Master theorem

If given recurrence expression in the form of

$$T(n) = aT(n/b) + f(n)$$

where always
$a \geq 1$
$b > 1$

& $f(n) = O(n^k \log^p n)$

then

Case I If $\log_b a > k$ then $T(n) = O(n^{\log_b a})$

Case II

If $\log_b a = k$

then

i) If $p > -1$ then $T(n) = O(n^k \log^{p+1} n)$

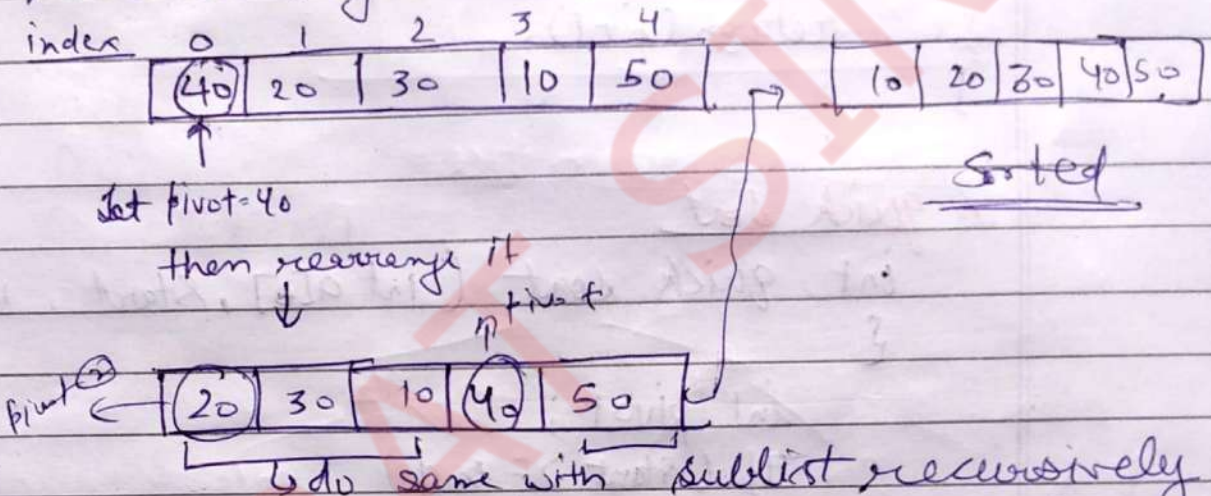
ii) If $p = -1$ then $T(n) = O(n^k \log \log n)$

iii) If $p < -1$ then $T(n) = O(n^k)$

Case III If $\log_a b < k$ If $p \geq 0$ then $T(n) = O(n^k \log^p n)$
$p < 0$ $T(n) = O(n^k)$

Ques 2 → **Quick Sort** :→ It is also known as partition-exchange sort. it is based on Divide & Conquer technique. In it we pick an element in list, called pivot. then all the elements ~~left~~ smaller than pivot is placed to it's left while all larger elements are placed to right.

Eg.



and terminating condition for recursion is when sublist formed contains only one element or no element..

→ **algorithm** :→

```
%% partion of array
int temp i, j, p;
p = a[end]
i = start - 1;
for (j=start ; j<=end-1 ; j++)
{
    if [a[j]<p)
    {
        i++
        temp = a[i]
```

```
            a[i] = a[j];
            a[j] = temp;
        }
    }
    temp = a[i+1];
    a[i+1] = a[end];
    a[end] = temp;
    return (i+1);
}
```

quick sort

```
int quick_sort (int a[], start, int end)
{
    int pivot;
    if (start >= end)
    {
        return;
    }
    pivot = division_of_array (a, start, end);   (partition)
    quicksort (a, start, pivot-1);
    quick_sort (a, pivot+1, end);
}
```

Time Complexity Using master theorm:-

Master theorm:-

- Utility method for analying recurrence relation
- Useful in many cases for divide & conquer algorithm

- These recurrence relation are of the form

$$T(n) = aT(a/b) + f(n) \quad \text{with } a >= 1 \text{ and } b > 1$$

where

- n = the size of current problem
- a = the number of subproblems in the recursion
- n/b = the size of each subproblem
- $f(n)$ the cost of work has to be done outside the recursive calls.

Eg for best case in Quick sort → The best case scenario for quick sort is when during every pivot step the median of the list is best chosen as pivot.

