Algorithm :

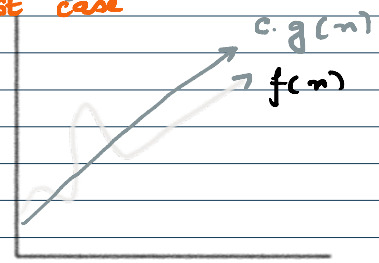
finite set of steps to solve a problem is called Algo

⇒ Analysis done wrt time, space etc.

Asymptotic Notations :

i) Big - oh (O)

Worst case



Least upper bound

$f(n) \leq c \cdot g(n)$

eg: $2n^2 + n$

↓

$O(n^2)$

as $2n^2 + n \leq 3n^2$

ii) Big - Omega (Ω)

Best Case

(greatest lower bound)

eg: $n^2 < 2n^2 + n$

$\Omega(n^2)$

iii) Theta (θ)

Avg case

$c_1 \cdot g(n) \leq f(n) \leq c_2 g(n)$

eg: $2n^2 \leq 2n^2 + n \leq 3n^2$

$\theta(n^2)$

Properties of Asymptotic notations

Big	O	$f(n) \leq c \cdot g(n)$
Big	Ω	$f(n) \geq c \cdot g(n)$
(Theta)	θ	$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$
Small	o	$f(n) < c \cdot g(n)$
Small	ω	$f(n) > c \cdot g(n)$

eg: $2n^2 + n$

$o(n^2)$

$\Omega(n^2)$

$\theta(n^2)$

$o(n^3)$

$\omega(n)$

Comparing Time Complexities

$O(1) < O(\log\log n) < O(\log n) < O(\sqrt{n}) < O(n) < O(n\log n) < O(n^k) < O(2^n) < O(n^n)$

as $k \to$ inc

$o \to$ inc

Reccurence Relations

A case where $T(n) = T(r) + k$.

$1 < r < n$

eg: $T(n) = T(\sqrt{n}) + \log n$

Sol: $T(n) = T(\sqrt{n}) + \log n$

$T(n^{1/2}) = T(n^{1/4}) + \log n$

$$T(n) = T(\sqrt{n}) + \log n$$

$$T(n^{1/2}) = T(n^{1/4}) + \frac{1}{2}\log n$$

$$T(n^{1/4}) = T(n^{1/8}) + \frac{1}{4}\log n$$

$$\vdots$$

$$T(n^{1/2^{r-1}}) = T(n^{1/2^r}) + \frac{1}{2^{r-1}}\log n$$

Add all:

$$T(n) = T(n^{1/2^r}) + \log n \left\{ 1 + \frac{1}{2} + \frac{1}{2^2} \cdots + \frac{1}{2^{r-1}} \right\}$$

$$= T(n^{1/2^r}) + \log n \left\{ \frac{1 - 1/2^r}{1/2} \right\}$$

$$= T(n^{1/2^r}) + 2\log n \left(1 - \frac{1}{2^r}\right)$$

$$= T(1) + 2\log n\,(1)$$

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

MASTER THEOREM

$$T(n) = a \cdot T(n/b) + \Theta(n^k (\log n)^p)$$

$a \geq 1$; $b > 1$; $k \geq 0$

Case 1 :

$a > b^k$ $\quad T(n) = n^{\log_b a}$

Case 2 : $a = b^k$

- $p < -1$ $\quad T(n) = n^{\log_b a}$
- $p = -1$ $\quad n^{\log_b a} \cdot (\log n)^2$
- $p > -1$ $\quad n^{\log_b a} \cdot (\log n)^{p+1}$

