6. One-to-One : $\log_b a = \log_b c$ iff $a = c$

9. Change of Base : $\log_b a = \frac{\log_c a}{\log_c b} = \frac{\log a}{\log b} = \frac{\ln a}{\ln b}$

Other Logarithmic Definitions :-

Common Logarithms :-

① Logarithms with a based '10' are called "common logarithms". It is customary to write $\log_{10} x$ as "$\log x$".

Natural Logarithms :-

Logarithms with the base of 'e' are called "natural" logarithms. It is customary to write $\log_e x$ as "$\ln x$"

Logarithms

Def:- If $x > 0$ and 'b' is a constant ($b \neq 1$), then $y = \log_b x$ if and only if "$b^y = x$".

In the equation $y = \log_b x$, 'y' is referred to as the logarithm, 'b' is the base, and 'x' is the argument.

→ $y = \log_b x$ — logarithmic form $\quad b^y = x$

$b^y = x$ — exponential form of $y = \log_b x$

Properties of logarithms:-

If b, a and c are positive real numbers, $b \neq 1$ and 'n' is a real number, then:

1. Product : $\log_b (a \cdot c) = \log_b a + \log_b c$

2. Quotient : $\log_b \frac{a}{c} = \log_b a - \log_b c$

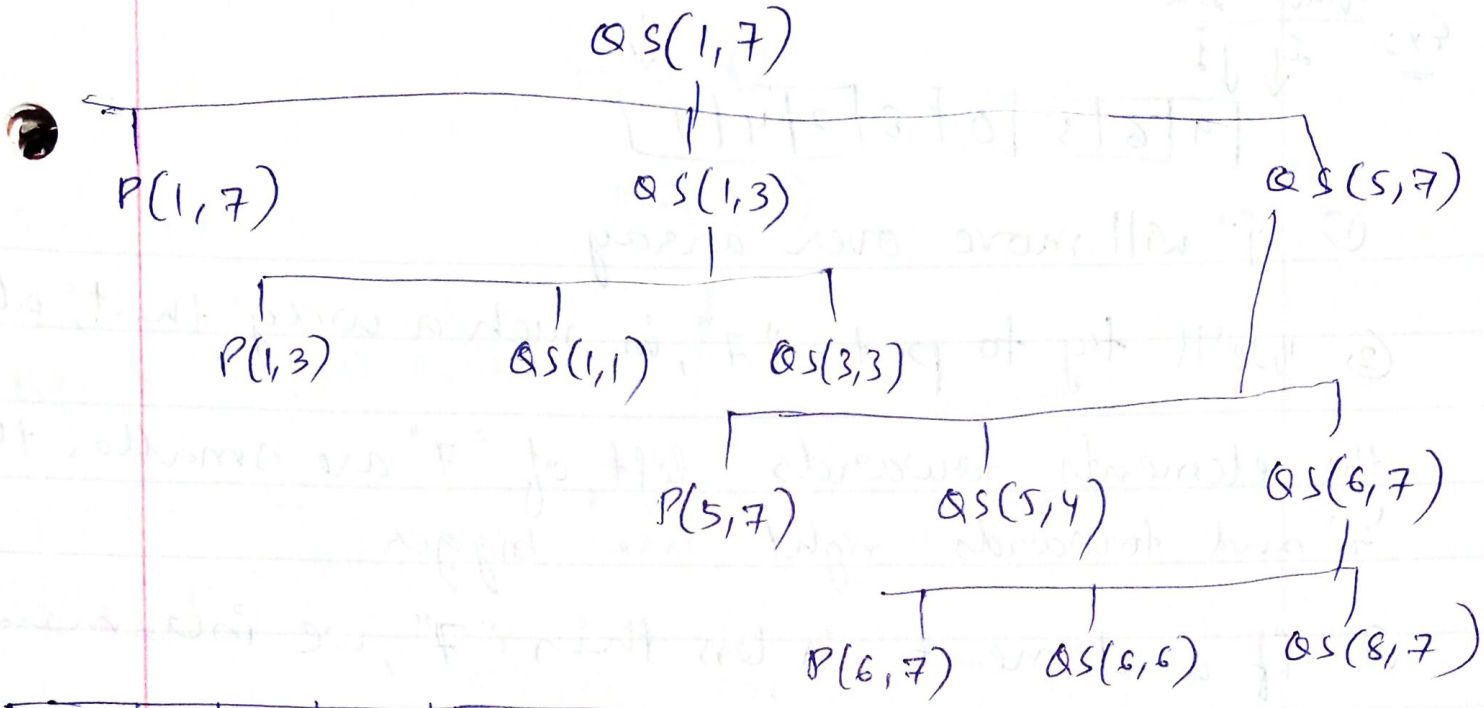
3. Power : $\log_b a^n = n \log_b a$

4. $\log_b 1 = 0$

5. $\log_b b = 1$

6. Inverse 1 : $\log_b b^n = n$

7. Inverse 2 : $b^{\log_b n} = n ; \quad n > 0$



5	7	6	1	3	2	4

Interesting Inputs for Quick-Sort:-

① Ascending order input.

