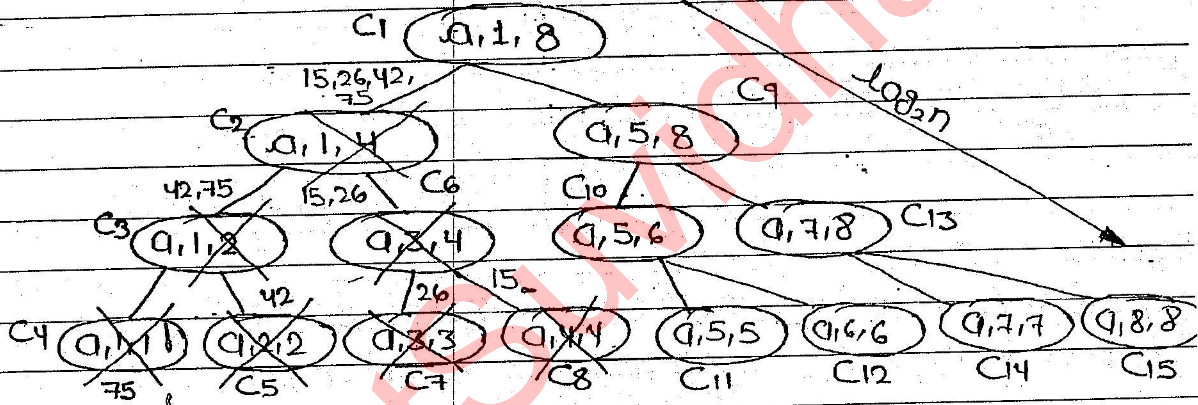
Page No. 8
Date : 21/02/16

IV Merge Sort -

Merging two sorted sub-arrays
i/p: an array of n-elements
o/p: sorted array
smaller-problem: 1-element

ex

1	2	3	4	5	6	7	8
75	42	26	15	99	84	16	21



Push - calling function
Pop - returning f^n

$\log_2 n$

~~C4~~	~~C5~~	~~C7~~	~~C8~~	C11	C12	C14	C15
~~C3~~	~~C6~~	C10	C13				
~~C2~~	C9						
C1							

Stack representation

Merge algorithm
i/p - two sorted sub-arrays
o/p - one sorted array

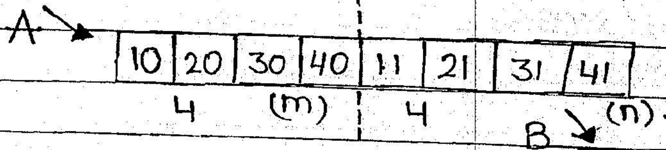
Merge-sort is a super-set of merge.

Combine step.

StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

In the algorithm -

1. If



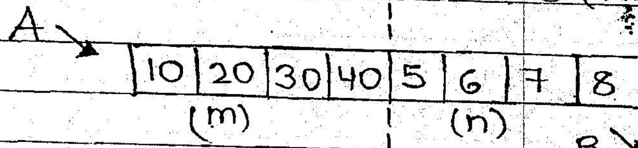
Moves = m+n

Comparisons = m+n-1

[WORST CASE] Out-place

Time Complexity = moves + comparisons

= O(m+n).



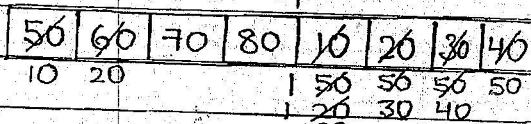
Moves = m+n

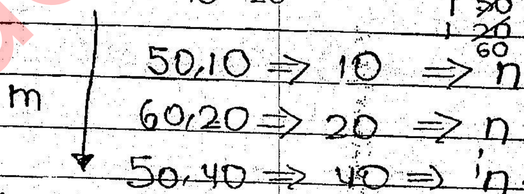
Comparisons = min (m, n) [BEST CASE]

Time complexity = moves + comparisons

= O(m+n) Out-place.

Merging two sub-arrays (sorted) each of size m and n will take O(m+n)





In-place

worst-case algorithm. ⇒ O(m.n).

If m=n O(n²)

Download all Notes and papers from StudentSuvidha.com

Merge-sort algorithm (a, i, j)

```
{
  if (i==j)
    then return a[i];                    } -> const.
  else {
    mid = (i+j)/2;
    merge-sort (a, i, mid);              T(n/2)
    merge-sort (a, mid+1, j);            T(n/2)
    merge (a, i, mid, mid+1, j)          O(n)   Every case
  }
}
```

Recurrance relation - Let T(n) be time complexity of merge-sort algorithm on n-elements

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n) + O(1) & \text{if } n > 1 \end{cases}$$

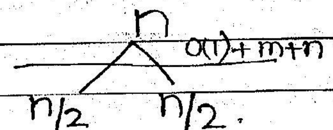
$$T(n) = 2T\left(\frac{n}{2}\right) + n \rightarrow \text{(divide + combine)}.$$

$$= 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 2^2 T\left(\frac{n}{2^2}\right) + n + n$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right] + n = 2^3 T\left(\frac{n}{2^3}\right) + n + n + n$$

$$\vdots$$

$$= 2^K T\left(\frac{n}{2^K}\right) + n \cdot K$$



$$K = \log_2 n$$

$$T(n) = 2^{\log_2 n} T(1) + n\log_2 n$$

$$= n + n\log_2 n$$

$$= O(n\log_2 n)$$

StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

Merge-sort (Inplace).

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

$$T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + \frac{n^2}{2^2}\right] + n^2$$

$$= 2^2T\left(\frac{n}{2^2}\right) + \frac{n^2}{2} + n^2$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{n^2}{2^4}\right] + \frac{n^2}{2} + n^2$$

$$= 2^3T\left(\frac{n}{2^3}\right) + \frac{n^2}{2^2} + \frac{n^2}{2} + \frac{n^2}{1}$$

$$= 2^kT\left(\frac{n}{2^k}\right) + n^2\left[\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots 1\right]$$

