DAA [DESIGN & ANALYSIS OF ALGORITHMS] Dated:

```
Algorithm sum(A,n)
{
    s = 0;  ————————————— 1
    for(i = 0; i<n; i++) —— n+1   (runs n+1 times)
    {
        s = s + A[i]; ——— n   (runs n times)
    }
    return s; ————————— 1
}
```

$f(n) = 2n+3$ $\quad O(n)$

Space :- size

A — n
n — 1
s — 1
i — 1

$S(n) = n+3 \quad -O(n)$

```
Algorithm Add(A,B,n)
{   for(i=0; i<n; i++) ——— n+1
    {
        for(j=0; j<n; j++) ——— n x (n+1)
        {   C[i,j] = A[i,j] + B[i,j]; — n x n
        }
    }
}
```

$f(n) = 2n^2 + 2n + 1$

$O(n^2)$

Space 2-

A — n^2
B — n^2
C — n^2
n — 1
i — 1
j — 1

$f(n) = 3n^2 + 3$

$O(n)$

Dated:

Algorithm Multiply (A, B, n)

```
{ for (i=0; i<n; i++)  ——— n+1
  {
    for (j=0; j<n; j++)  ——— n (n+1)
    { c[i,j] = 0;  ——— n (n)
      for (k=0; k<n; k++) —— n (n) (n+1)
      {
        c[i,j] = c[i,j] + A[i,k] * B[k,j]; — n(n)(n)
      }
    }
  }
```

$f(n) = 2n^3 + 3n^2 + 2n + 1$

$O(n^3)$.

Space :-

A — n^2
B — n^2
C — n^2
n — 1
i — 1
j — 1
k — 1

$S(n) = 3n^2 + 4$

$O(n^2)$

Time Complexity

① `for (i=0; i<n; i++)` — n+1
`{ stmt; — n }`

$O(n)$

② `for (i=n; i>0; i--)`
`{ stmt; — n }`

$O(n)$

③ `for (i=1; i<n; i=i+2)`
`{ stmt; —— n/2 }`

degree of polynomial is still n.

$f(n) = n/2$
$O(n)$

④ `for (i=0; i<n; i++)` — n+1
`{ for (j=0; j<n; j++) — n(n+1)`
`{ stmt; — n x n }`

$O(n^2)$

```
for (i=0; i<n; i++)
{
  for (j=0; j<i; j++)
  {
    stmt;
  }
}
```

i	j	no. of times statement executes
0	0	0
1	0	1
	1 x	
2	0	2
	1	
	2 x	
3	0	3
	1	
	2 3 x	
n	0 ...	n

$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$

$f(n) = \frac{n^2+1}{2}$

$O(n^2)$

Dated:

⑥
```
p = 0;
for (i=1; p<=n; i++)
{   p = p + i;
}
```

i	p
1	0+1 = 1
2	1+2 = 3
3	3+3 = 6
⋮	
k	1+2+3+4+---+k

Assume $p > n$

$$p = \frac{k(k+1)}{2}$$

$$\frac{k(k+1)}{2} > n$$

$$k^2 > n$$

$$k > \sqrt{n} \quad \therefore O(\sqrt{n})$$

⑦
```
for (i=1; i<n; i=i*2)
{   stmt;
}
```

Assume $i >= n$

$\therefore i = 2^k$

$2^k >= n$

$2^k = n$

$k = \log_2 n$ $\quad \boxed{O(\log_2 n)}$

i
1
1×2 = 2
2×2 = 4 = 2^2
$2^2 \times 2 = 2^3$
$2^3 \times 2 = 2^4$
⋮
2^k

⑧
```
for (i=1; i<n; i=i*2)
{   stmt;   ---- log2n
}
```

$i = 1 \times 2 \times 2 \times 2 \cdots = n$ (k times)

$2^k = n$

$k = \log_2 n$

Ex:- $n = 8$

i
1
2
4

runs 3 times

8 ×

$n = 10$

i
1
2
4
8

runs 4 times

16 ×

⑨
```
for (i=1; i<=n; i++)
{   stmt;
}
```

$i = 1 + 1 + 1 + 1 \cdots + 1 = n$ (k times)