(1 2 3 4 5 6)

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

$= O(n^2)$

② Descending Order

(6 5 4 3 2 1) ⇒ (1 5 4 3 2 6)

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

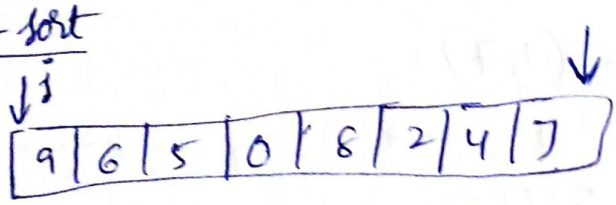
③ Entire array has same elements

(2 2 2 2 2 2) ⇒ $T(n) = O(n^2)$

④

quick-sort

Ex:



① 'j' will move over array

② We'll try to put "7", in such a way that, all the elements towards left of '7' are smaller than '7' and towards right are bigger

③ if a element is less than "7", we increment 'i' and swap, a[i] with a[j]

④ In the algorithm, ~~till~~ from 0^{th} index to i^{th} element elements will be less than '7' & i^{th} to j^{th} element, greater than '7', and after 'j', yet to be examined

→ QUICKSORT(A, p, r)

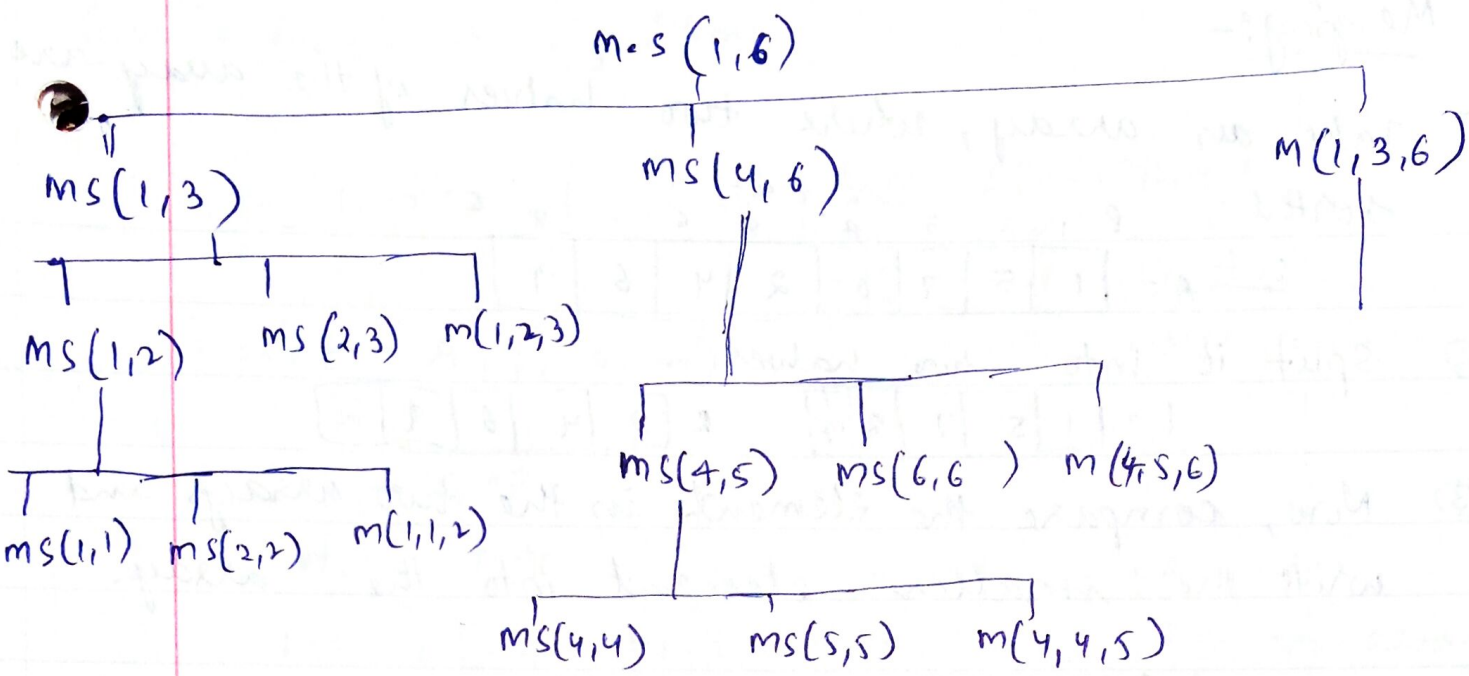
```
{ if (p<r)
  { q = PARTITION(A, p, r)
    QUICKSORT(A, p, q-1)
    QUICKSORT(A, q+1, r)
  }
}
```