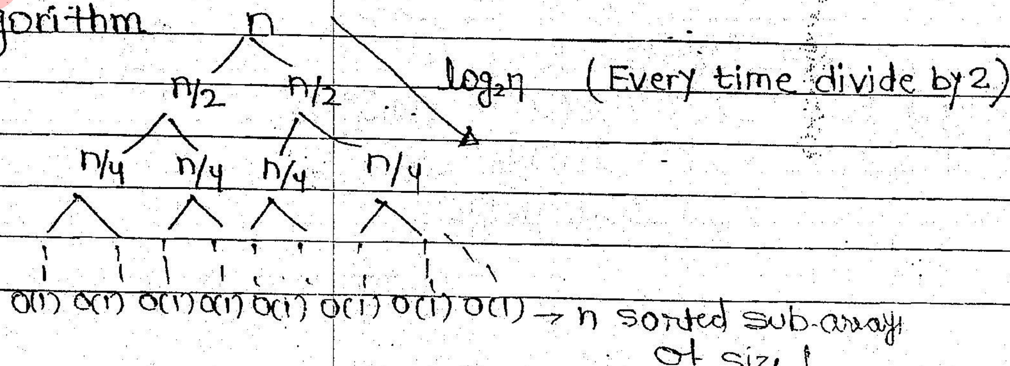
$$k = \log_2 n$$

$$T(n) = 2^{\log_2 n} T(1) + n^2\left[\frac{1}{2^0} + \frac{1}{2^1} + \cdots \frac{1}{2^{\log_2 n - 1}}\right]$$

$$= n + n^2\left[\frac{1\left[1 - (1/2)^{\log_2 n}\right]}{1 - 1/2}\right]$$

$$= O(n^2)$$

Merge-sort algorithm is out-place algorithm bcoz in merge we are taking another array
If the array size is very large - merge sort else insertion sort is advisable.
Stable algorithm

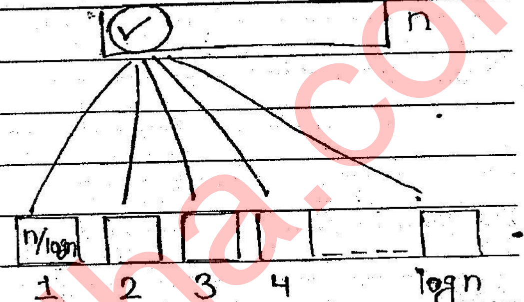


Download all Notes and papers from StudentSuvidha.com

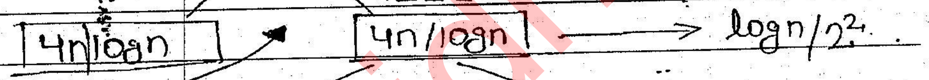
Day-5

Ex 1

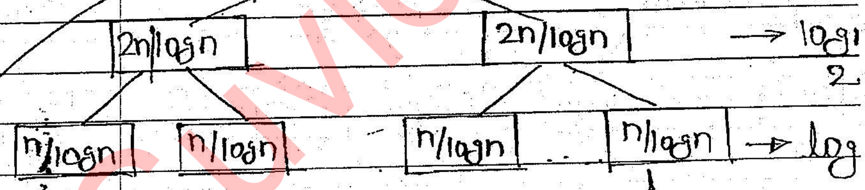
i/p : logn sorted sub-arrays each of size n/logn
o/p : sorted array of size n.
Best algorithm, worst case.



Direct merging
$O(n \log n)$.
$\log n + \log n + \log n + \dots$ n times



Divide & Conquer
log(logn)



$\frac{2n}{\log n} \times \frac{\log n}{2} = O(n)$ | $\log n \times \frac{n}{\log n} = O(n)$.

K times. $\frac{2^K n}{\log n} \times \frac{\log n}{2^K} = O(n)$

$\frac{\log n}{2^K} = 1 \Rightarrow K = \log_2 \log_2 n$

Total cost $O(n \log_2 \log_2 n)$ [Divide & Conqueor]

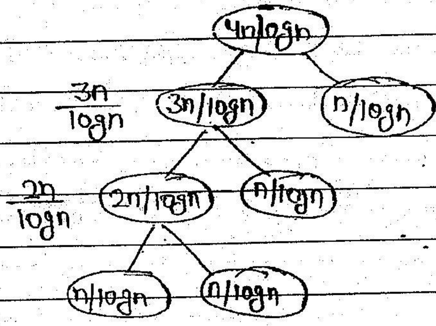
Merge algorithm.

Taking one at a time

$\frac{n}{\log n}[1+2+3+\dots \log n]$

$\frac{n}{\log n}\left[\frac{\log n(\log n+1)}{2}\right]$

$= O(n \log n)$



Download all Notes and papers from StudentSuvidha.com

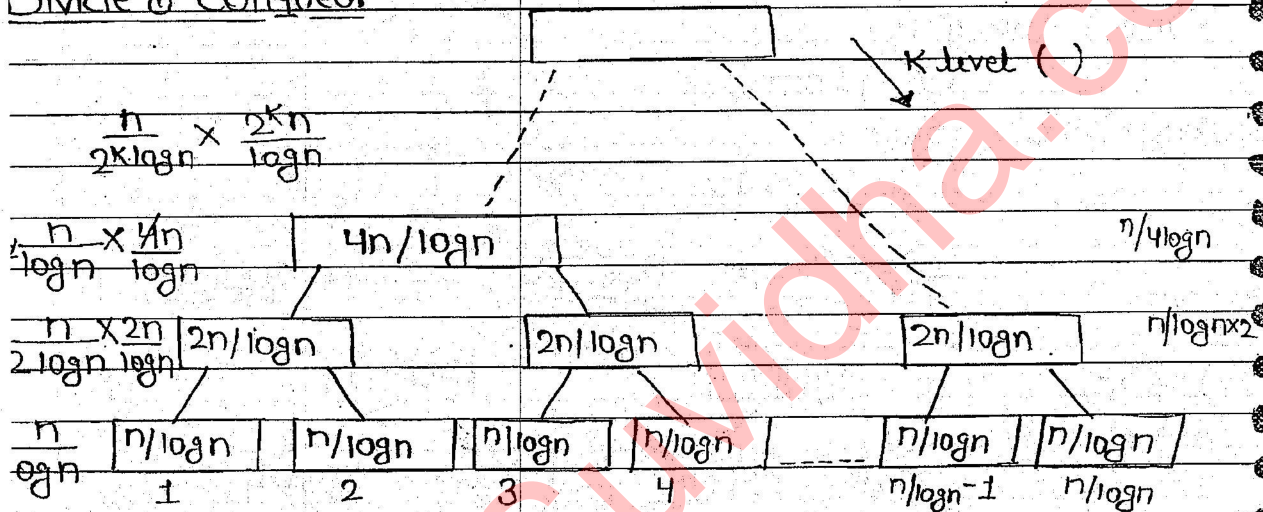
Page No. 2
Date : 27/02/16

Ex-2

I/P: $\frac{n}{\log n}$ sub-arrays each of size $\frac{n}{\log n}$

O/P: single sorted array.

