Quick Sort

- Quick Sort is often the best Practical choice for Sorting.
- Worst Case running time $O(n^2)$
  Average case " " $O(n \log n)$
- Uses Divide and Conquer Paradigm

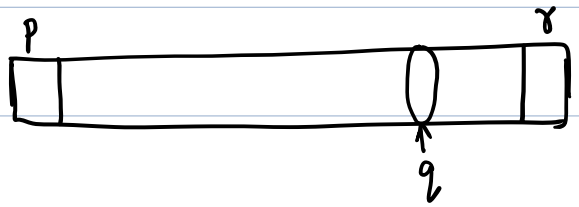
$A[p \ldots r]$

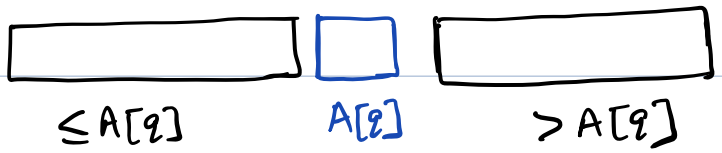
Divide: Partition the array into two (possibly empty) subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ such that

(i) Every element of $A[p \ldots q-1]$ is less than or equal to $A[q]$

(ii) Every element of $A[q+1 \ldots r]$ is greater than $A[q]$

[Computing index q is also a part of this procedure]





* $A[q]$ is called Pivot element
* The divide step puts Pivot in the right position.
  (ie, In the sorted list it appears in the same place)

Conquer: Sort the two subarrays $A[p..q]$ and $A[q+1, ..r]$ recursively.

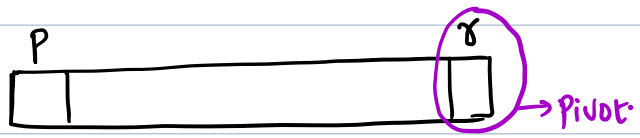
Combine: As the subarrays already sorted, no work is needed for Combine step. The array $A[p..r]$ is sorted.

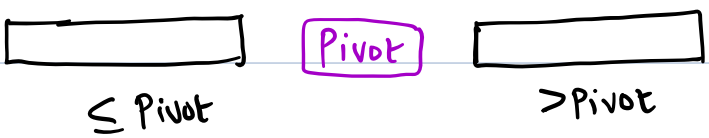
Algorithm:

```
Quick Sort (A, p, r)
    If p<r
        q = PARTITION(A, p, r)
        Quick Sort (A, p, q-1)
        Quick Sort (A, q+1, r)
```

