$f(n) = 3n+2$

$g(n) = n$ (2)

Asymptotic Notation

(1) Big Oh (O) (Worst)

Used for generating worst time or the max. time required by a algo.

$f(n) = O(g(n))$, there exists constants $c > 0$, $n_0 \geq 1$, $n_0 \Rightarrow$ min no. of elements.

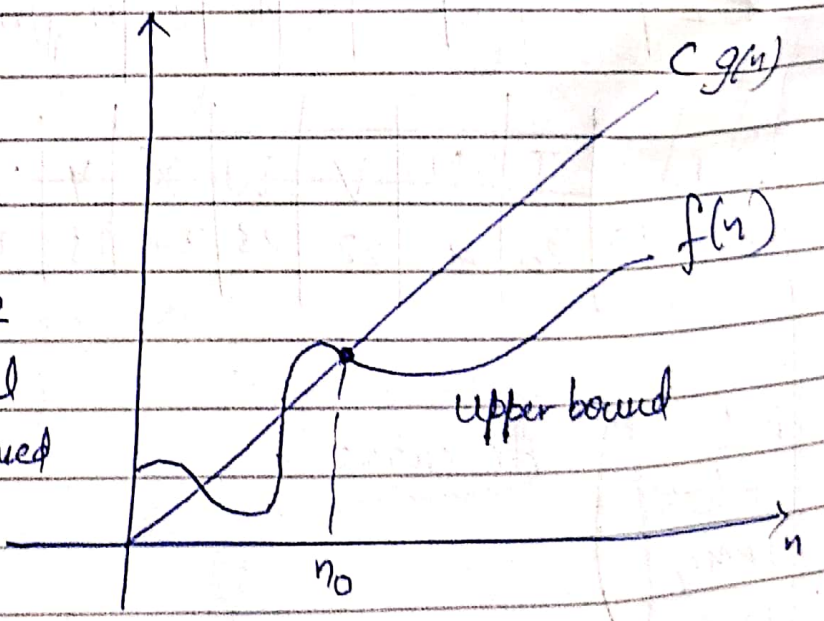
and $f(n) \leq c\,g(n)$

$f(n)$ & $g(n)$ are functions of time

$c \rightarrow$ constant

$n_0 \Rightarrow$ min no. of of elements,

such that $f(n)$ should have lesser or equal value as compared to $c\,g(n)$



Eg $f(n) = 3n+2$, $g(n) = n$

so $f(n) = O(g(n))$

$\therefore f(n) \leq c \cdot g(n)$, $c > 0$ and $n_0 \geq 1$

$3n+2 \leq cn$

$3n+2 \leq 4n$

$n_0 \geq 2$

(best)

(2) Big Omega (Ω)

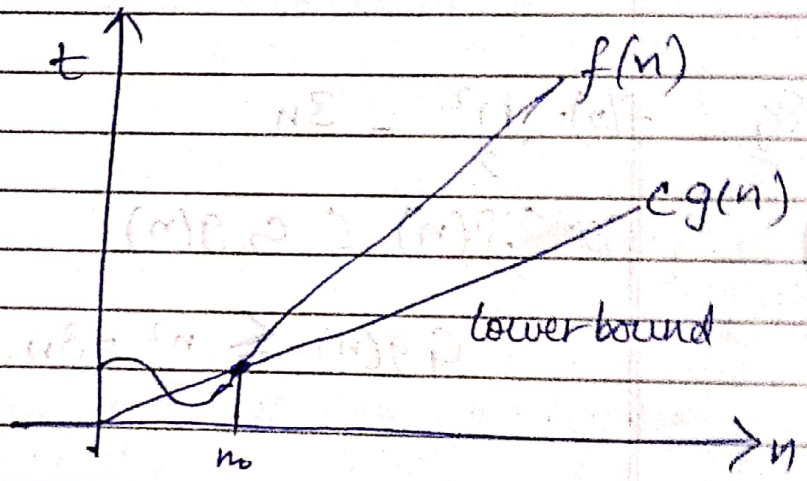
Generates the best time or the min time required by any algorithm.

$f(n) = \Omega(g(n))$ if there exists $c > 0$

& $f(n) \geq c \cdot g(n)$, $n \geq n_0$

$c > 0$, $n_0 \geq 1$

$f(n) \Rightarrow$ actual time
$g(n) \Rightarrow$ min. time



Eg $f(n) \geq 3n+2 = \Omega(n)$

$\therefore f(n) \geq c \cdot g(n)$, $c > 0$ & $n_0 \geq 1$

hence

$3n+2 \geq c \cdot n$

$c = 1$

$n_0 / n_0 = 1$.