$k = n$

Dated:

(10) `for(i=n; i>=1; i=i/2)`
`{ stmt; }`

i = n
i = i/2
n/2
$n/2^2$
$n/2^3$
.
.
$n/2^k$

Assume $i < 1$

$\therefore \frac{n}{2^k} < 1$

$n < 2^k$, $2^k > n$

$k = \log_2 n$

$O(\log_2 n)$

(11) `for(i=0; i*i<n; i++)`
`{ stmt; }`

Assume $i * i >= n$

$i^2 \geq n$

$i = \sqrt{n}$

(12) `p=0`
`for (i=1; i<n; i=i*2)`
`{ p++; }` —— $p = \log n$

`for (j=1; j<p; j=j*2)`
`{ stmt; }` —— $\log p$

$O(\log \log n)$

Consider this as a pattern :-

`for (i=1; i<n; i=i*2)`
`{ stmt; }` —— $\log n$

`for (i=1; i<p; i=i*2)`
`{ stmt; }` —— $\log p$

(13)
```
for (i=0; i<n; i++)        —— n+1 → can take it as n.
{
    for (j=1; j<n; j=j*2)  —— n x log n
    { stmt; }              —— n x log n
}
```

$2n \log n + n$

$O(n \log n)$

(14)
```
for(i=0; i<n; i++)          - O(n)
    for(j=0; i<n; i=i+2)    n/2 → O(n)
for(i=n; i>1; i--)          - O(n)
for(i=1; i<n; i=i*2)        - O(log2 n)
for(i=1; i<n; i=i*3)        - O(log3 n)
for(i=n; i>1; i=i/2)        - O(log2 n)
```

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \_\_\_ < 2^n < 3^n < \_\_ < n^n$

Asymptotic Notations :-

- O big-oh upper bound
- Ω big-omega lower bound
- Θ theta Average bound

Dated:

Big - Oh :-

$f(n) = O(g(n))$ iff there are two positive constants c & n_0 such that $|f(n)| \leq c|g(n)|$ for all $n \geq n_0$.

- If $f(n)$ is nonnegative, we can simplify the last condition to $0 \leq f(n) \leq c\,g(n)$ for all $n \geq n_0$.
- $g(n)$ is an asymptotic upper bound on $f(n)$.

Eg:- $f(n) = 2n+3$

$2n+3 \leq 10n$, $n \geq 1$ or $2n+3 \leq 5n$, $n \geq 1$

($f(n)$, c, $g(n)$) → Any $c.g(n)$ is possible here just $|f(n)| \leq c|g(n)|$, just this should be followed.