So

recurrence relation = $T(N) = 2(T(\frac{N-1}{2}) + N-1)$

$\approx T(N) = 2T(N/2) + N-1$

So using master theorem (which is submitted as a seprate pdf)

$\Rightarrow a = 2$

$\Rightarrow b = 2$

$\Rightarrow n^k \log^p n = n-1$

$\Rightarrow n^k \log^p n \approx n$

So $p = 0$

by this we get $k = 1$, $p > -1$

and $\log_2 2 = 1$

$\log_b a = k$

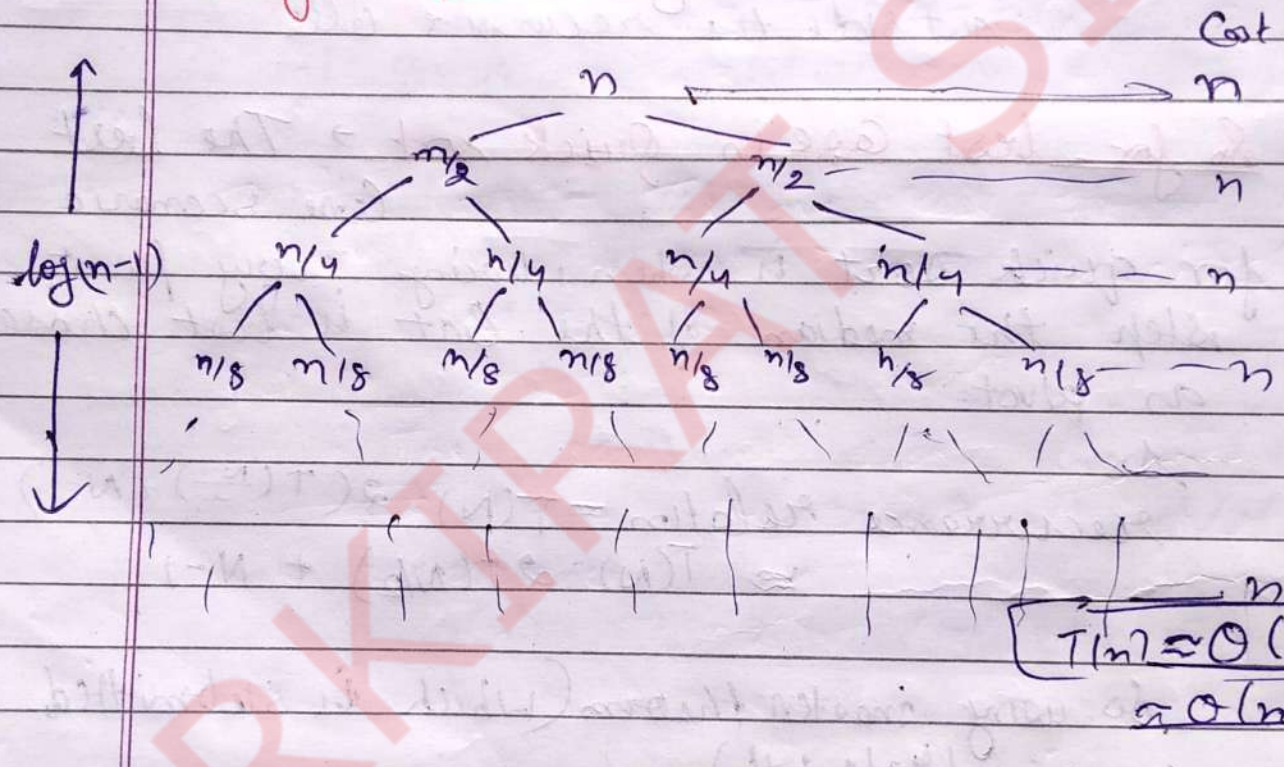
So

Sol^n $\Theta(n^k \log^{p+1} n)$

$\rightarrow \Theta(n^1 \log^{0+1} n)$

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

Using recursion tree :-



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

$= O(n \lg(n))$

for Worst Case → Quick sort is worst case when either the largest ← smallest element is chosen as pivot.

