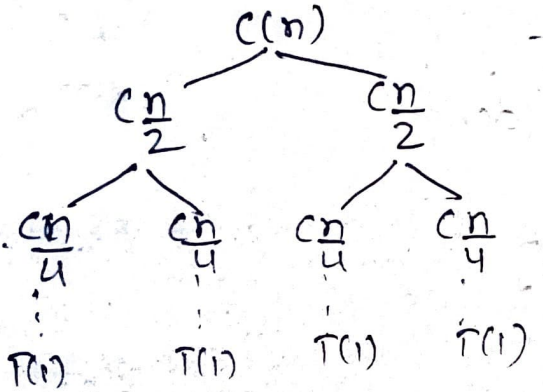
$O(n \log n)$

Notes

If the time of merging operation is proportion to n then the computing time for merge sort is described by recurrance relation.

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

★ $T(n) = 2T(n/2) + C(n)$



Height of Tree

$$\frac{n}{2^i} = 1$$

$$2^i = n$$

$$i = \log_2 n$$

No. of levels $= i+1$

$= \log_2 n + 1$

No. of nodes at last level. 2^i

$$2^{\log_2 n} \Rightarrow n^{\log_2 2} = n$$

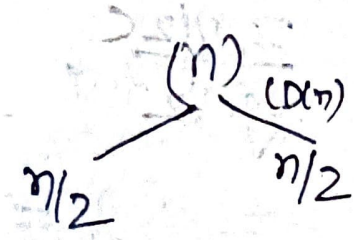
Total cost $= cn + \frac{2cn}{2} + \frac{4cn}{4} + \cdots C(\log_2 n - 1) + \Theta(n)$

$= cn(1+1+\cdots \log n \text{ times}) + \Theta(n)$

$= cn^2 + \Theta(n) - cn$ $cn(\log n)$

$= \Theta(n^2)$ $\Theta(n \log n)$

Analysis of Merge Sort



Divide

In Divide Step, Merge sort just take a const. time C_1 for calculating the middle index of input array.

Conquer

In this step, we have 2 subproblem and it contributes $2T(n/2)$ time for total running time.

Combine / Merge –

$n=6$ [2|6|7] [4|5|8] [1|3|8]

1st [1|2|3|4|5|6|7|8]

It take n time. $O(n)$

And if 1st array is smaller then 2nd.

[1|2|3] [4|5|6]

1st [1|2|3|4|5|6] It takes $n/2$ time.

Hence we can say that merge problem is $O(n)$

Let $H(n)$ is time when smaller problem is done.

Therefore recurrance relation for merge sort

$$T(n) = \begin{cases} 2T(n/2) + \theta(1) + \theta(n) & \text{if } n > 1 \\ H(n) & \text{if } n = 1 \end{cases}$$

Replace

$$T(n) = \begin{cases} 2T(n/2) + \theta(n) & \text{if } n > 1 \\ c_2 & \text{if } n = 1 \end{cases}$$

$$T(n) = 2T(n/2) + c_1(n)$$

Here $= a = 2,\ b = 2,\ f(n) = n$

$$n^{\log_b a} \Rightarrow n^{\log_2 2} = n$$

here $f(n) = n \log_b a$

Apply II case.

$$T(n) = \theta(n^{\log_b a} \lg^k n) \quad [k = 0]$$

$$T(n) = \theta(n \lg n)$$

Quick Sort :- (Example of DAC)

→ Best Algorithm for sorting.

→ Sorting in place, means no need of any additional array.

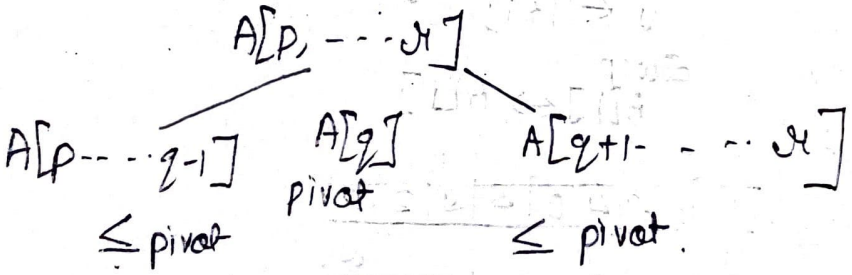
→ Very practical, very fast in average case performance.