Divide & Conquer



$$\frac{n}{2^K \log n} = 1$$

$$\frac{n}{\log n} = 2^K \Rightarrow K = \log_2\left[\frac{n}{\log n}\right]$$

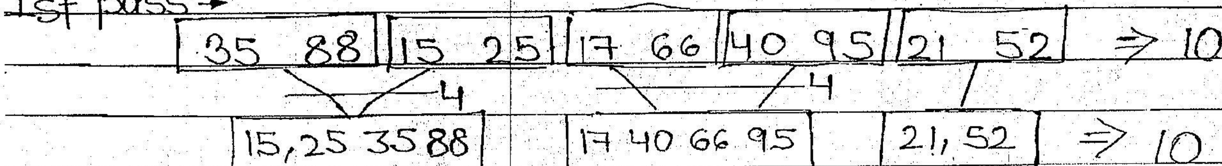
Total cost $O\left(\frac{n^2}{(\log n)^2} \log_2 \frac{n}{\log n}\right)$

Ex-3. Consider the following array

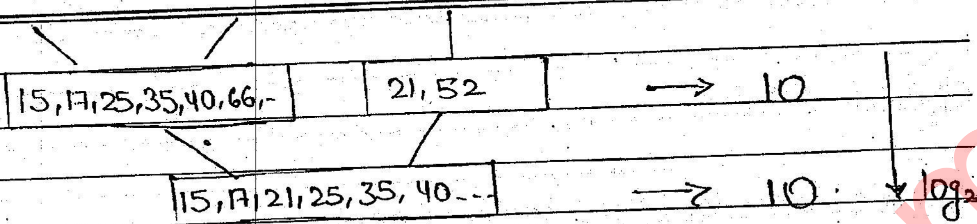
88, 35, 25, 15, 66, 17, 95, 40, 52, 21

What will be the O/P after second pass of straight two-way merge sort algorithm.

Ist pass →



Download all Notes and papers from StudentSuvidha.com



Ex.4 I/p : 2-sorted sub-arrays A → m B → n
O/p : find A∩B, A∪B

Intersection -

I Linear search → O(mn) II Binary search → O(m log n)

III Merge algorithm → skip the smaller one
if equal, then print. Complexity same.
O(m+n)

UNION -

I Linear search → O(mn) II Binary search O(n log m)

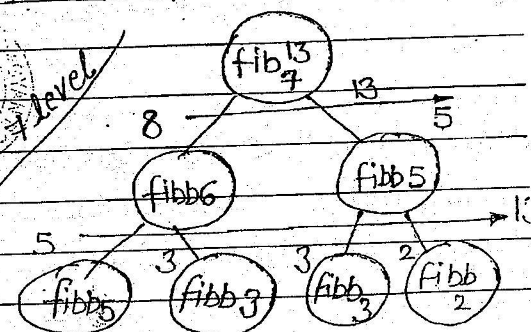
III Merge algorithm → Print one time, skip both if equal
otherwise print both O(m+n).

Ex.5

n	0	1	2	3	4	5	6	7
fib(n)	0	1	1	2	3	5	8	13

7-level O(7 · fibb7)

K-level O(K · fibbK)



Divide time - constant
fibbonacci merge-sort
↳ divide fib series manner
Combine O(m+n)

Download all Notes and papers from StudentSuvidha.com

Page No. 4
Date : 27/02/16

V. QUICK-SORT

1. Application of Divide & Conqueor.
2. Inplace algorithm (that's why Quick)
3. Not stable

Decreasing GP series takes constant time

Before	20	10_a	15	10_b	45
After	10_b	10_a	15	20	45

↳ repeated element order may change.

4. Most practical used sorting algorithm.
5. If array already sorted, then $O(n^2)$

40	25	70	90	35	16	44	88	15	72	10	55
1	(2	3	4	5	6	7	8	9	10	11	12)

↓ Divide meaningfully

(10 15 16 35 25) 40 (70 90 44 88 72 55)

↓ sort ↓ sort (conqueor)

[10 15 16 25 35] 40 [44 55 70 72 88 90]

Pivot (40)

No combine

Partial divide & conqueour application

Divide time n (n comparisons)

Conqueor time $2T(n/2)$

$$2T(n/2) + n \Rightarrow O(n \log n)$$

for merge sort $2T(n/2) + n + c$

$$\Rightarrow O((n+c)\log n)$$

StudentSuvidha.com

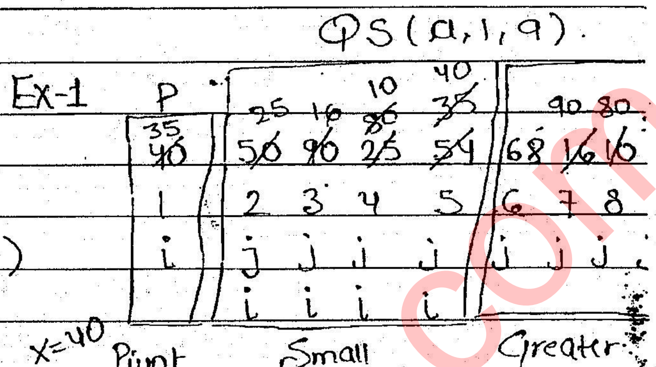
Download all Notes and papers from StudentSuvidha.com

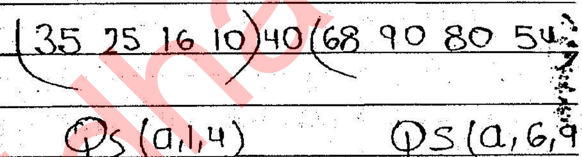
Partition algorithm

```
partition (a,p,q)
{
  x = a[p];
  i = p;
  for (j=i+1; j<=q; j++)
  {
    if (a[j] <= x)
    {
      i++;
      swap (a[i], a[j]);
    }
  }
  swap (a[i], a[p]);
  return (i);
}
```

QS(a,1,9).