$2 \geq (c-3)n$

$2 \geq n(c-3)$

$c - 3 = 2$

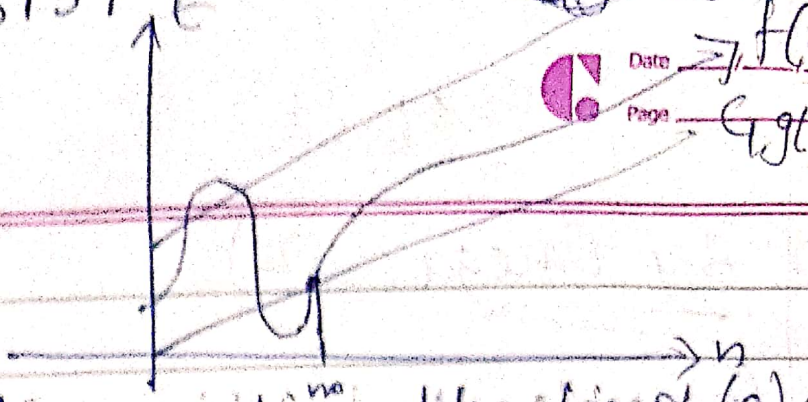
$c = 5$

16218134

Date
Page

average

Theta (Θ)



Generates average time or time b/w beg-oh (O) and minimum, big-omega (Ω)

$f(n) = \Theta(g(n))$; $c_1 > 0$, $c_2 > 0$

and $c_1 g(n) \leq f(n) \leq c_2 g(n)$ $\forall$ $n \geq n_0$ and $n_0 \geq 1$

Eg. $f(n) = \frac{n^2}{2} - 3n$

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

$c_1 g(n) \leq \frac{n^2}{2} - 3n \leq c_2 g(n)$

$c_2 g(n) \geq \frac{n^2 - 3n}{2}$

$c_2 g(n) \geq \frac{1}{2} - \frac{3}{n}$

$c_2 \cdot n \geq \frac{1}{2} - \frac{3}{n}$

$\therefore c_2 \geq \frac{1}{2}$

$c_1 g(n) \leq \frac{n^2 - 3n}{2}$

$c_1 \leq \frac{1}{2} - \frac{3\theta}{n}$

Donor FH

④ Little-Oh (o)

Using big oh notation, we may or may not be able to get asymtotically tight, littleoh is used to denote upper bound asymptotically tight

$f(n) = o(g(n))$, $c > 0$

$n_0 > 0$

and $f(n) < c \cdot g(n)$ $n \geq n_0$.

↳ holds true for all constants in little-oh

Thus, $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

n! , its best example

⑤ Little-Omega (ω)

Using big-Omega notation we may or may not be able to get asymtotically tight, but little-Omega (ω) is used to denote lower bound asymptotically tight

$f(n) = \omega(g(n))$, $c > 0$

$n_0 > 0$

$f(n) > c \cdot g(n)$ $n \geq n_0$

↳ equality is eliminated

$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$

OR

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

Recurrence Relations:

i) Iterative Method

45 Eg $T(n) = 1 + T(n-1)$ here $\begin{cases} 1 & \text{if } n=1 \\ T(n-1) & \text{if } n>1 \end{cases}$

so,

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

$T(n-2) = 1 + T(n-3)$

so as

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

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

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

$= 2 + 1 + T(n-3)$

$= 3 + T(n-3)$

.

.

.

$= k + T(n-k)$

↓

Now apply base value as we need it to stop

so for $n-k = 1$

$k = n-1$

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

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

$= n-1 + T(1)$

$= n-1+1$

$T(n) = n$

$T(n) = O(n)$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \end{cases}$$

$T(n) = 2T\left(\frac{n}{2}\right) + n$

So

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2}$

$T\left(\frac{n}{3}\right) = 2T\left(\frac{n}{6}\right) + \frac{n}{3}$; $T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$

So as $T(n) = 2T\left(\frac{n}{2}\right) + n$

hence $T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 4T\left(\frac{n}{4}\right) + 2n$

$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n \Rightarrow 8T\left(\frac{n}{8}\right) + 3n$

$= 2^k T\left(\frac{n}{2^k}\right) + kn$

So

$k = \log_2 n$

$2^k T\left(\frac{n}{2^k}\right) = 1$

$2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n \times n$

Eg

$$T(n) = \begin{cases} 1 & \text{, if } n=1 \\ n + T(n-1) & \text{if } n>1 \end{cases}$$

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

$T(n-2) = n-2 + T(n-3)$

$\therefore$ SO $T(n) = n + T(n-1)$

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

$= n + (n-1) + (n-2) + T(n-3)$

$= n + (n-1) + (n-2) + \cdots (n-k) + T(n-(k+1))$

$\Downarrow$

$\therefore n-(k+1) = 1$

$n-k-1 = 1$

$\boxed{k = n-2}$

$\therefore$

$(n + (n-1) + (n-2) + \cdots (n-(n-2)) + T(n-(n-2+1))$

$n + (n-1) + (n-2) + \cdots 2 + 1$

$\frac{n(n+1)}{2} \Rightarrow O(n^2)$

Eg:- $T(n) = 3T\left(\frac{n}{4}\right) + n$

$\therefore T\left(\frac{n}{4}\right) = 3T\left(\frac{n}{16}\right) + \frac{n}{4}$

$T\left(\frac{n}{16}\right) = 3T\left(\frac{n}{64}\right) + \frac{n}{16}$

$\therefore$ substituting,

$T(n) = 3T\left(\frac{n}{4}\right) + n$

$= 3\left(3T\left(\frac{n}{16}\right) + \frac{n}{4}\right) + n$

$\Rightarrow 9T\left(\frac{n}{16}\right) + \frac{3n}{4} + n$

$\Rightarrow 9\left(3T\left(\frac{n}{64}\right) + \frac{n}{16}\right) + \frac{3n}{4} + n$