$\therefore f(n) = O(n)$

$2n+3 \leq 5n^2$

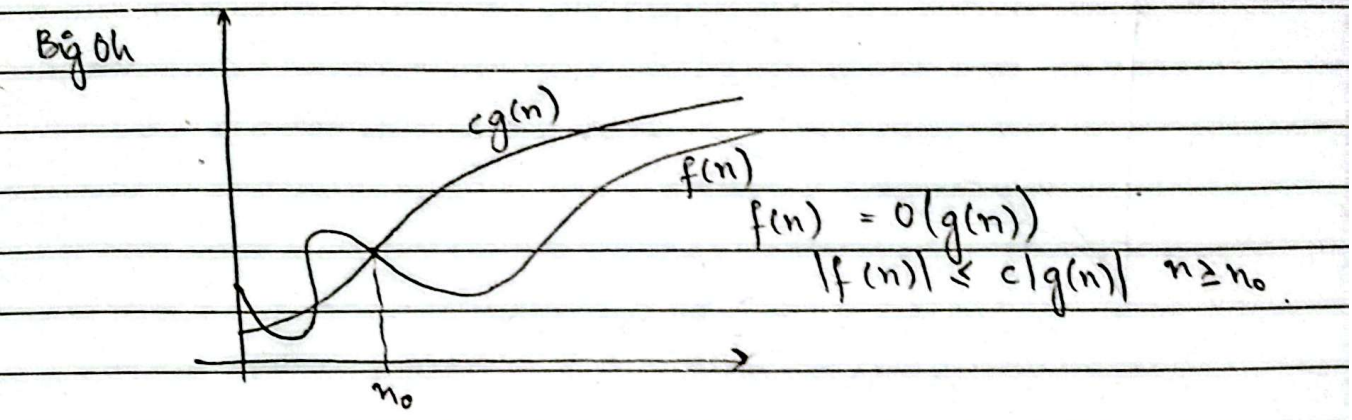
↳ this is also valid

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

↑ since this tells the upper bound.

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \text{---} < 2^n < 3^n < \text{---} < n^n$

Lower bound — $f(n)$ Average bound — Upper bound



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

$|f(n)| \leq c|g(n)|$ $n \geq n_0$.

Proof:- $f(n) = n^2 + n$, $g(n) = n^3$.

$n \geq 1$, $n^2 + n \leq n^3$

$f(n) \leq c \cdot g(n)$, $n \geq 1$

Big - Ω notation :-

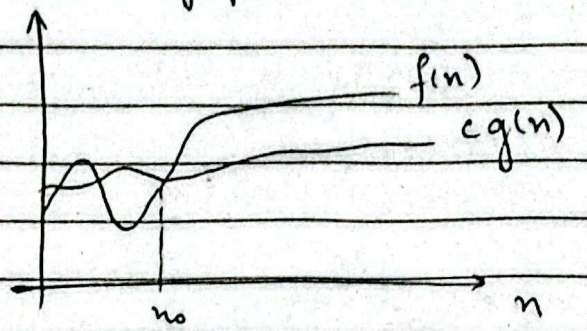
$f(n) = \Omega(g(n))$ iff there are two positive constants c & n_0 such that $|f(n)| \geq c|g(n)|$ for all $n \geq n_0$.

- If $f(n)$ is nonnegative, we can simplify the last condition

Dated:

to :- $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$.

∘- $g(n)$ is an asymptotic lower bound on $f(n)$.



$f(n) = \Omega(g(n))$

E.g:- $n^3 + 4n^2 = \Omega(n^2)$

$n^3 + 4n^2 \geq n^2$

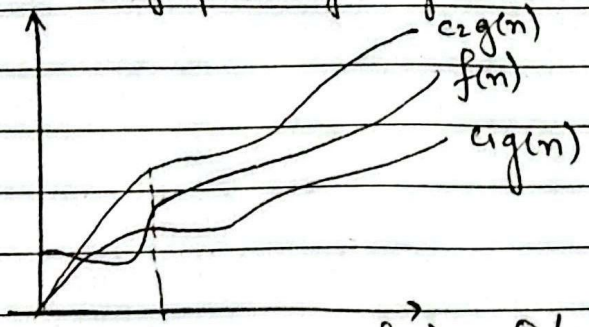
$n \geq 1 \quad n^3 \geq n^2$

Big Theta Θ notation :-

$f(n) = \Theta(g(n))$ iff there are three positive constants c_1, c_2 and n_0 such that $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$ for all $n \geq n_0$.

∘- If $f(n)$ is nonnegative, we can simplify the last condition to $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

∘- $g(n)$ is asymptotically tight bound on $f(n)$.



$f(n) = \Theta(g(n))$

E.g:- $n^2 + 5n + 7 = \Theta(n^2)$

∘- when $n \geq 1$,

$n^2 + 5n + 7 \leq n^2 + 5n^2 + 7n^2 \leq 13n^2$

∘- when $n \geq 0$,

$n^2 \leq n^2 + 5n + 7$

∘- Thus, when $n \geq 1$

$n^2 \leq n^2 + 5n + 7 \leq 13n^2$

$n_0 = 1, \quad c_1 = 1, \quad c_2 = 13$

Dated:

Arithmetic of Big-O, Ω and Θ notations :-

- Transitivity :-
  - $f(n) \in O(g(n))$ & $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$
  - $f(n) \in \Theta(g(n))$ & $g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$
  - $f(n) \in \Omega(g(n))$ & $g(n) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n))$
- Scaling :- If $f(n) \in O(g(n))$ then for any $k > 0$, $f(n) \in O(kg(n))$
- Sums :- $f_1(n) \in O(g_1(n))$ & $f_2(n) \in O(g_2(n))$ then $(f_1+f_2)(n) \in O(\max(g_1(n), g_2(n)))$

□ $f(n) = 2n^2 + 3n + 4$

$2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2$

$2n^2 + 3n + 4 \leq 9n^2 \quad n \geq 1$

$g(n)$

$O(n^2)$

$2n^2 + 3n + 4 \geq n^2$

$\Omega(n^2)$

$n^2 \leq 2n^2 + 3n + 4 \leq 9n^2$

$\Rightarrow \boxed{\Theta(n^2)}$

□ $f(n) = n^2 \log n + n$

$n^2 \log n \leq n^2 \log n \leq 10 n^2 \log n$

$O(n^2 \log n) \quad \Omega(n^2 \log n)$

$\Rightarrow \boxed{\Theta(n^2 \log n)}$

□ $f(n) = n! = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1$

$1 \times 1 \times 1 \times \dots \times 1 \leq 1 \times 2 \times 3 \times \dots \times n \leq n \times n \times n \dots \times n$

$1 \leq n! \leq n^n$

$\Rightarrow \boxed{\Omega(1) \quad O(n^n)}$

□ $f(n) = \log n!$

$1 \leq \log n! \leq \log n^n$

$n \log n$

$\Rightarrow \boxed{\Omega(1) \quad O(n \log n)}$

□ Show that $(n \log n - 2n + 13) = \Omega(n \log n)$

$n \log n - 2n \leq n \log n - 2n + 13$

$n \log n \leq n \log n - 2n$

$\therefore g(n) \leq f(n)$

Dated:

c. $\frac{n\log(n)}{n\log(n)} \leq \frac{n\log(n) - 2n}{n\log(n)}$

$c \leq 1 - \frac{2n}{n\log n}$, $c \leq 1 - \frac{2}{\log n}$, when $n > 1$

∴ If $n \geq 8$, then $2/(\log n) \leq 2/3$, & picking $c = 1/3$ suffices.

Thus, if $c = 1/3$ & $n_0 = 8$, then for all $n \geq n_0$, we have

$0 \leq cn\log n \leq n\log n - 2n \leq n\log n - 2n + 13$

Thus $(n\log n - 2n + 13) = \Omega(n\log n)$.

PROPERTIES OF ASYMPTOTIC NOTATIONS :-

∴ If $f(n)$ is $O(g(n))$ then $a * f(n)$ is $O(g(n))$

eg:- $f(n) = 2n^2 + 5$ is $O(n^2)$

then $7 \cdot f(n) = 7(2n^2 + 5) = 14n^2 + 35$ is $O(n^2)$

Reflexive :-

If $f(n)$ is given then $f(n)$ is $O(f(n))$

e.g :- $f(n) = n^2$ $O(n^2)$

Transitive :- If $f(n)$ is $O(g(n))$ & $g(n)$ is $O(h(n))$ then $f(n)$ is $O(h(n))$.

e.g :- $f(n) = n$, $g(n) = n^2$ $h(n) = n^3$

n is $O(n^2)$ and n^2 is $O(n^3)$

then n is $O(n^3)$.

Symmetric :- If $f(n)$ is $\Theta(g(n))$ then $g(n)$ is $\Theta(f(n))$.

e.g :- $f(n) = n^2$ $g(n) = n^2$

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

$g(n) = \Theta(n^2)$.

Transpose Symmetric :- if $f(n) = O(g(n))$ then $g(n)$ is $\Omega(f(n))$

e.g :- $f(n) = n$ $g(n) = n^2$

then n is $O(n^2)$ and n^2 is $\Omega(n)$.

If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

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

$\therefore f(n) = \Theta(g(n))$.