Ex-1





return (a,1,9)

```
Quicksort (A,p,q) {
m = partition (a,p,q)
  Quicksort (A,p,m);
  Quicksort (A,m+1,q);
}
```

comparisons n-1 (BC, WC, AC). O(n)

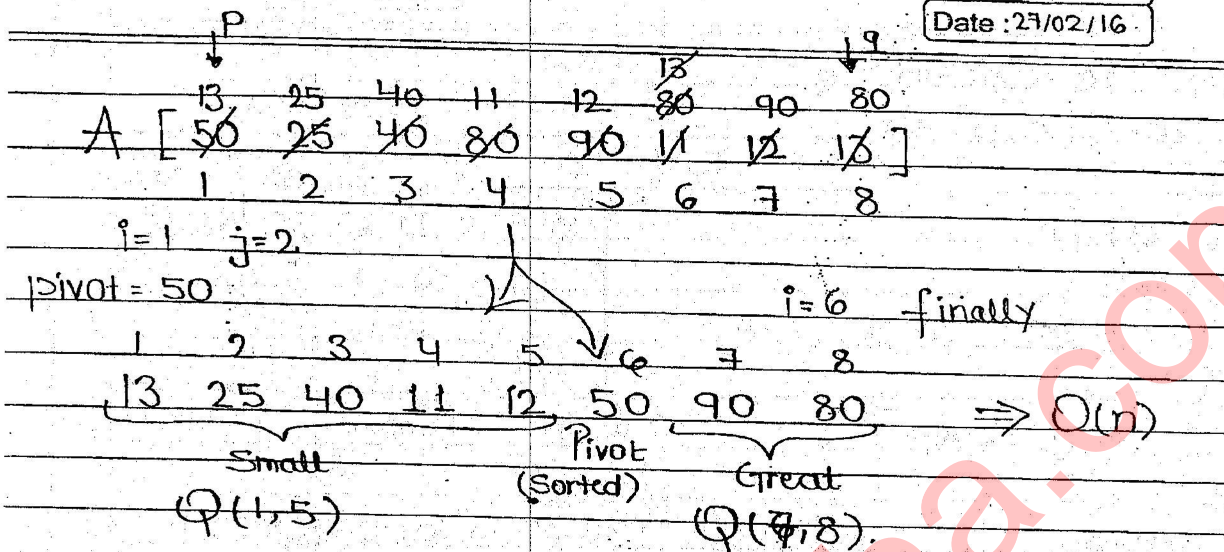
Best case   No. of swaps 1
Worst case   No. of swaps n   [Partition Algorithm

⇒ Time complexity of n elements on partition algorith is O(n) [every case] and it is in-place.

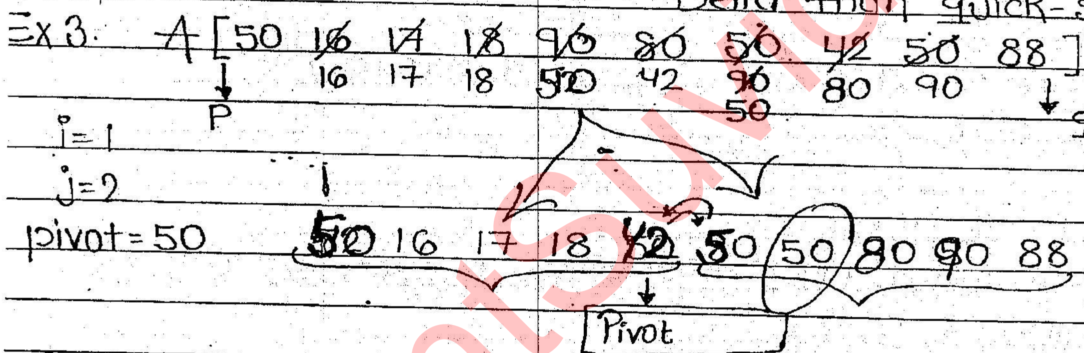
Ex 2. A [~~56~~ 25, ~~25~~ ~~56~~ 40, ~~40~~ ~~50~~ 80, ~~80~~ ~~56~~ 90, ~~96~~ 50, 11, 12, 13]
1 2 3 4 5 6 7 8

x=50 i=1 j=2 p=1 q=8

Download all Notes and papers from StudentSuvidha.com



Randomized Quick-sort → pivot selection random
→ better than quick-sort



```
Quicksort (A, p, q)                          // Algorithm
{
  if ( p==q )                                // small problem.
     return A[p];
  else {
   int m = partition (A, p, q);              // Divide
   Quicksort (A, p, m-1);                    // Conqueor
   Quicksort (A, m+1, q);
     return (A);                             // No combine
  }
}
```

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

StudentSuvidha.com

Download all Notes and papers from StudentSuvidha.com

Let T(n) be time complexity of quick-sort algorith then recurrance relation

Case → $T(n) = \begin{cases} O(1) & \text{if } n=1 \\ 2T(n/2)+n & \text{if } n>1 \end{cases}$ when $m = n/2$.

In general,

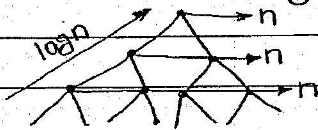
$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ O(n)+T(m-p)+T(q-m) & \text{if } n>1 \end{cases}$

P-----q

Best case

1. Balanced tree,
   exactly two partitions

$T(n) = O(n)+2T(n/2)$
$= 2T(n/2)+n$
$= O(n \log n)$

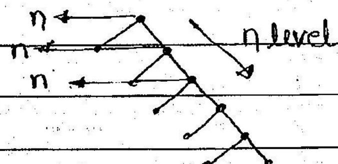


Stack space - $\log_2 n$
Total space - $n + \log_2 n$

Worst case

Unbalanced tree.
One & n-1 partition

$T(n) = O(n)+T(n-1)$
$= T(n-1)+n$
$= O(n^2)$



Stack space – n
Using better program – $\log_2 n$

Average case

$T(n) = 2T(n/2)+n$
$= 2\left[T\left(\frac{n}{2}-1\right)+\frac{n}{2}\right]+n$
$= 2T\left(\frac{n}{2}-1\right)+2n$ [Small constants does not matter]
$= O(n \log n)$ = Best case (same recurrance rel^n)

[Insertion sort will give best case O(n)]

Worst case if array is already sorted or almost sorted.

Download all Notes and papers from StudentSuvidha.com

Ex- Quicksort algorithm is run on following two i/p to keep in ascending order.

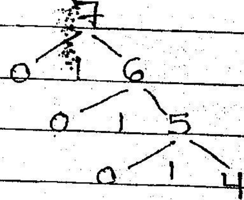
Ist i/p - 1, 2, 3 --- n   Left Gap

2nd i/p - n, n-1, n-2 ... 1   Alternate Gap

Let C_1 and C_2 be the no. of comparisons made for the i/p I and II respectively. Then relation between C_1 & C_2.