So

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

so $a = 1$, $b = 1$

Now

Let, $m = 2^n$

taking log both side

$\log m = n \log 2$

$\boxed{\log m = n}$

Let

$S(m) = T(\log m)$

So

$S(m) = S(m/2) + \log(m)$

Now using Master theorm

$a = 1, \quad b = 2, \quad k = 0$

$\log_b a = k$

$\log_2 1 = k$

$0 = 0$

$P = 1$

$P > -1$

So

Using Case II + Method I. of Master theorm

Sol^m

$O(m^k \log^{p+1} m)$

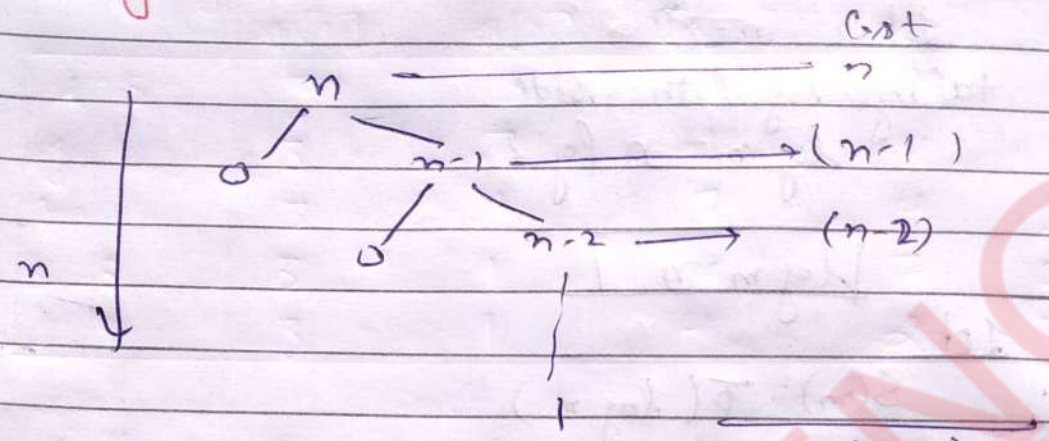
$= O(m^0 \log^{1+1} m)$

$= O\{(\log m)^2\}$ put $m = 2^n$

$= O\{(\log 2^n)^2\}$

$\Rightarrow \underline{O(n^2)}$

b) Using recursion tree :-



$T_n \simeq O(n^2)$

for Avg. Case :- The avg.-case ~~running~~ time of quick sort is much closer to best case

~~for eg.~~ Let

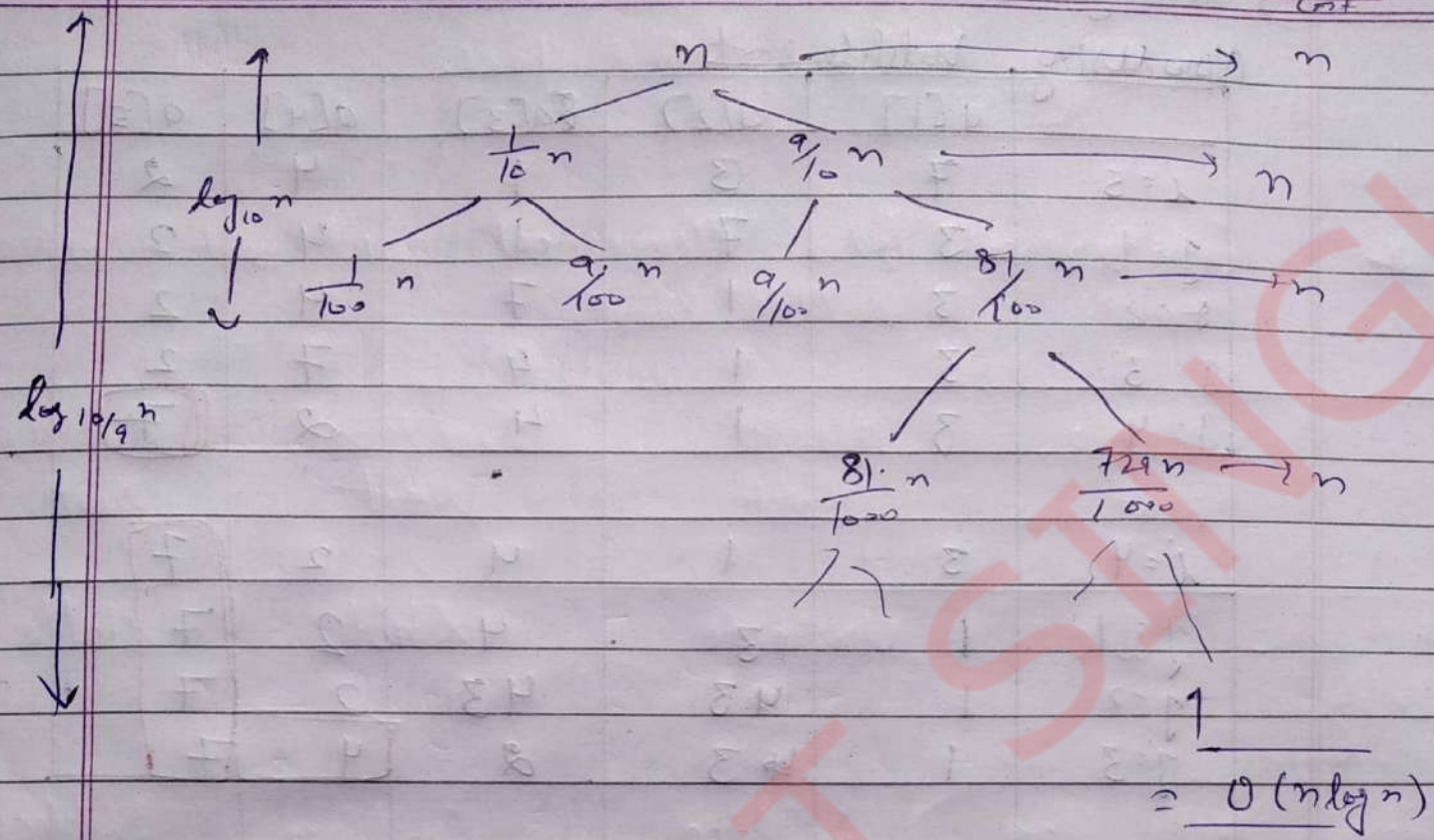
The partitioning algo. always produce ~~8 to 2~~ 9 to 1 proportional split. So recurrence.

