QUICKSORT

→ Quicksort is based on principle of divide-and-conquer.

→ Quicksort works by partitioning an array $A[p..r]$ into two non empty sub arrays $A[p..q]$ and $A[q+1..r]$ such that every key in $A[p..q]$ is less than or equal to every key in $A[q+1..r]$.

→ Then, again two subarrays are sorted by recursive calls to quick sort.

→ The three step divide-and-conquer process for quicksort is:

Divide: partition the array $A[p..r]$ into two subarrays $A[p..q-1]$ and $A[q+1..r]$ such that each element of $A[p..q-1]$ is less than or equal to $A[q]$ which is less than or equal to each element of $A[q+1..r]$

Conquer: Sort the two subarrays $A[p..q-1]$ and $[q+1..r]$ by recursive call to quicksort.

Combine: Since the subarrays are sorted in place, no work is needed to combine them: the entire array $A[p..r]$ is now sorted.

```
QUICKSORT(A, p, r)
if p < r
   then q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q-1)
        QUICKSORT(A, q+1, r)
```

```
PARTITION(A, p, r)
x ← A[r]
i ← p-1
for j ← p to r-1
    do if A[j] ≤ x
        then i ← i+1
             exchange A[i] ↔ A[j]
exchange A[i+1] ↔ A[r]
return i+1
```

75

→ The running time of quicksort depends on whether the partitioning is balanced or unbalanced. It depends on which elements are used for partitioning.

→ If partitioning is balanced, the algorithm runs asymptotically as fast as merge sort. If the partitioning is unbalanced, it can run asymptotically as slowly as insertion sort.

Scanned by CamScanner

$x = A[r] = 4$

Step-1: p ⇒

i	j							r
	2	8	7	1	3	5	6	4

Step-2:

i	j						r
2	8	7	1	3	5	6	4

Step 3:

i		j					r
2	8	7	1	3	5	6	4

Step 4:

i			j				r
2	8	7	1	3	5	6	4

Step 5:

	i			j			r
2	1	7	8	3	5	6	4

Step 6:

		i			j		r
2	1	3	8	7	5	6	4

Step 7:

		i				j	r
2	1	3	8	7	5	6	4

Step 8:

		i					r
2	1	3	8	7	5	6	4

Step 9:

		i					r
2	1	3	4	7	5	6	8

Best Case time complexity of Quick sort:

If the two array contains n/2 elements each then at that time we will go for best case analysis:

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

$$n^{\log_b a} = n \quad f(n) = \Theta(n^{\log_b a})$$
$$= \Theta(n^{\log_2 2})$$
$$= \Theta(n)$$

$$\text{So, } T(n) = \Theta(\cancel{n^{\log_b a} \log n}) = \Theta(n \log n)$$
$$\Theta(n^{\log_b a} \log n)$$

Worst case time complexity of quick sort

→ This situation will occur when the array is divided into two parts such as one part contains zero element and 2nd part contains (n-1) elements.

→ Let us assume that the unbalanced partitioning arises on each recursive call. The partitioning costs Θ(n) times.

The recursive call on an array of size 0 gives
$T(0) = \Theta(1)$

The recurrence for running time is
$T(n) = T(n-1) + T(0) + \Theta(n)$
$T(n) = T(n-1) + \Theta(n)$ —— ①
$T(n-1) = T(n-2) + \Theta(n)$ —— ②
$T(n-2) = T(n-3) + \Theta(n)$ —— ③
$T(n-3) = T(n-4) + \Theta(n)$ —— ④

~~put eqn ② in eqn ①, we get~~
~~$T(n) = T(n-2)$~~

$T(n) = T(n-1) + \Theta(n)$
$\Rightarrow T(n) = T(n-1) + n$ —— ①
$\Rightarrow T(n-1) = T(n-2) + (n-1)$ —— ②

put eqn ② in eqn ①, we get
$T(n) = T(n-2) + (n-1) + n$ —— ③
$T(n-2) = T(n-3) + n-2$ —— ④

put eqn ④ in eqn ③, we get
$T(n) = T(n-3) + (n-2) + (n-1) + n$
$T(n) = T(n-4) + (n-3) + (n-2) + (n-1) + n$
— — — — — —
— — — — — —

$T(n) = 1 + 2 + 3 + \cdots + n$
$= \frac{n(n+1)}{2} = \frac{n^2+n}{2} = \Theta(n^2)$

proof for worst case Analysis:

Let $T(n)$ be the worst case running time of the algorithm QUICKSORT on an input of size n.

$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n) \quad \because 0 \le q \le n-1$

By using substitution method, we guess that
$T(n) \le cn^2$

By using induction method
$T(q) \le cq^2$
$T(n-q-1) \le c(n-q-1)^2$

Then, $T(n) \le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n)$
$\le c \cdot \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n)$

$q^2 + (n-q-1)^2 \le (n-1)^2 \quad (\because 0 \le q \le n-1)$
$q^2 + (n-q-1)^2 \le n^2 - 2n + 1$

77

$$T(n) \le cn^2 - c(2n-1) + \Theta(n)$$
$$\le cn^2$$

we can pick the constant c large enough so that the $c(2n-1)$ term dominates the $\Theta(n)$ term.