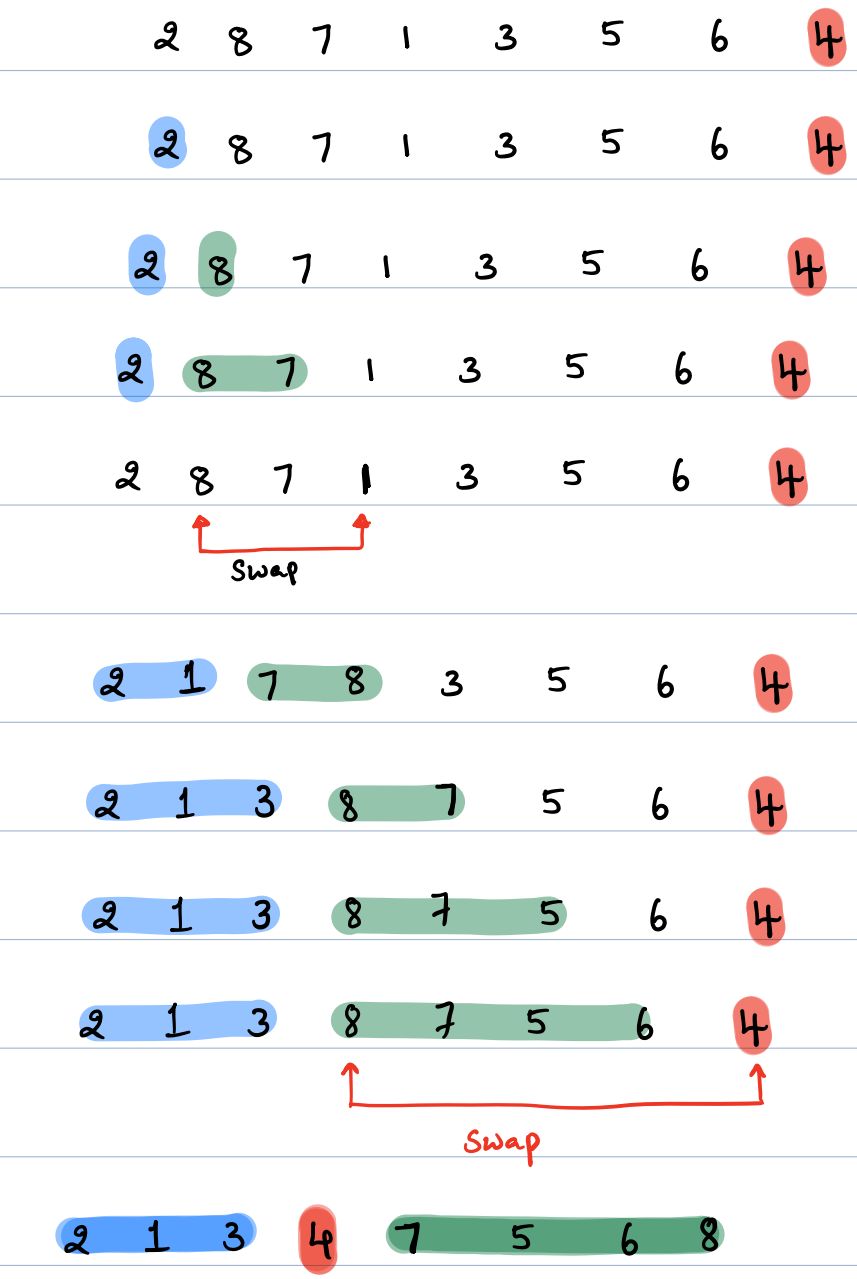
PARTITION Subroutine

Select Pivot element as last element of the array

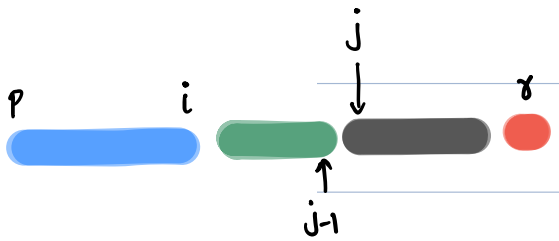




Example:


2 8 7 1 3 5 6 4
2 8 7 1 3 5 6 4
2 8 7 1 3 5 6 4
2 8 7 1 3 5 6 4
2 8 7 1 3 5 6 4
Swap
2 1 7 8 3 5 6 4
2 1 3 8 7 5 6 4
2 1 3 8 7 5 6 4
2 1 3 8 7 5 6 4
Swap
2 1 3 4 7 5 6 8

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



Loop invariant (lines 3-6)

Before the iteration j, for any index k,

(i) If $p \le k \le i$, then $A[k] \le x$

(ii) If $i+1 \le k \le j-1$, then $A[k] > x$

(iii) If $k = r$, then $A[k] = x$.

The indices b/w j & r-1 are not covered as these entries have no relationship to the pivot x

Running time of PARTITION:

$\Theta(n)$, where $n = r - p + 1$

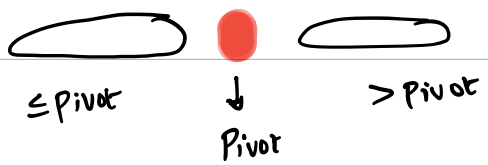
```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1                    → r − p iterations
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Running time of QUICKSORT