$\Rightarrow 27T\left(\frac{n}{64}\right) + \frac{9n}{16} + \frac{3n}{4} + n$

$\Rightarrow n + \frac{3n}{4} + \frac{9n}{16} + 27T\left(\frac{n}{64}\right) \dots + 3^i T\left(\frac{n}{4^i}\right)$

$\therefore \frac{n}{4^i} = 1$

$i = \log_4 n$

$T(n) = n + \frac{3n}{4} + \frac{9n}{16} + \dots + 3^{\log_4 n} T(1)$

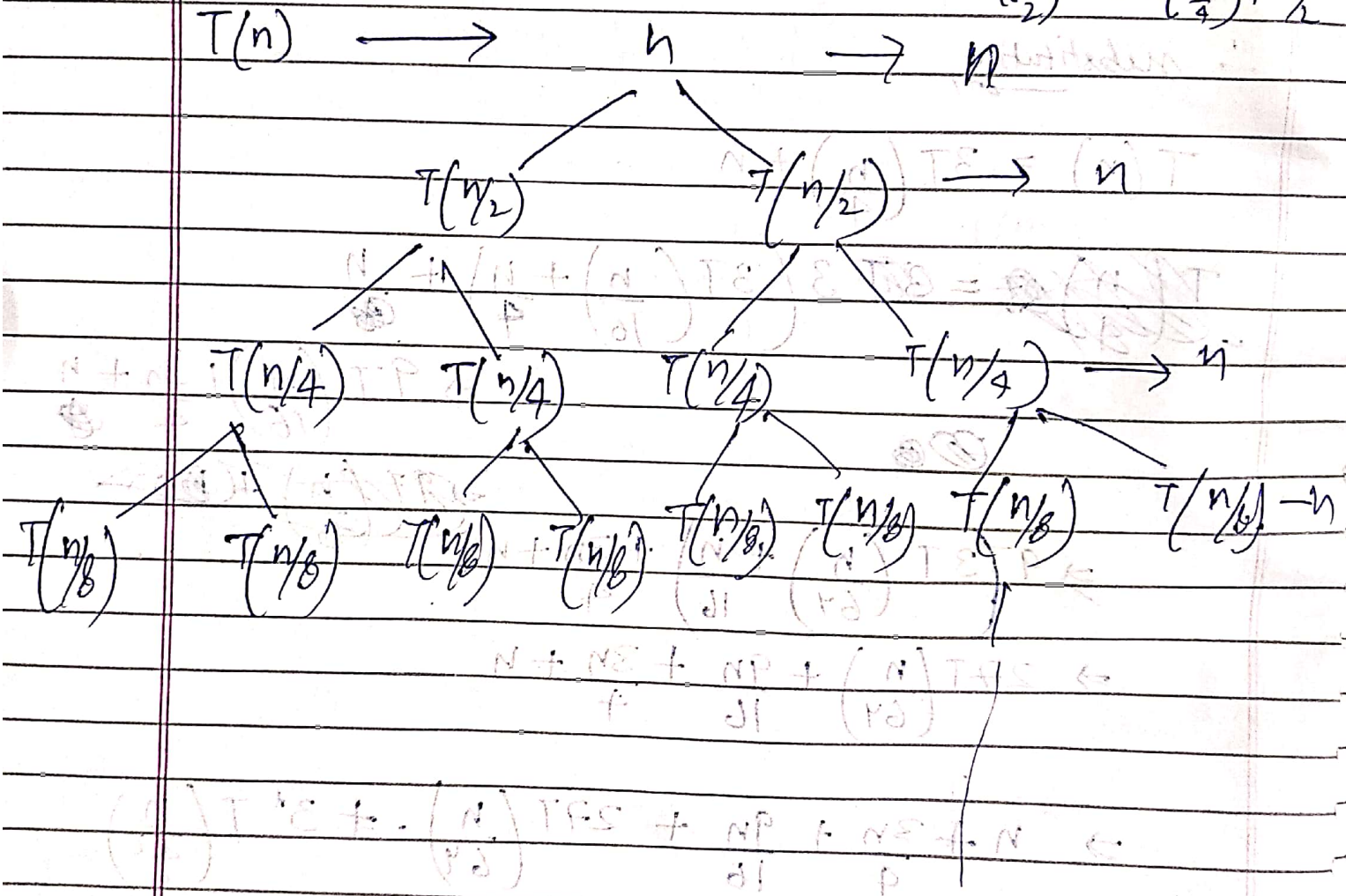
ii) Recursive Tree Method

Eg

$T(n) = 2T\left(\frac{n}{2}\right) + n$, $n > 1$

& $n = 1$ when 1

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + n/2$



$\frac{n}{2^0} \rightarrow \frac{n}{2^1} \rightarrow \frac{n}{2^2} \rightarrow \frac{n}{2^3} \cdots \frac{n}{2^k}^{1}$