$$\therefore T(n) = O(n^2)$$

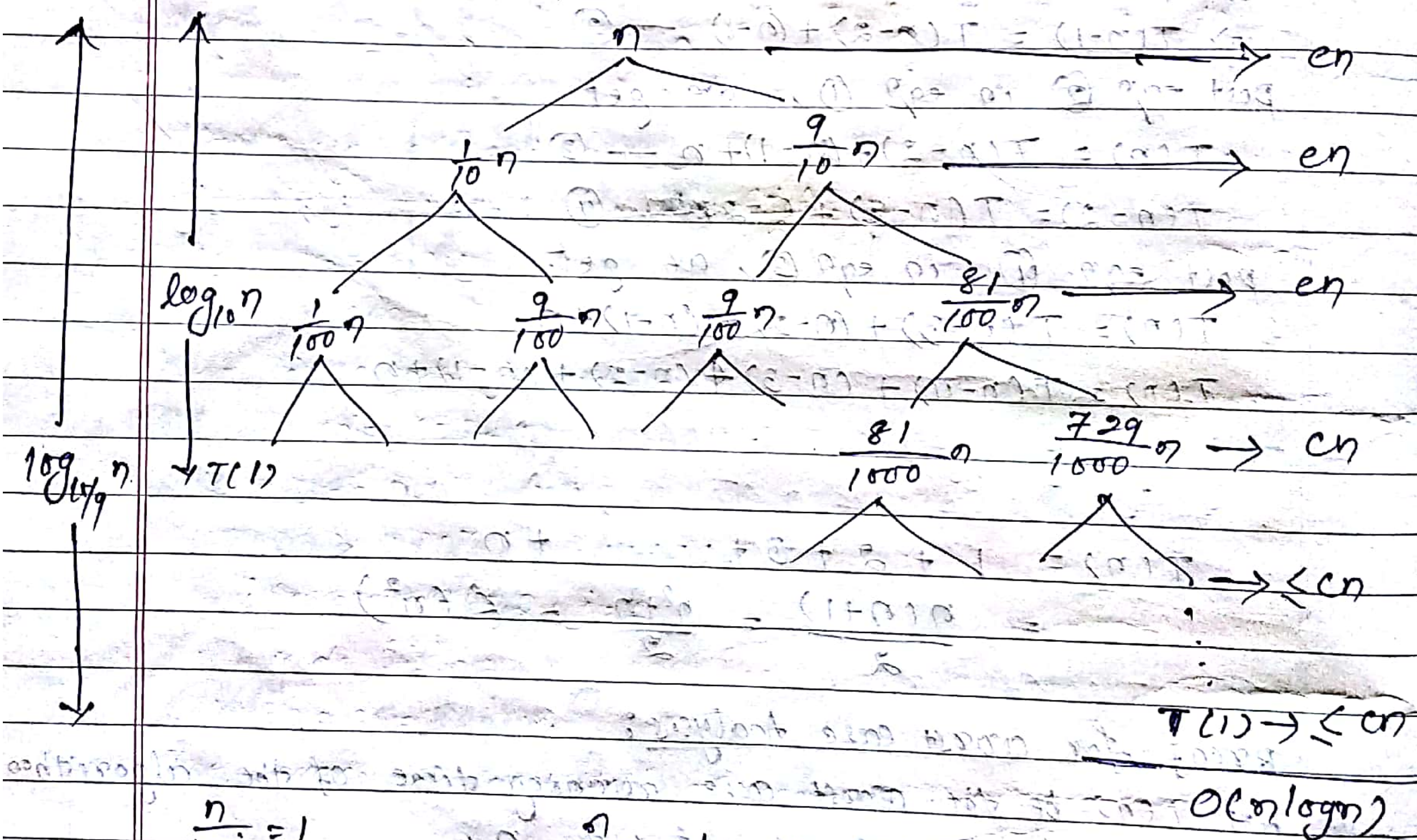
$\Rightarrow$ when the partition is unbalanced, quick sort takes $\Omega(n^2)$ time. i.e. $T(n) = \Omega(n^2)$

Thus, the worst case running time of quicksort is $\Theta(n^2)$

Balanced partitioning:

The partitioning algorithm always produces a 9-to-1 proportional split which is unbalanced.

$$T(n) \le T(9n/10) + T(n/10) + cn$$



$$\frac{n}{10^i} = 1 \qquad\qquad \frac{n}{\left(\frac{10}{9}\right)^i} = 1$$
$$\Rightarrow n = 10^i$$
$$\Rightarrow \boxed{i = \log_{10} n} \qquad\qquad \Rightarrow \boxed{i = \log_{\frac{10}{9}} n}$$

$$T(n) = cn + cn + cn + \dots + \log_{\frac{10}{9}} n \text{ times}$$
$$= cn(1 + 1 + 1 + \dots + \log_{\frac{10}{9}} n)$$
$$= cn \log n$$
$$T(n) = O(n \log n)$$

A RANDOMIZED VERSION OF QUICKSORT :-

→ In average case ~~bad~~ of quicksort, all the input numbers are equally likely. we can sometimes add randomization to an algorithm in order to obtain good average case performance over all inputs.

→ Instead of always using A[r] as the pivot, we will use a randomly chosen element from the subarray A[p..r]

→ we do so by exchanging element A[r] with an element chosen random from A[p..r].

→ The pivot element x = A[r] is equal to be any of the (r-p+1) elements in the subarray.

RANDOMIZED-PARTITION (A, p, r)

1. i ← RANDOM (p, r)
2. exchange A[r] ⟷ A[i]
3. return PARTITION (A, p, r)

RANDOMIZED-QUICKSORT (A, p, r)

1. if p < r
2. then q ← RANDOMIZED-PARTITION (A, p, r)
3. RANDOMIZED-QUICKSORT (A, p, q-1)
4. RANDOMIZED-QUICKSORT (A, q+1, r)

→ By using RANDOMIZED-PARTITION, the running time of quicksort is O(n log n).