```
Quick Sort (A, p, r)
    If p < r
        q = PARTITION(A, p, r)        → Θ(r-p+1)
        Quick Sort (A, p, q-1)   } ?
        Quick Sort (A, q+1, r)   }
```

$$T(n) = T(q-p) + T(r-q) + \Theta(r-p+1)$$

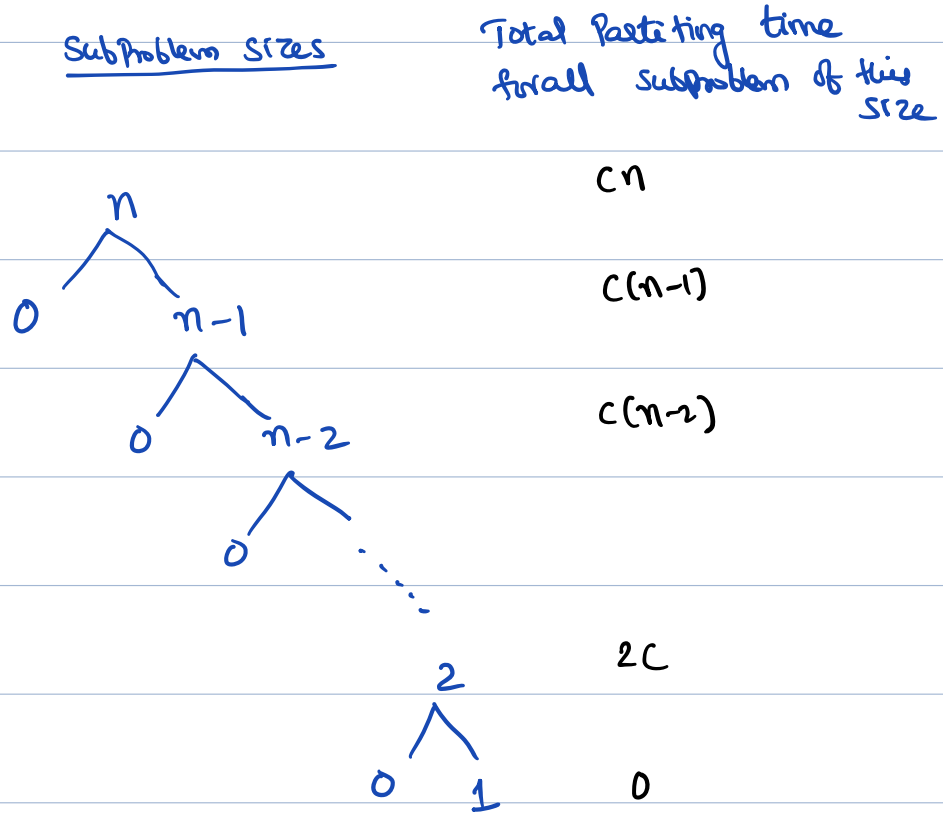
The running time of quicksort depends on the Partitioning.



Worst-case

This happens when the Partitioning Subroutine produces one subproblem with $n-1$ elements and one with zero elements (in the each recursive call).

Then



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

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

$$\leq c(1 + 2 + \cdots + n) \quad \text{for some const } c > 0$$

$$= c \cdot \frac{n(n+1)}{2}$$

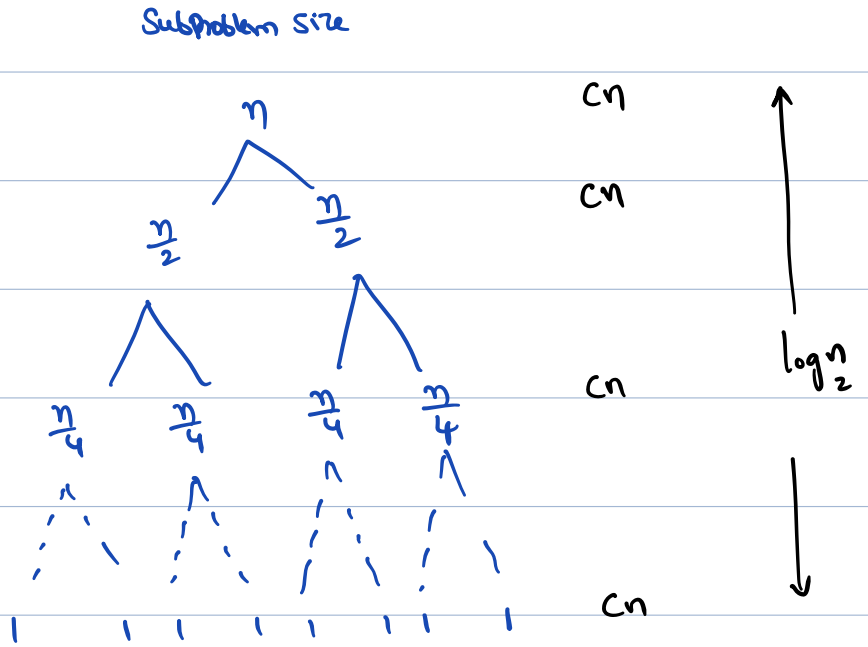