$(n) = 2^k$

so $(k+1)$ levels

as $n = 2^k$

$\log n = k \log 2$

$\Rightarrow k \log$

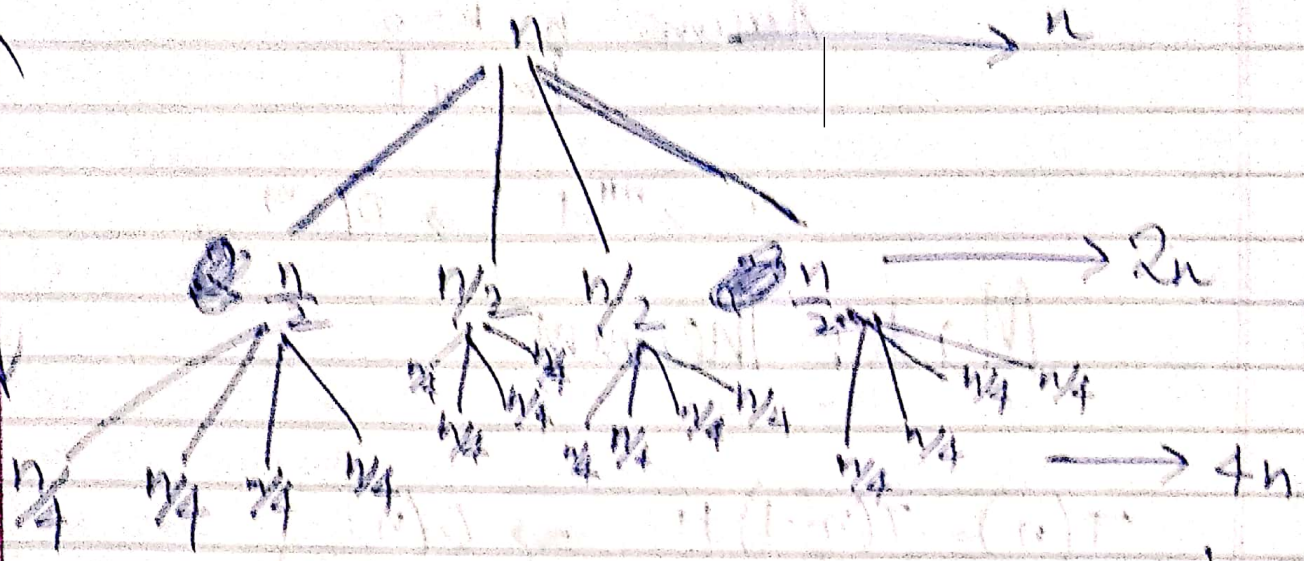
$k = \log n$

$(k+1)$

$(\log n + 1)\, n$

so $O(n \log n)$

Eg) $T(n) = 4T\left(\frac{n}{2}\right) + n$

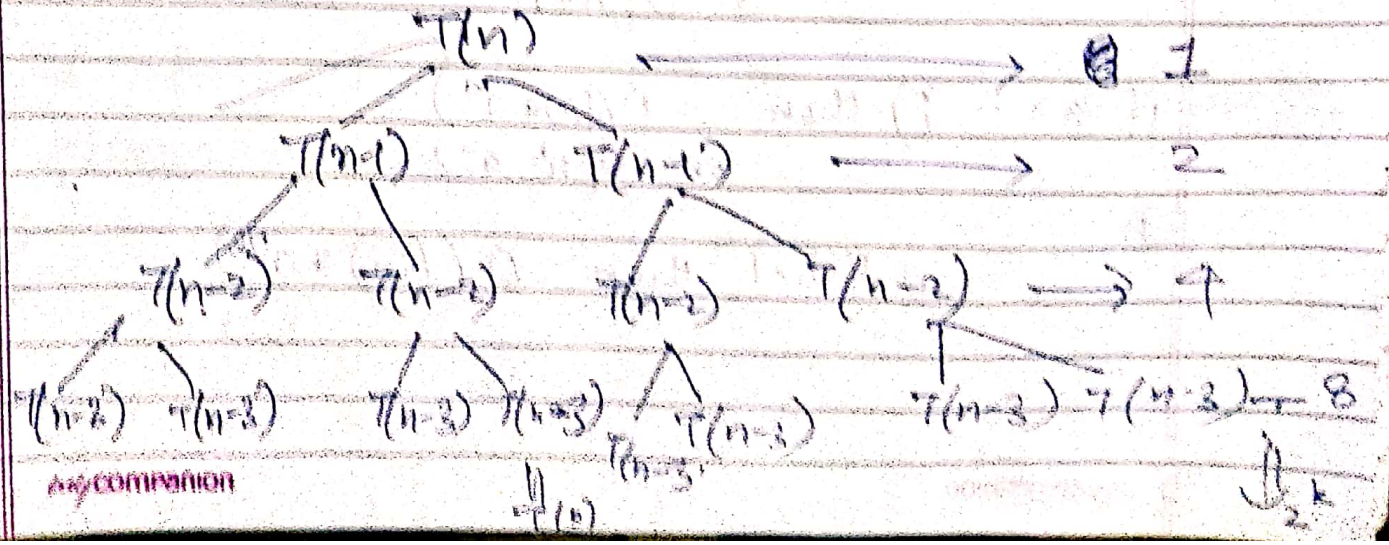


$n + 2n + 4n \ldots$ logn times.