Solution - Take any i/p of the form to apply partition

Ist I/P   10 20 30 40 50 60 70



⇓ 7 elements
6 comparisons, 1 swap

10 (20 30 40 50 60 70)

⇓ 6 elements
5 comparisons, 1 swap

⇓

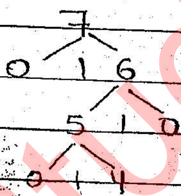
2 elements
1 comparison, 1 swap

In general $(n-1)+(n-2)+(n-3)+ \dots 2+1$

$n(n-1)/2 = O(n^2)$ Comparisons

n-1 Swaps

2nd I/P   ~~70~~ 10 ~~60~~ ~~50~~ ~~40~~ 40 ~~30~~ 30 ~~20~~ 20 ~~10~~ 70



⇓ 7 elements, 6 comparisons, 7 swaps

(10 60 50 40 30 20) 70

⇓ 6 elements, 5 comparisons
16 swaps

10 (60 50 40 30 20) 70

⇓ 4 elements, 3 comparisons
4 swaps

3 elements, 1 swap, 2 comparisons

2 elements, 2 swap, 1 comparison

Comparisons = $O(n^2)$

Swaps = $n+1+n+1+ \dots$

$= \frac{n}{2} \times n + \frac{n}{2} \times 1 = O(n^2)$

So, $C_1 = C_2$ (Answer) $S_1 < S_2$ (swaps)

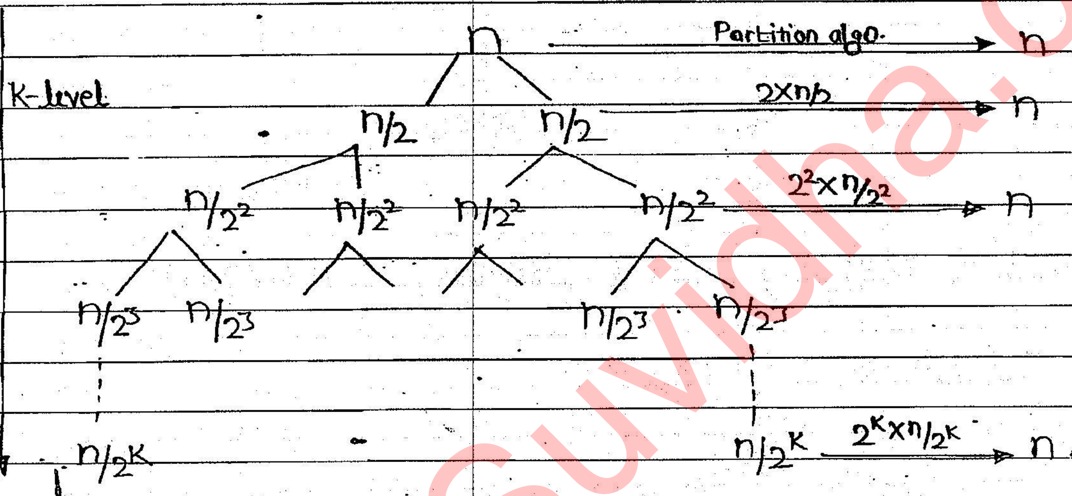
Download all Notes and papers from StudentSuvidha.com

⇒ Quick-sort gives best case & average case performance if the array is not sorted.

Recursive Tree method →

Two recursion calls

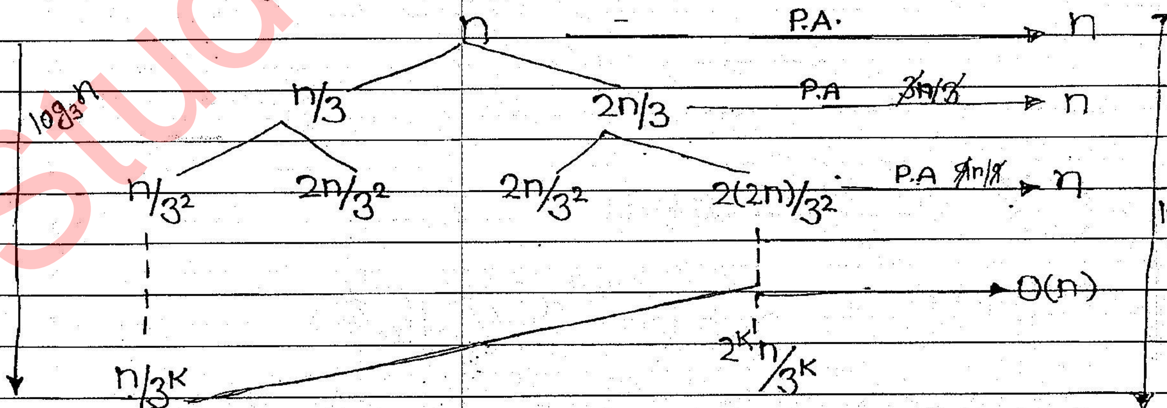
1. $T(n) = T(n/2) + T(n/2) + n$



$\frac{n}{2^K} = 1 \Rightarrow n = 2^K \Rightarrow K = \log_2 n$

$T(n) = nK$
$= n\log_2 n$
$= \Theta(n\log_2 n)$

2. $T(n) = T(n/3) + T(2n/3) + n$



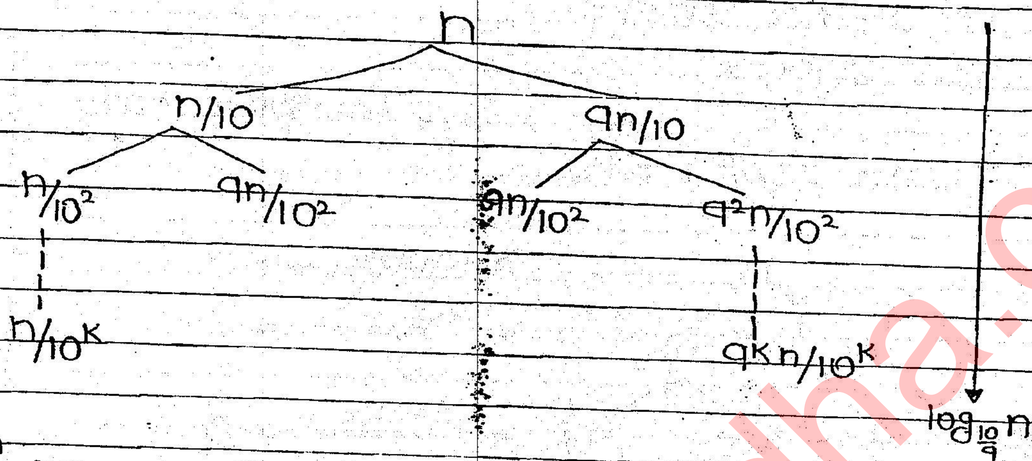
$\frac{n}{(3/2)^K} = 1 \Rightarrow K = \log_{3/2} n$