→ Expected / Avg. Running $O(n \lg n)$ with small constant factor.

→ Based on DAC.

Divide:- Partition / Rearrange input array into two subarrays, such that every element in first array is less or equal $A[q]$ (pivot) and in second subarray, such that every element is greater or equal to $A[q]$

[Subarray :- $A[p, \cdots q-1]$ and $A[q+1, \cdots r]$]



Compute the index q as the result of partition.

Conquer :- Recursively call quicksort for each subarray.

Combine – No need to combine because sorting in place.

→ Height of Tree = log n
→ Every level we trave all node n

log n x n | n log n

Appointments / Meetings

Quick Sort

(Partition exchange method)

→ Based on pivot element
→ Pivot element could be {first, middle or last} element
→ all values Smaller then Pivot → Placed left of Pivot
→ " " Greater then " → Placed right of Pivot

lb = 0, ub = 8

0	1	2	3	4	5	6	7	8
7	6	10	5	9	2	1	15	7

start, end

Pivot = 7

7	6	7	5	9	2	1	15	10

start, end

7	6	7	5	1	2	9	15	10

~~end start~~

2	6	7	5	1	(7)	9	15	10

```
Partition (A, lb, ub)
{
  Pivot = a[lb]
  start = lb
  end = ub
  while (start < end)
  {
    while [a[start] ≤ Pivot)
    { start++
    }
    while [a[end] > Pivot)
    {
      { end--
      }
    }
    if (start < end)
    { swap (a[start], a[end])
    }
  }
  swap (a[lb], a[end])
  returned end;
}
```

Notes

Complexity = O | n(log n)

MONDAY

Week-03 2013

014-351

```
Quicksort(A, lb, ub)
{
  if(lb < ub)
  {
    loc = Partition(A, lb, ub);
    Quicksort(A, lb, loc-1);
    Quicksort(A, loc+1, ub);
  }
}
```

/* Bubble Sort

Analysis of Quick sort:-

It depends upon weather the partion balanced or unbalanced. Thats why its depend upon which element is choosen as pivote.

* Best case Partioning →

Suppose we are lucky, we always get a partion of size $\lfloor n/2 \rfloor$ and $\lfloor n/2 \rfloor - 1$

Divide: for loop is run n times

hence $D(n) = \Theta(n)$

Conquer.

$$T(n) \leq 2T(n/2) + \Theta(n)$$

Solve it by master method.

here $\Rightarrow a=2, \quad b=2$

$$n^{\log_b a} = n^{\log_2 2} = n$$

Here I case is applied.

$$T(n) = \Theta(n \lg n)$$

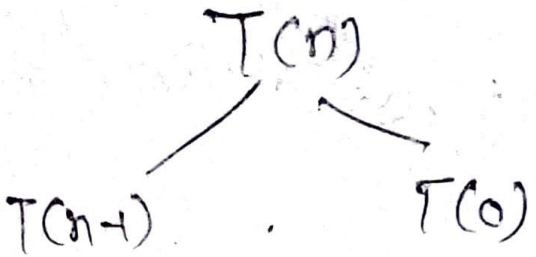
* Worst case Partioning → If we are not lucky, we always get a partion which is fairly unbalanced.

It occurs when we choose

→ Smallest no. in the array

→ Largest no. in the array

In this case we get 1 half of size n-1 elm
and second half of size 0 element.



Thats why we get T(n)

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

$$T(n) = T(n-1) + \Theta(n)$$

Solve it by Iteration method.

$$T(n) = \Theta(n^2)$$