$n(1 + 2 + 4 \ldots)$

$n = \frac{1(2^{\log_2 n} - 1)}{2-1} \rightarrow n^2 - n \Rightarrow O(n^2)$

Eg) $T(n) = \begin{cases} 2T(n-1)+1 & n > 0 \\ 1 & n = 0 \end{cases}$



$1 + 2^1 + 2^2 + 2^3 + \cdots + 2^k = 2^{k+1} - 1$

Assume $n - k = 0$

$k = n$

$\therefore 2^{n+1} - 1 \Rightarrow O(2^n)$

Master Theorum

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

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

$T(n) = T(n-1) + \log n \Rightarrow O(n \log n)$

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

$\rightarrow f(n)$

So, for form: General form

$T(n) = aT(n-b) + f(n)$ $\quad O(n^k \log^p n)$

$a > 0, \; b > 0$ & $f(n) = n^k$ where $k \geq 0$

if $a = 1$ then $O(n^{k+1})$

or $O(n * f(n))$

if $a > 1$ i) then $O(f(n) * a^n)$

or $O(n^k \cdot a^n)$

ii) if $b > 1$ then $O(f(n) * a^{n/b})$

It provides a "cook book" for solving:-

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$a \geq 1$ & $b > 1$

It divides problem into 'a' subproblems each of size $\frac{n}{b}$,

'a' subproblems are solved recursively, each in time $T\left(\frac{n}{b}\right)$.

The cost of dividing the problem and combining the results of the subproblem is described by the $f(n)$.

We either replace each of the a terms $T\left(\frac{n}{b}\right)$ with either

$T\left(\left\lceil \frac{n}{b} \right\rceil\right)$ or $T\left(\left\lfloor \frac{n}{b} \right\rfloor\right)$.

The $T(n)$ can be bounded asymptotically as follows:-

(1) If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$

$$T(n) = \Theta(n^{\log_b a})$$

(2) If $f(n) = \Theta(n^{\log_b a})$ then

$$T(n) = \Theta(n^{\log_b a} \log_2 n)$$

(3) If $f(n) = \Omega(n^{\log_b a + \epsilon})$, for $\epsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$; $c < 1$ then $T(n) = \Theta(f(n))$

(regularity cond.)

Cont'd. After 1 Page

if $n^{\log_b a} > f(n) \rightarrow (1)$

If $n^{\log_b a} < f(n) \rightarrow$ then regularity cond. should also satisfy so (3)

If $n^{\log_b a} = f(n) \Rightarrow (2)$

but not polynomically

Master

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$, and $f(n) = \Theta(n^k \log^p n)$

Case 1: if $\log_b a > k$ then $\Theta(n^{\log_b a})$

Case 2: $\log_b a = k$

if $p > -1$ $\Theta(n^k \log^{p+1} n)$

if $p = -1$ $\Theta(n^k \log \log n)$

if $p < -1$ $\Theta(n^k)$

Case 3: if $\log_b a < k$

if $p \geq 0$ $\Theta(n^k \log^p n)$

if $p > 0$ $\Theta(n^k \log^p n)$

if $\log_b a > k$, $\Theta(n \log_b a)$

if $\log_b a = k$ $\Theta(n^{\log_b a} \log n)$

if $\log_b a < k$ $\Theta(f(n))$

$\Theta(n^k)$

Eg- $T(n) = T\left(\frac{2n}{3}\right) + 1$

$a = 1$

$b = 3/2$

$f(n) = 1$

No $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

As $n^{\log_b a} < f(n)$ so check for regulatory condition,

$af\left(\frac{n}{b}\right) \leq cf(n)$,

$1f\left(\frac{1}{3/2}\right) \leq c1$

$\frac{2}{3} \leq c1$.

$\therefore T(n) = \theta(n^{\log_b a} \cdot \lg n) \Rightarrow \theta(1 \lg n)$

Eg= $T(n) = 3T\left(\frac{n}{4}\right) + n\log n$

$a = 3, b = 4, f(n) = n\lg n$.

$n^{\log_a b} = n^{\log_4 3} = \theta(n^{0.793})$

$f(n) = \theta(n^{(\log_4 3 + \epsilon)})$ so case ③ applies

$\epsilon = 0.2 - -0$

hence

$T(n) = \theta(n\log n)$. i.e $\theta(f(n))$.

$n = 2^k$ $\quad k = \frac{\log_2 n}{\log_2 2}$

$\log_2 n = k \log_2 2$

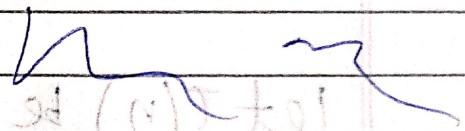
Date ___/___/___

Page _____

Eg

$$T(n) = 2T\left(\frac{n}{2}\right) + n\log n$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$



$a = 2$

$b = 2$

$f(n) = n\log n$

$n^{\log_b a} = n^{\log_2 2} = n^1 = n$

as $n < f(n)$ is asymptotically larger than $n^{\log_b a}$ but not polynomically

Watch

Subtract & Conquer Method

Let $T(n)$ be a func defined on the n

$$T(n) = \begin{cases} c, & \text{if } n \geq 1 \\ a\,T(n-b) + f(n) & \text{if } n > 1 \end{cases}$$

for some constant $c, a>0, b>0, d \geq 0$ & $f(n)$ if $f(n)$ is in $O(n^d)$ then

$$T(n) = \begin{cases} O(n^d), & \text{if } a < 1 \\ O(n^{d+1}), & \text{if } a = 1 \\ O(n^d a^{n/b}), & \text{if } a > 1 \end{cases}$$

~~$\Rightarrow T(n) = T(n-1) + n$ if $n>1$ $\Rightarrow$ and $T(n-1) = 1$~~

$\frac{a(r^n-1)}{r-1}$ for $\sum_{i=0}^{n} a^i$ Finite GP.