$T(n) = O(n\log_{3/2} n)$
$T(n) = \Omega(n\log_3 n)$
$= \Theta(n\log n)$

Download all Notes and papers from StudentSuvidha.com

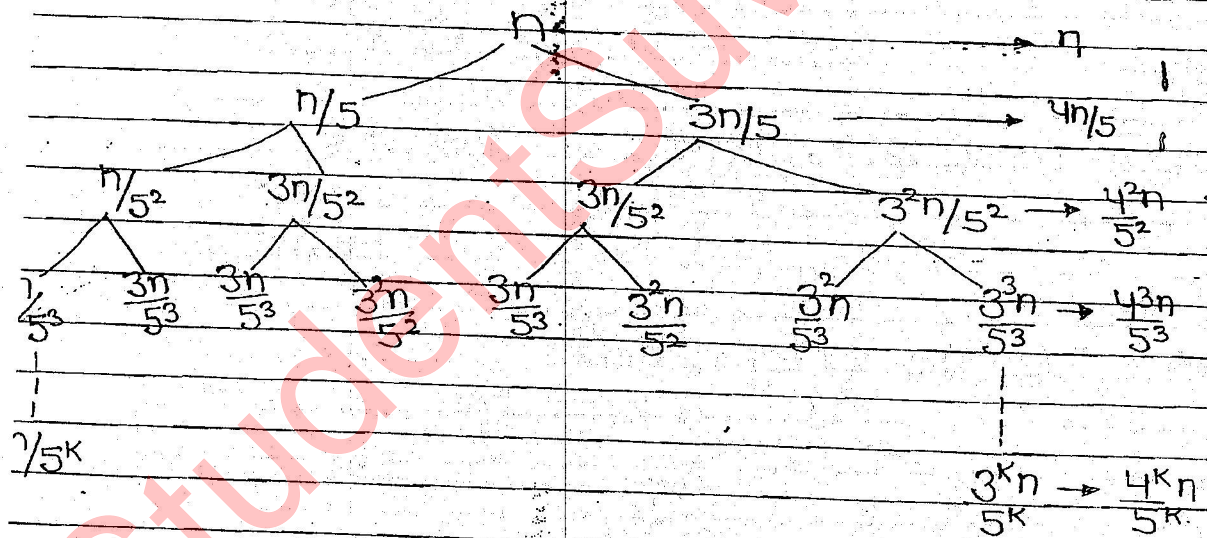
Page No. 10
Date : 27/02/16

$T(n) = T(n/10) + T(9n/10) + n$



$T(n),\ O(n\log_{10/9} n),\ \Omega(n\log_{10} n),\ \Theta(n\log n)$

$T(n) = T(n/5) + T(3n/5) + n$



$n = \left(\frac{5}{3}\right)^K \Rightarrow K = \log_{5/3} n$

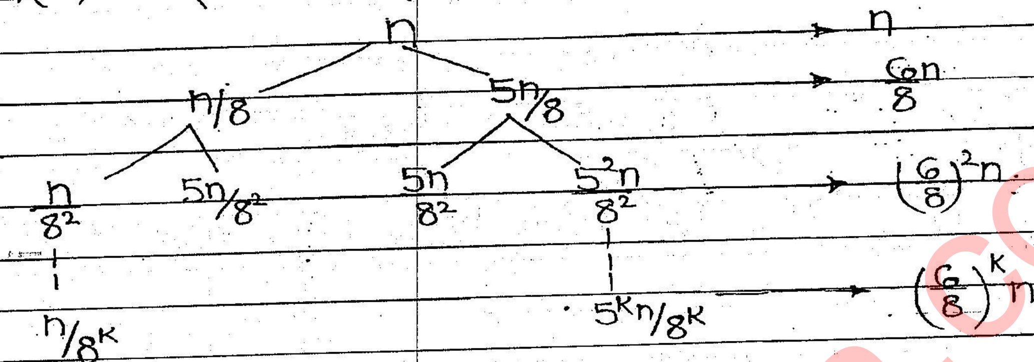
$T(n) = \left[n + \frac{4n}{5} + \frac{4^2 n}{5^2} + \cdots \left(\frac{4}{5}\right)^{\log_{5/3} n} n\right]$

$= n \left[\frac{1-(4/5)^{\log_{5/3} n}}{1/5}\right] = O(n) = O(n)$

$\Omega(n) = \Theta(n)$

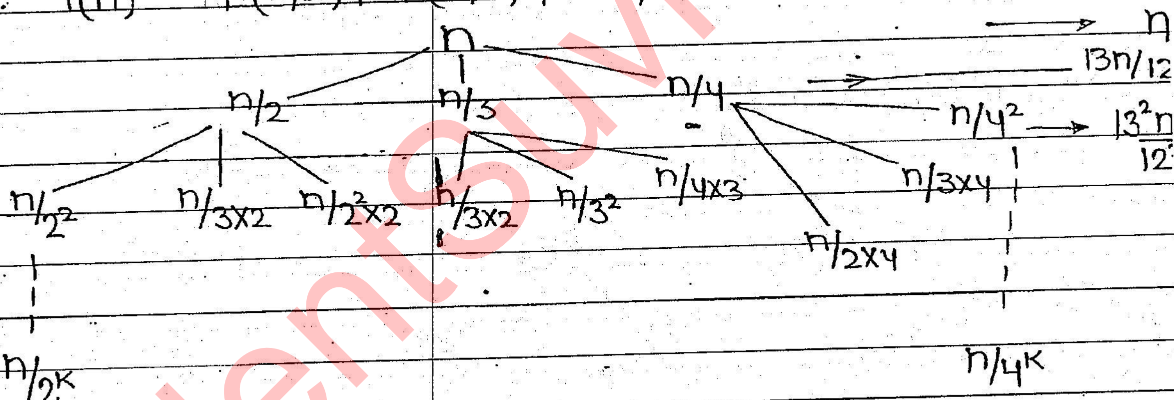
Download all Notes and papers from StudentSuvidha.com