Dated:

- if $f(n) = O(g(n))$ & $d(n) = O(e(n))$
  then $f(n) + d(n) = O(\max(g(n), e(n)))$

eg: $f(n) = n^2 = O(n)$
$d(n) = n = O(n^2)$
$f(n) + d(n) = n + n^2 = O(n^2)$

- if $f(n) = O(g(n))$ and $d(n) = O(e(n))$ then
  $f(n) * d(n) = O(g(n) * e(n)) = n^3 = n(n^2)$

COMPARISION OF FUNCTIONS :-

$n^2 < n^3$

$2^2 = 4$ $2^3 = 8$
$3^2 = 9$ $3^3 = 27$
$4^2 = 16$ $4^3 = 64$

or check by Applying log on Both sides.

$\log n^2$ $\log n^3$
$2\log n < 3\log n$

$f(n) = n^2 \log n > g(n) = n(\log n)^{10}$

Apply log
$\log[n^2 \log n]$ $\log n + \log(\log n)^{10}$
$\log n^2 + \log\log n$ $\log n + 10 \log\log n$
$2\log n + \log\log n >$

$f(n) = 3n^{\sqrt{n}}$ $g(n) = 2^{\sqrt{n}\log n}$

$3n^{\sqrt{n}}$ $\sqrt{n}\log n \Rightarrow \log n^{\sqrt{n}}$ $(\log a^b = b\log a)$

$2^{\log_2 n^{\sqrt{n}}}$ ← by default base of log is 2.

$3n^{\sqrt{n}}$ $(n^{\sqrt{n}})^{\log_2 2}$ $(a^{\log_c b} = b^{\log_c a})$

$\frac{1}{3n^{\sqrt{n}}}$ $n^{\sqrt{n}}$

↳ value wise this fn is greater
but asymptotically they are equal
$O(3n^{\sqrt{n}}) = O(n^{\sqrt{n}})$

LOG PROPERTIES :-

$\log ab = \log a + \log b$
$\log a/b = \log a - \log b$
$\log a^b = b \log a$
$a^{\log_c b} = b^{\log_c a}$
↳ $a^{\log_c b} = b^{\log_c a}$
$a^b = n$ then $b = \log_a n$

Dated:

$f(n) = n^{\log n} < g(n) = 2^{\sqrt{n}}$

Apply $\log_2$

$\log n^{\log n}$ $\quad$ $\log 2^{\sqrt{n}} = \sqrt{n} \log_2 2$

$\log n \log n < \sqrt{n}$

$\log^2 n$ $\quad$ $\sqrt{n} = n^{1/2}$

$2 \log \log n$ $\quad$ $\frac{1}{2} \log n$

$f(n) = 2^{\log n} < g(n) = n^{\sqrt{n}}$

$\log n \times \log_2 2$ $\quad$ $\sqrt{n} \log n$

$\log n$ $\quad$ $< \sqrt{n} \log n$

$f(n) = 2n \qquad g(n) = 3n$

Asymptotically, both are equal.

$f(n) = 2^n \qquad g(n) = 2^{2n}$

$\log 2^n \qquad \log 2^{2n}$

$n \log_2 2 \qquad 2n \log_2 2$

$n \qquad 2n$

Asymptotically, both are equal.

$$g_1(n) = \begin{cases} n^3 & n < 100 \\ n^2 & n \geq 100 \end{cases}$$

$$g_2(n) = \begin{cases} n^2 & n < 10{,}000 \\ n^3 & n \geq 10{,}000 \end{cases}$$

$n \rightarrow$ 100 $\quad$ 10,000

$g_1(n) > g_2(n)$ $\quad$ n^2 $\quad$ n^2 $\quad$ $g_2(n) > g_1(n)$

$g_1 = g_2$

TRUE OR FALSE :-

(i) $(n+k)^m = \Theta(n^m)$ $\quad$ TRUE

Relate it to $(n+k)^m$

$(n+3)^2$

↳ highest term here is n^2. so $\Theta(n^2)$.