$\frac{a}{1-r}$ for $\sum_{i=0}^{n} \frac{1}{a^i}$ If r is < 1

Eg 1 $T(n) = T(n-1) + n$ if $n>1$ then $T(1) = 1$

$c=1,\ a=1,\ b=1,\ f(n) = n^{\textcircled{1}}$ so $d=1$

as $a=1$

$\Rightarrow$ so $T(n) = O(n^{d+1}) = O(n^2)$.

Proof

$$T(n) = \begin{cases} c & \text{if } n \le 1 \\ aT(n-b) + f(n) & \text{if } n > 1 \end{cases}$$

Ans as $T(n-b) = aT(n-b-b) + f(n-b)$

$$T(n) \le a\left[aT(n-2b) + f(n-b)\right] + f(n)$$

$$T(n) = a^2 T(n-2b) + af(n-b) + f(n)$$

~~$a^3 T$~~ ⇓ for one more value

$$= a^3 T(n-3b) + a^2 T(n-2b) + af(n-b) + f(n)$$

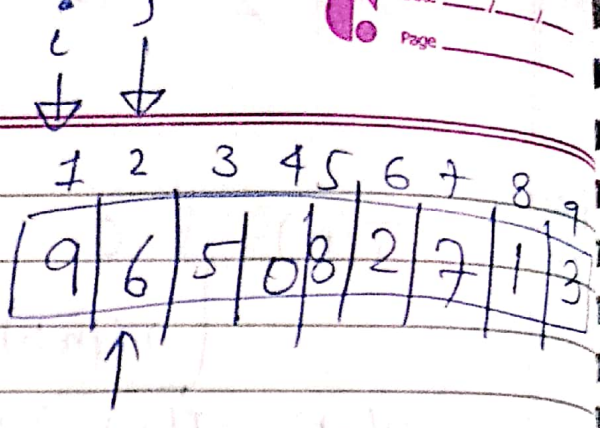
$$T(n) = \sum_{i=0}^{n/b} a^i f(n-ib) + \text{constant}$$

Now $f(n-ib)$ should be equal $O(n^d)$

$$T(n) = O\left(n^d \sum_{i=0}^{n/b} a^i\right)$$

$O(1)$ if $a < 1$
$O(n)$ if $a = 1$
$O(a^{n/b})$ if $a > 1$

Insertion Sort




```
Insertion_Sort (A)
{
  for j = 2 to A.length
      key = A[j]   // key is made because we might let over write it
  // insert A[j] into sorted sequence A[1 ... j-1].
      i = j - 1
  while (i > 0 and A[i] > key)
      A[i+1] = A[i]
      i = i - 1
  A[i+1] = key
}
```

Array Index	Moves	Comparison	Total
2	1	1	1+1 = 2 2(2-1)
3	2	2	2+2 = 4 2(3-1)
4	3	3	3+3 = 6 2(4-1)
5	4	4	4+4 = 8 2(4)
⋮			
n	(n-1)	(n-1)	2(n-1) = 2(n-1)

Worst Case (Reversed arranged)

$j = 2 \quad 3 \quad \ldots \quad n$

Move & comparisons $2(1) + 2(2) + \ldots \ldots 2(n-1)$

$2(1 + 2 + \ldots \ldots (n-1)) \Rightarrow \frac{2(n-1)(n)}{2} \Rightarrow O(n^2)$

Best Case

$j = 2 \quad 3 \quad \ldots \ldots \ldots (n-1)$

Only comparisons as both already sorted: $1 \quad 1 \quad \ldots \ldots \ldots 1$

so $(n-1)$ comparisons and 0 movements.

$O(n-1)$ so $\Omega(n-1) \Rightarrow \Omega(n)$.

OR

$$T(n) = C_1 n + C_2(n-1) + C_3(n-2) + C_4 \sum_{i=2}^{n} t_j + C_5 \sum_{i=2}^{n} t_j - 1$$

$$+ C_6 \sum_{i=2}^{n} t_j - 1 + C_7(n-1)$$

$\Rightarrow C_1 n + C_2(n-1) + C_3(n-2) + C_4(n-1) + C_7(n-1)$

$= an + 6 \Rightarrow T(n) = O(n)$.

$$T(n) = C_1 n + C_2(n-1) + C_3(n-2) + C_4\left[\frac{n(n+1)}{2} - 1\right] + C_5\left[\frac{(n-1)n}{2}\right]$$