	best case, partition at n/2	worst case
	T(n)	T(n)
	O(n)	O(n)
	T(n/2)	T(0)
	T(n/2)	T(n-1)
	$T(n) = 2*T(n/2) + O(n)$	$T(n) = T(n-1) + O(n)$
	$= O(n \log n)$	$= T(n-1) + cn$

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

$= T(n-2) + c(n-1) + cn$

$= T(n-3) + c(n-2) + c(n-1) + cn \quad = O(n^2)$



QUICK - SORT ALGORITHM :-

$\Omega(n \log n)$

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

$O(n^2)$

```
PARTITION (A, p, r)
{
    x = A[r]
    i = p - 1
    for (j = p to r - 1)
    {
        if (A[j] ≤ x)
        {
            i = i + 1
            exchange A[i] with A[j]
        }
    }
    exchange A[i+1] with A[r]
    return i+1
}
```

Merging:-

① Take an array, where two halves of the array are sorted

Ex:- A =

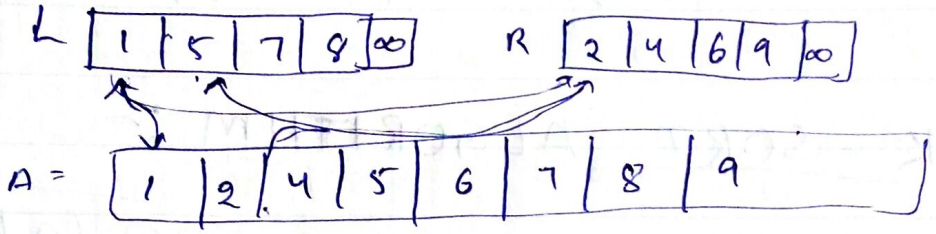
p	1	2	3	q	q+1	6	7	8	r
	1	5	7	8	2	4	6	9	

② Split it into two halves

L: | 1 | 5 | 7 | 8 | ∞ |

R | 2 | 4 | 6 | 9 | ∞ |

③ Now, compare the elements in the two arrays and write the smaller element into the array.



→ Recursive function :-

```
merge_sort (A, p, r)                    — T(n)
{  if (p < r)
       q = ⌊(p+r)/2⌋
       merge_sort (A, p, q)             — T(n/2)
       merge_sort (A, q+1, r)           — T(n/2)
       merge (A, p, q, r)               — O(n)
}
```

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

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

Divide & conquer Algorithms:

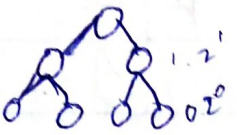
Merge Sort:- (out of place Algorithm)

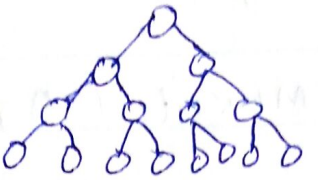
(we don't sort array in the same array)

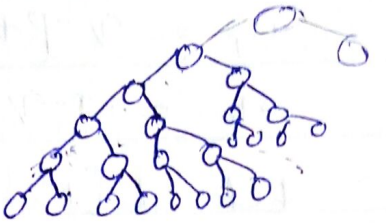
```
MERGE (A, p, q, r)
{
    n1 = q-p+1
    n2 = r-q
    Let L[1 ... n1+1] and R[1 to n2 +1] be new arrays
    for (i=1 to n1)
        L[i] = A[p+i-1]
    for (j=1 to n2)
        R[j] = A[q+j]
    L[n1+1] = ∞
    R[n2+1] = ∞

    i=1, j=1

    for (k=p to r)
        if (L(i) ≤ R[j])
            A[k] = L[i]
            i = i+1
        else A[k] = R[j]
            j=j+1
```

$S(0) = 0$ 0

$S(1) = 1$ 

$S(2) = 1+1 = 2$ 

$S(3) = (1+2+1) + (1+2+1) = 8$ 

$S(4) = ((1+2+1) + 3 + (1+2+1)) + ((1+2+1) + 3 + (1+2+1))$ 

$= 22$

$S(5) = ((1+2+1) + (3) + (1+2+1) + 4 + (3) + (1+2+1) + (1+2+1)) + ((1+2+1) + (3) + (1+2+1) + 4 + (3) + (1+2+1) + (1+2+1))$

$= 52$

$S(6) = 114$