$$T(n) = \Theta(n^2)$$

That is the worst case running time of quick sort is same as insertion sort.

Best Case: Pivot element is median in each recursive call.

$$T(n) = 2\,T(n/2) + \theta(n)$$

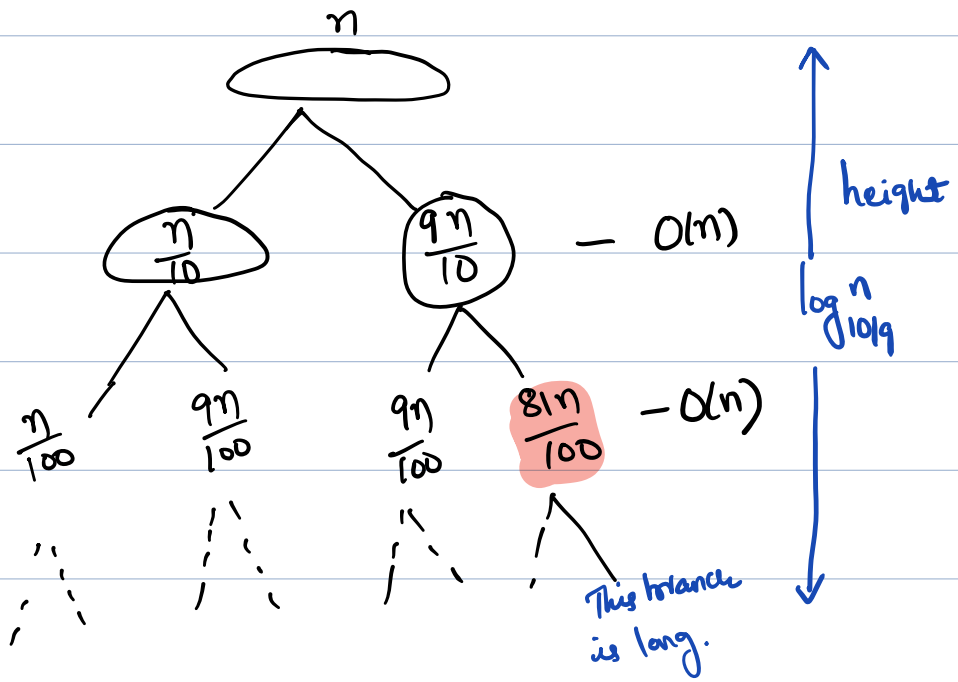
$$T(n) = \theta(n \log n)$$



$$= \theta(n \log n)$$

$\left(\frac{1}{10}, \frac{9}{10}\right)$-split [Partitioning algorithm always produces a 9 to 1 proportional split]

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



Exercise $\log_{10/9} n = O(\log_2 n)$

Hint: $\frac{\log_2 n}{\log_2^{10/9}}$

So running time $= O(n \log n)$

- Even a 99-to-1 split yields an $O(n \log n)$ running time.
- Any split of constant proportion gives a running time $O(n \log n)$.