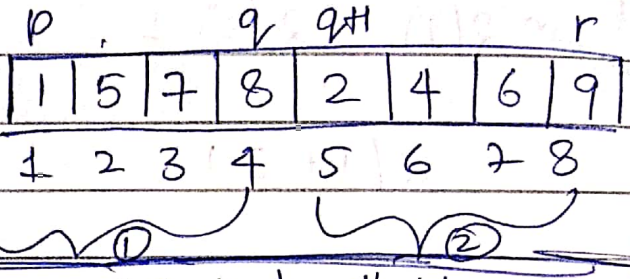
$$+ C_6 \frac{n(n-1)}{2} + C_7(n-1)$$

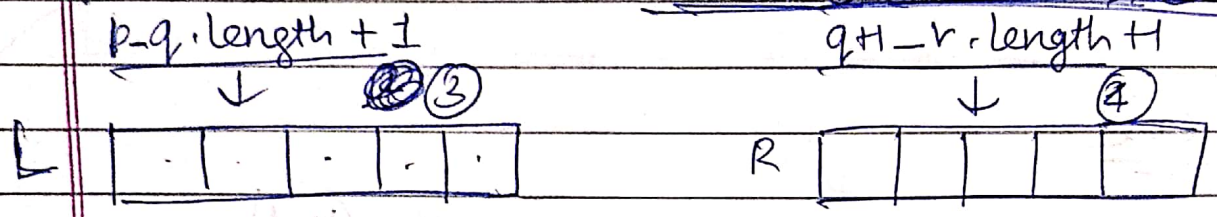
$T(n) = An^2 + Bn + C$

$= An^2 + Bn^2 + Cn^2$

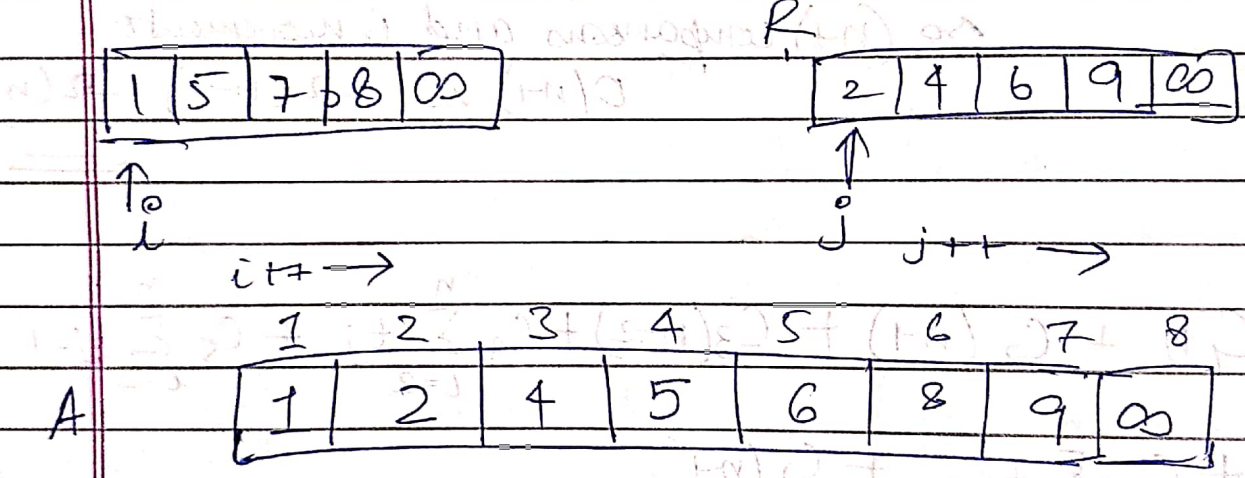
$T(n) = O(n^2)$

Merge Sort



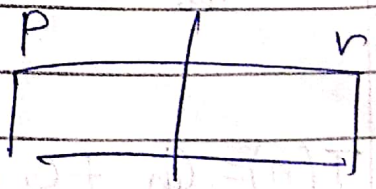


Now copy ① into ③ & ② into ④



Compare i & j and write

Algo




```
merge_sort(A, P, r)
if p < r
    q = [(p+r)/2]
merge-sort (A, P, q)
merge-sort (A, q+1, r)
merge (A, P, q, r)
```

Merge (A, p, q, r)

{

$n_1 = (q - p) + 1$

$n_2 = r - q$

Let $L[1 \ldots n_1+1]$ and $R[1 \ldots n_2+1]$ new arrays.

for (i = 1 to n_1)

    $L[i] = A[p + i - 1]$

for (j = 1 to n_2)

    $R[j] = A[q + j]$

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

$L[n_1+1] = \infty$

$R[n_2+1] = \infty$

i = 1 && j = 1;

for (k = p to r) RH

if ($L[i] \leq R[j]$)

    A[k] = L[i]

    i++

else (A[k] = R[j])

    j++

}

if ($L[i] \leq R[j]$)

    A[k] = L[i]

    i++

else if ($L[i] > R[j]$)

    A[k] = R[i]

    j++

else ($L[i] == R[j]$)

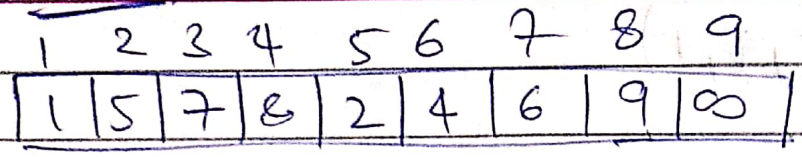
    A[k] = L[i].

    i++;