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

Same as best case using master theorm

$T(n) = O(n \log n)$

Using recursion tree :-



Bubble Sort: {array (a[])}

1) Begin
2) for i=1 to n-1 (taking index 1)
3) for j=1 to n-i
4) if (a[j] > a[j+1]) then
5) Swap (a[j], a[j+1])
6) end

Example

Let ~~a list of no.~~ =
A Array = [7, 3, 1, 4, 2]

→ taking index 1

Now using bubble sort

	a[1]	a[2]	a[3]	a[4]	a[5]
i=5	7	3	1	4	2
j=1	3	7	1	4	2
j=2	3	1	7	4	2
j=3	3	1	4	7	2
j=4	3	1	4	2	7
i=4	3	1	4	2	7
j=1	1	3	4	2	7
j=2	1	3	4	2	7
j=3	1	3	2	4	7
i=3	1	3	2	4	7
j=1	1	3	2	4	7
j=2	1	2	3	4	7
i=2	1	2	3	4	7
j=1	1	2	3	4	7
i=1	1	2	3	4	7

→ Sorted

Note → This algo. can be Optmized as we can see be can break loop in b/w the process bcz array is sorted when i=3 (See my optimized ~~Array~~ Algo. on github)

Now time Complexity of Bubble sort

So
taking previous example.. we can see there is total
$n-1$ iterations for outer loop.
So
for $n-1$ iterations.

No. of Comparisons
$= (n-1) + (n-2) + (n-3) + \cdots + 3 + 2 + 1$
$= n(n-1)/2 \simeq O(n^2)$

Swaps ⇒ So for every comparison ~~(in worst case)~~ we have to do a swap
Swap = Comparison
$= \frac{n(n-1)}{2} = O(n^2)$

So
$T(n) = O(n^2) + O(n^2)$
$= 2O(n^2)$
$\simeq O(n^2)$

Space Complexity

Bubble Sort → As we see there is not any extra space is required in bubble sort so it's space complexity = Constant
$= O(1)$, so it's an inplace algo

quickSort :- quick sort has space complexity of $O(\log n)$ but it is a inplace algo. bcz it does not consume heap space

it only consume stack space.

Comparison b/w Merge sort, quick sort, Insertion sort & Bubble sort.

	Merge sort	quick sort	Insertion sort	Bubble sort
Sort Stability	Stable	Unstable	Stable	Stable
Category	out of place	Inplace	Inplace	Inplace
Time Complexity (Worst Case)	$O(n \log n)$	~~$O(n^2)$~~ $O(n^2)$	$O(n^2)$	$O(n^2)$
space Complexity	$O(n)$	$O(\log n)$	$O(1)$	$O(1)$