5. $T(n) = T(n/8) + T(5n/8) + n$



$$T(n) = n\left[1 + \frac{2}{4} + \left(\frac{3}{4}\right)^2 + \cdots \left(\frac{3}{4}\right)^{\log_{8/5} n}\right] = O(n)$$
$$= \Omega(n)$$
$$= \Theta(n)$$

6. $T(n) = T(n/2) + T(n/3) + T(n/4) + n$



$$T(n) = n\left[1 + \frac{13}{12} + \frac{13^2}{12^2} + \cdots \left(\frac{13}{12}\right)^{\log_4 n}\right]$$

$$= n\left[\frac{1\left((13/12)^{\log_4 n} - 1\right)}{1/13}\right] = 13n\left[\left(\frac{13}{12}\right)^{\log_4 n} - 1\right]$$

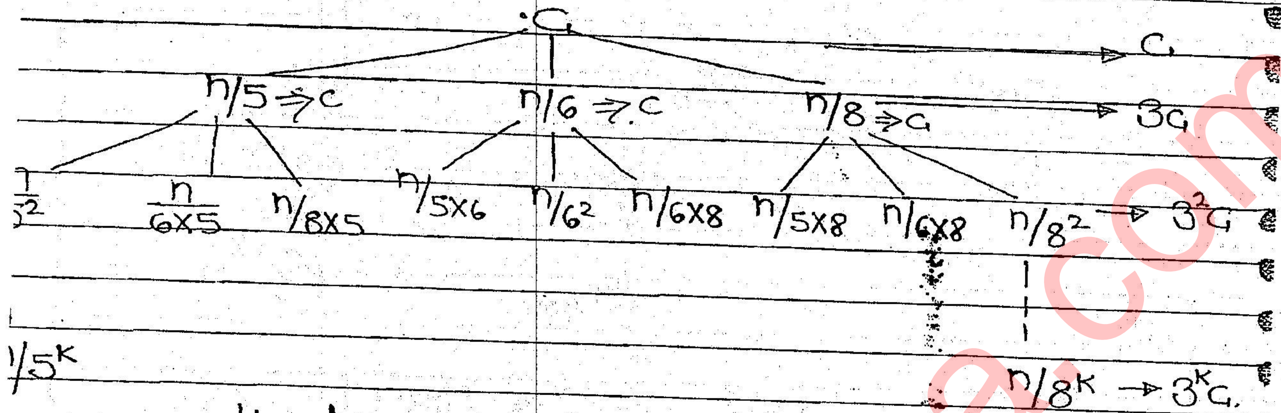
$$= 13n\left[n^{\log_4 13 - \log_4 12} - 1\right] = 13n\left[\;\;\right].$$

$$= \Omega(\;\;)$$

for upper bound, take $\log_2 n$.

Download all Notes and papers from StudentSuvidha.com

1. $T(n) = T(n/5) + T(n/6) + T(n/8) + c$



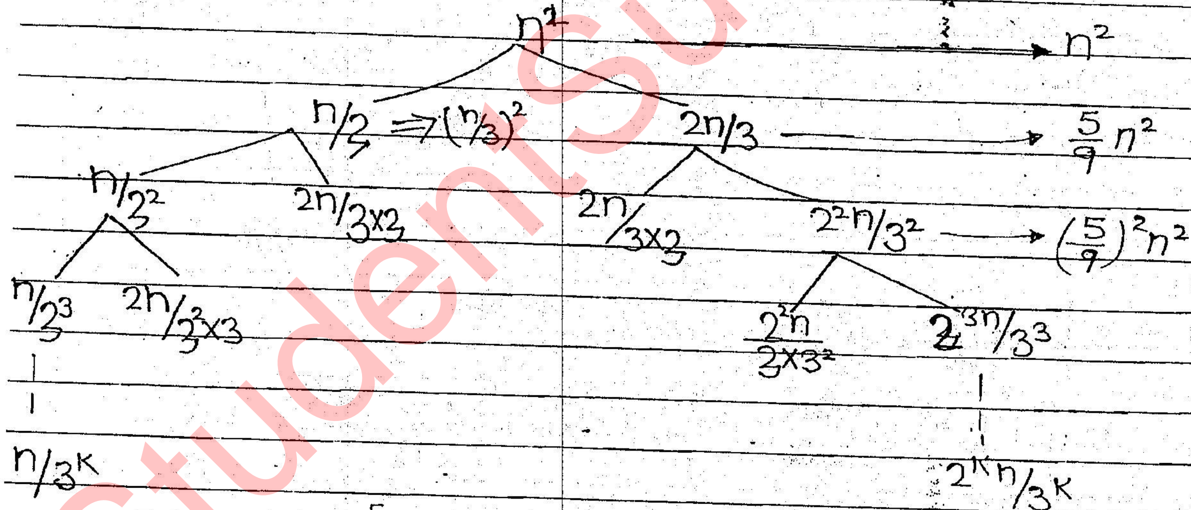
$k = \log_5 n$ upper-bound

$T(n) = (1 + 3 + 3^2 + \dots 3^{\log_5 n}) c$

$= \frac{c}{2}(3^{\log_5 n} - 1) = c(3^{\log_5 n})$

$= O(3^{\log_5 n})$

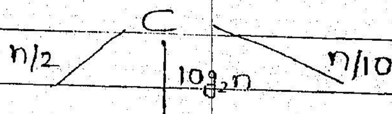
3. $T(n) = T(n/2) + T(2n/3) + n^2$



$T(n) = n^2 \left[1 + \frac{5}{9} + \frac{5^2}{9^2} + \dots \left(\frac{5}{9}\right)^{\log_3 n}\right]$

$= O(n^2)$

9. $T(n) = T(n-2) + T(n/2) + T(n-10) + c$



Download all Notes and papers from StudentSuvidha.com

Page No. 13
Date : 27/02/16

Best & Average case -

$T(n) = n + T(n/2) + T(n/2) \Rightarrow O(\log_2 n \cdot n)$

$T(n) = n + T(n/5) + T(4n/5)$

$T(n) = n + T(n/10) + T(9n/10)$

[base depend on α]

In general,

$T(n) = n + T(\alpha n) + T((1-\alpha)n) \quad 0 < \alpha < 1$

fraction of n one side, remaining another side

base $= \max\left[\log_{\frac{1}{\alpha}} n \text{ or } \log_{\frac{1}{1-\alpha}} n\right]$