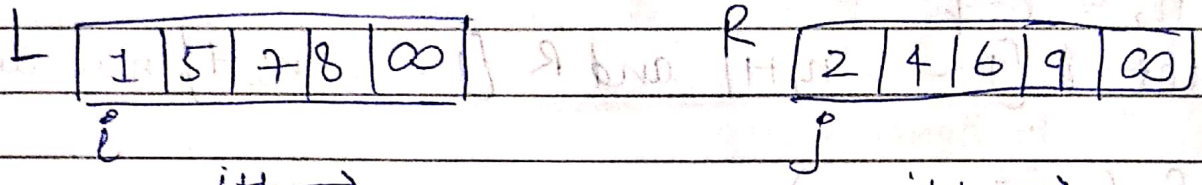
    j++

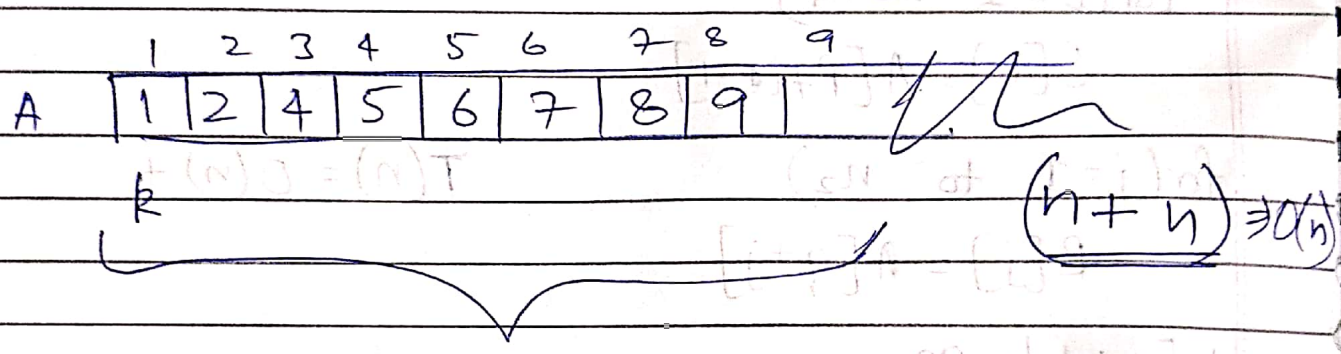
}

Analysis (Merge)



copying time O(n)





If total no. of elements in A = n, then
in order to write 1 element to A,
1 comparison b/w L and R is required.

So for n elements, 'n' comparisons.

&

after every comparison, one of them has to be written,

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ 2T(n/2) + O(n) & \text{if } n>1 \end{cases}$$

so 'n' writtens or n writes.

so $O(n) + O(n)$

$O(n+n)$

but

If size of arrays differ like n & m so,

$O(n+m)$

$T(n) = O(1) + T(n/2) + T(n/2) + O(n)$.

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

$= O(n \log n)$

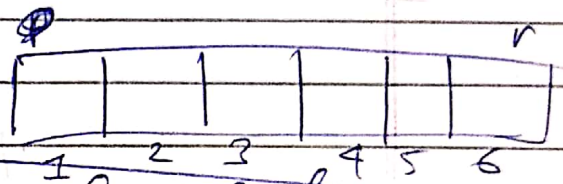
else

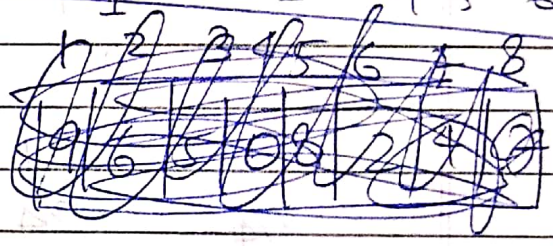
$j = j+1$

$T(n) = O(1) + T(n/2) + T(n/2) + O(n)$

$= 2T(n/2) + O(n)$ [Using Master's theorum].

$= O(n \log n)$.





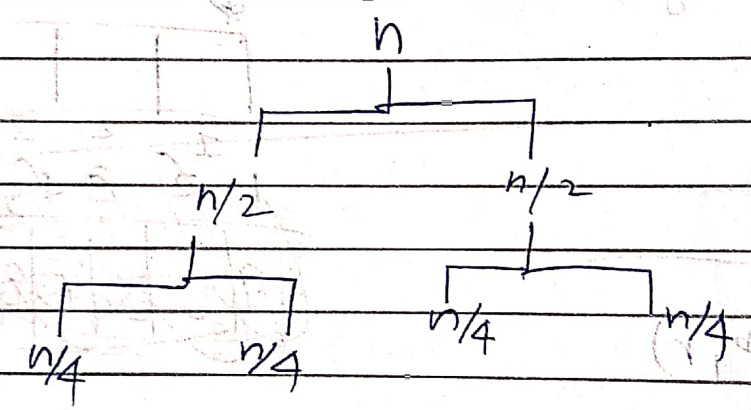
Quick Sort

```
Partition (A, p, r)
{
  x = A[r]   // Pivot or last element
  i = p-1

  for (j = p to r-1)
  {
    if (A[j] ≤ x)
    {  i = i+1
       exchange A[i] with A[j]
    }
    else (A[j] > x)
       j = j+1 }

}
exchange A[i+1] with A[r]
return i+1
}
```

It makes it one element come to its correct position so $O(n)$

```
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)
  }
}
```

1	2	3	4	5	6	7
5	7	6	1	3	2	4



Worst Case

$$T(n) = O(n) + T(n-1) + T(1)$$
$$= O(n) + T(n-1) + T(n-2) \text{ ---- } + T(n)$$
$$= O\left(\sum_{i=1}^{n} n\right)$$
$$= O\left(\frac{n(n+1)}{2}\right)$$
$$= O(n^2)$$

Best Case

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

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

$$= O(n \log n) \qquad \Omega(n \log n)$$

Average Case

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

$$T(n) \geq \left(\frac{\log n + 1}{4}\right)^n$$

$$T(n) \geq \frac{1}{k_2}(n \log n + n)$$

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

28
72