Dated:

(2.) $2^{n+1} = O(2^n)$ TRUE

2×2^n

(3.) $2^{2n} = O(2^n)$ FALSE

$4^n > 2^n$

Should be $O(4^n)$

(4.) $\sqrt{\log n} = O(\log \log n)$ FALSE

$\sqrt{\log n} > \log \log n$ so false.

(5.) $n^{\log n} = O(2^n)$ TRUE.

$\log n^{\log n}$

$\log n \times \log n$

$2^n > \log n \log n$ so true.

Best, Worst & Average Case Analysis :-

(1.) Linear Search

	0	1	2	3	4	5	6	7	8	9
A	8	6	12	5	9	7	4	3	16	18

Key = 7

Key = 20

Best Case – searching key element present at first index.

Best Case Time – 1 $O(1)$

$B(n) = O(1)$

Worst case :- searching a key at last index.

worst case time = n

$W(n) = O(n)$

Average case – all possible case time / no. of cases

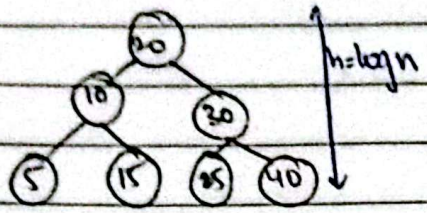
$$\text{Avg. time} = \frac{1+2+3+\dots+n}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2}$$

Asymptotic Notations :-

$B(n) = 1$	$W(n) = n$
$B(n) = O(1)$	$W(n) = O(n)$
$B(n) = \Omega(1)$	$W(n) = \Omega(n)$
$B(n) = \Theta(1)$	$W(n) = \Theta(n)$

Dated:

Binary Search Tree :-



$h = \log n$

Best case — search root element

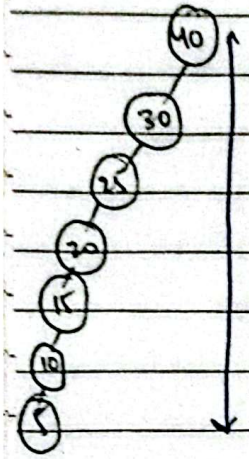
Best case Time $B(n) = 1$

Worst case — search for leaf element

Worst case Time $W(n) = \log n$

$\min W(n) = \log n$

$\max W(n) = n$

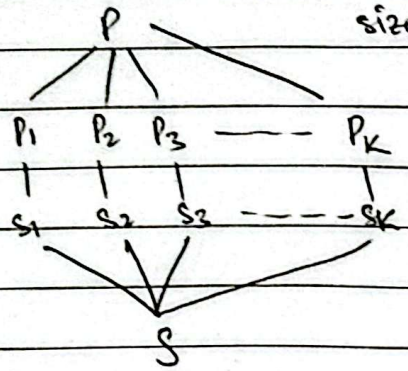


Left skewed BST

$h = n$ — no. of nodes

height

Divide And Conquer



P : problem

$P_1, P_2, P_3 \_ P_K$ → dividing problem into subproblems.

$S_1, S_2, \_ S_K$ → solutions

S : solution

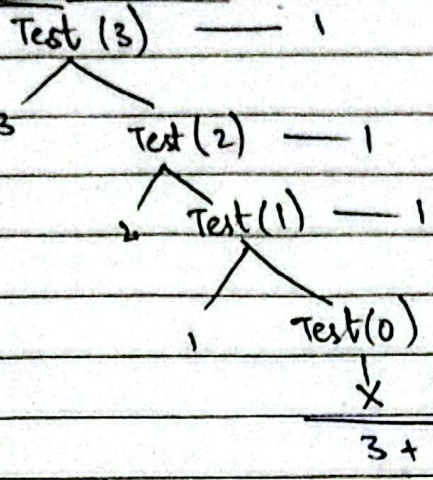
```
DAC (P)
{ if (small (P))
  { S(P);
  } else
    { divide P into P1, P2, P3, —— PK.
      Apply DAC (P1), DAC (P2), ——.
      Combine ( DAC (P1), DAC (P2), ——.)
    }
}
```

Divide & Conquer.