Case 3 : $a < b^k$

- $p < 0$ $\quad T(n) = O(n^k)$
- $p \geq 0$ $\quad T(n) = n^k (\log n)^p$

eg: $8T(n/2) + n^2 = T(n)$

$a = 8$
$b = 2$
$k = 2$
$p = 0$

$a > b^k$ ✓ $\Longrightarrow$ Case 1 $\quad T(n) = n^{\log_b a}$
$= n^3$

eg: $T(n) = 2T(n/2) + \underbrace{cn}_{\Theta(n)}$

$a = 2$

eg: $T(n) = 2T(n/2) + \underbrace{cn}_{\Theta(n)}$

$a = 2$
$b = 2$
$k = 1$

$[a = b^k]$

Case 2: $p = 0$ $\rightarrow$ $n^{\log_b a} \cdot (\log n)^1$

$= n^1 \cdot \log n$

eg: $T(n) = 3T(n/4) + cn^2$

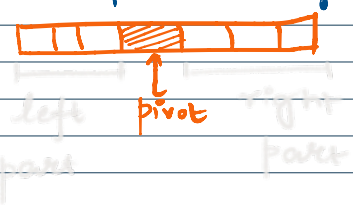
$a = 3$
$b = 4$
$k = 2$
$p = 0$

$\{a < b^k\} \longrightarrow$ Case 3 $\rightarrow$ for $p = 0$ $\rightarrow$ $\Theta(n^2)$

---

1. Quick Sort

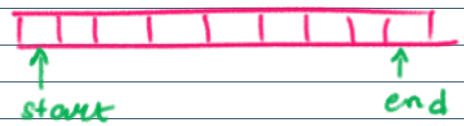
We find a pivot & we split the array on the basis of correct position of pivot.



Then we apply quick sort on the left and right part individually.

Thus they both are sorted ind.

→ Algorithm (partition)




```
pivot = arr[start]
while (start <= end) {
        while (arr[start] < pivot)
                start++;
        while (arr[end] > pivot)
                end--;
        if (start <= end)
                swap(arr[start], arr[end]);
}
swap(pivot, arr[end])
return end
```

⟹ Quick Sort :

```
if(start <= end) {
int index = partition (0, n-1);
QuickSort (0, index - 1);
QuickSort (index +1, n-1);
}
}
```

Analysis

[for worst case]

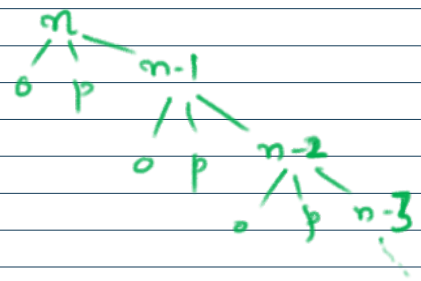
$T(n) = T(n-1) + n$

Best Case : $n \log n$

Worst Case : n^2

(already Sorted)

[ascending or descending]

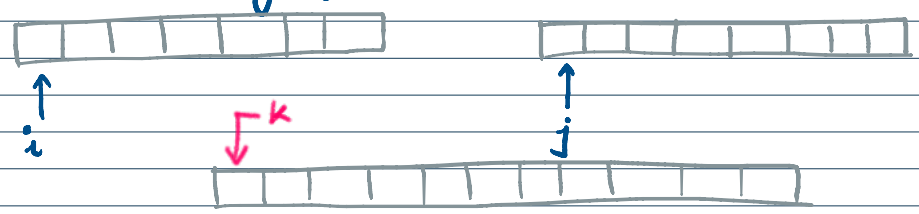


Avg. Case : $n \log n$

2. Merge Sort :

Here the array is split into 2 by the middle & the divided arrays are sorted by merge sort & then finally merged together.

⟶ Algorithm (Merging)




```
if ( i < size(arr1) && j < size(arr2))
  if (arr1 [i] < arr2[j]){
        arr [k] = arr [i];
        i++; }
  if (arr1 [i]  > arr2[j]) {
        arr[k] = arr [j];
        j++; }
  if ( arr1[ i] == arr2 [j]){
        arr [k] = arr [i];
        i++;
        k++;
        arr [k] = arr [j];
        j++; }
  k++;
if ( i >= size(arr1))
```

```
k++;
if ( i >= size(arr1))
    while ( j < size (arr 2)) {
        arr [k] = arr2[ j]
        k++ ;
        j++ ; }

if ( j >= size(arr2))
    while ( i < size (arr 1)) {
        arr [k] = arr [ i]
        k++ ;
        i++ ; }
```

→ Merge Sort

```
mid = (start + end) / 2
merge sort ( start ,  mid        );
merge sort (  mid    +1, end) ;
while ( i < mid) {
    new_arr1[i] = arr [i]; }
while (i < end)
    new_arr2 [ i-mid] = arr [i]; }
merge ( new_arr1, new_arr2) ;
```

Analysis :

All cases : $n \log_2 n$