Stack size $= \max[\ \ ]$

Worst case -

$T(n) = n + T(n-1) + T(0) = O(n^2)$

$T(n) = n + T(1) + T(n-2) = \frac{n^2}{2} \times n$

$= O(n^2)$

n → n
n-1, 0 → n-1
n-2, 0 → n-2

$T(n) = n + T(9) + T(n-10)$

$= \frac{n}{10} \times O(n) = O(n^2) \quad = O(n^2)$

In general

$T(n) = n + T(c-1) + T(n-c)$

$= \frac{n}{c} \times O(n) = O(n^2)$

Constant remains one side & remaining another side

Stack size $= O(n/c) = O(n)$

better programming $= O(\log n)$

StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

. In quick sort, the sorting of n numbers, ~~a pivot~~ the $n/5$ th element is selected as pivot using $O(n)$ time (smallest) complexity algorithm. Then what will be time complexity of quick sort in worst case.
(a) $O(n\log n)$ (b) $O(n^2)$ (c) $O(n)$ (d) $O(n^3)$

Solution: $T(n) = O(1) + O(n) + T(n-1) + T(0)$

(Pivot selection) (Partition algorithm) (Worst case)

$$T(n) = O(n) + O(n) + T\left(\frac{n}{5}-1\right) + T\left(n-\frac{n}{5}\right)$$

(given) Left Right

$$= 2n + T(n/5) + T(4n/5) \quad \text{(given by problem)}$$

$$= 2n\log_{5/4} n = O(n\log n)$$

. In quicksort, the sorting of n numbers, the $(n/5)^{th}$ element is selected as pivot using $O(\log n)$ time complexity. Then what will be best case time complexity of quick sort.
i $\Theta(n\log n)$ ii $O(n^2)$ iii $O(n)$ iv $O(n^3)$

Pivot selection (not constant)

1 2 3 4 5 n/5 n

If Ist minimum is selected as pivot, it will go to the first place (its correct position)
But $(n/5)^{th}$ element can go any-where.

$$T(n) = O(\log n) + O(n) + T(n/5) + T(4n/5)$$

(Pivot) (Partition) (Best case (given) (any))

$$= n + T(n/5) + T(4n/5)$$

$$= O(n\log_{5/4} n)$$

Download all Notes and papers from StudentSuvidha.com

Page No. 2
Date : 28/02/16

3. In quick-sort, the sorting of n numbers, the $\left(\frac{n}{10}\right)^{th}$ largest element is taken as pivot using $O(n^2)$. What will be the best case time complexity of quick-sort.

$$T(n) = O(n^2) + O(n) + T\left(n - \frac{n}{10}\right) + T\left(\frac{n}{10} - 1\right)$$

(neglect; left)

$$= O(n^2) + O(n) + T\left(\frac{n}{10} - 1\right) + T\left(n - \frac{n}{10}\right)$$

(Right)

$$= n^2 + T(n/10) + T(9n/10)$$

$$= O(n^2) \quad \text{[By recursive tree method]}$$

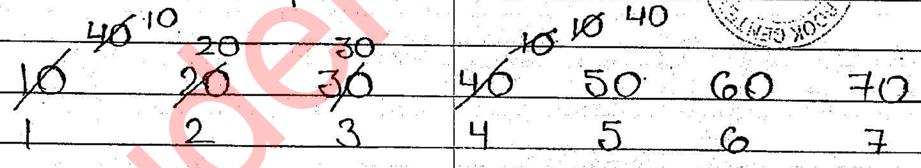
The problem is one-way, no other cases.

Median is nothing but middle person in sorted array
↳ $\left(\frac{n}{2}\right)^{th}$ smallest element

Middle element is $(n/2)^{th}$ element.

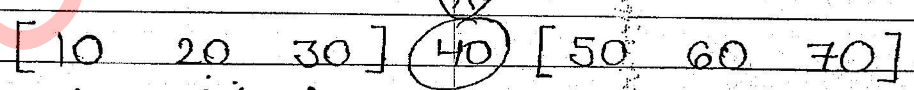
[Best / Worst case depends on the selection of the pivot]. (middle element as pivot in sorted array)

Randomized Quick-Sort -



r = RG (1,7)
Swap (a[p], a[r])
Apply partition

[10 20 30] (40) [50 60 70]



r = RG(1,3)   r = RG (5,7)
Swap (a[p], a[r])   Swap (a[p], a[r])
Apply partition   Apply partition.

Download all Notes and papers from StudentSuvidha.com

Page No. 3
Date :28/02/16

Even if the array is ~~not~~ sorted, randomized quick sort will not give the worst case.
Selecting pivot element randomly is called randomized quick sort.
Always gives the best / average performance.
Worst case when there is a gap on either side always (chances are very less) $O(n^2)$

VI SELECTION PROCEDURE -

i/p: An array of n elements and integer K
O/p: find K^{th} smallest element.

Case I - Apply merge sort & access a[K]
Case II Apply K passes of selection sort.
Case III If array sorted, return a[K].
Case IV Divide & Conqueor

Example

A [50 25 40 70 15 88 92 64 35 80 20 17] K=3
1 2 3 4 5 6 7 8 9 10 11 12

⇓ Partition

7
m (25 40 15 35 20 17) (50) (70 88 92 64 80) K=3

⇓ Partition

4
m (15, 20 17) (25) (35 40)

⇓ Partition

(15) (20) (17) m ⇒ m=3 (17) 20 return a[K]

StudentSuvidha.com
Download all Notes and papers from StudentSuvidha.com

```
Selection (a,K,i,j)
{
 if (i==j)
   return a[i]                                  // small
 else {
 m = partition (a,i,j)  ~~~~> n ( for partition)
 if (m==K)
   then return a[m]      ]~O(1)       // Best case
 else if (m<K)
 Selection (a,K,m+1,j)
 else
 Selection (a,K,i,m-1)
 }
  }
```

Recurrance relation - Let T(n) be the time complexity of above algorithm on n elements

then $T(n) = \begin{cases} O(1) & \text{if } n=1 \\ T(m-i) \text{ or } T(j-m) + O(1) + O(n) & \text{if } n>1 \end{cases}$

(Average to Worst case)

Best case / Average case

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

⇓

$O(n)$

Worst case

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

⇓

$O(n^2)$

1. Better partitions
2. Immediately matching fraction of n

Stack space $O(n)$

using better program $O(\log n)$

Download all Notes and papers from StudentSuvidha.com