1. Binary Search
2. Finding max. & min.
3. Mergesort
4. Quicksort
5. Strassen's Matrix Multiplication.

Dated:

Recurrence Relation :-



3 + 1 calls for Test(3).

```
void Test (int n)
{ if (n>0)
  { printf("%d",n);
    Test (n-1);
  }
}
```

T(n-1) → 1 → T(n) = T(n-1) + 1

n+1 calls for n.

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

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

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

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

$[T(n-2-1) + 1] + 2$

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

$T(n) = T(n-4) + 4$

⋮ continue for k times.

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

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

Assume $n-k = 0$, $\therefore n = k$

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

$T(n) = T(0) + n$

$T(n) = 1 + n$ $\quad \theta(n)$.

Iterative Substitution :-

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

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

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

$= 2^3 T\left(\frac{n}{2^3}\right) + 3bn$

Dated:

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

$T(n) = b$ → base case

$2^i = n$

$\log 2^i = \log n$

$i \log 2 = \log n$

$\boxed{i = \log n}$

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

$T(n) = T(n) + bn \log n$

$T(n) = b \therefore \boxed{T(n) = b + bn \log n}$

Solving Recurrences by Substitution : Guess-and-Test

Ex:- $T(n) = 2T(n/2) + n$

Guess #1 $T(n) = O(n)$

Need $T(n) <= cn$ for some constant $c > 0$.

Assume $T(n/2) <= cn/2$ Inductive hypothesis

$T(n) <= 2cn/2 + n = (c+1)n$

Our guess was wrong.

Guess #2:- $T(n) = O(n^2)$ → $\frac{n}{2}$

Need :- $T(n) <= cn^2$ $\left(\frac{n}{2}\right)^2$ for some constant $c > 0$.

Assume :- $T(n/2) <= cn^2/4$ Inductive hypothesis.

Thus :- $T(n) <= 2cn^2/4 + n = cn^2/2 + n$

works for all n as long as $c >= 2$!! But there is alot of "slack".

Guess #3 :- $T(n) = O(n \log n)$

Need $T(n) <= cn \log n$

Assume $T(n/2) <= c(n/2)(\log(n/2))$

$T(n) <= 2c(n/2)(\log(n/2)) + n$

$<= cn \log n - cn + n <= cn \log n$

works for all n as long as $c >= 1$!! This is the correct guess.

Dated:

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

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

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

⋮

$2^k T(n-k)$, Typically base case = 1 , $n-k=1$, $k=n-1$.

Let's $k=n-1$

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

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

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

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

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

$= 2^2\left(2T\left(\frac{n}{2^3}\right) + n\right) + 2n$

$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$

⋮

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

Typically, base case = 1 , $\frac{n}{2^k} = 1$, $n = 2^k$, $\log n + k \log_2 2$

$k = \log_2 n$

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

$\Rightarrow n T\left(\frac{n}{n}\right) + n \log_2 n$

$\Rightarrow n T(1) + n \log_2 n \Rightarrow cn + n \log_2 n = \Theta(n \log n)$

MID-1 FALL 2020

Q2) $T(n) = 3T(n/3) + n^3$, $T(1) = 1$

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

$= 3^2 T\left(\frac{n}{3^2}\right) + \frac{n^3}{9} + n^3$

$= 3^2\left[3\left(T\left(\frac{n}{3^3}\right) + \left(\frac{n}{3^2}\right)^3\right)\right] + \frac{n^3}{9} + n^3$

Dated:

$= 3^3 T\left(\frac{n}{3^3}\right) + 3^2 \cdot \frac{n^3}{(9)^{3-2}} + \frac{n^3}{9} + n^3$

$= 27\, T\left(\frac{n}{27}\right) + \frac{n^3}{9^2} + \frac{n^3}{9} + n^3$

we can see a pattern which can be written as:-

$T(n) = 3^k T\left(\frac{n}{3^k}\right) + \left(\sum_{i=0}^{k-1} \frac{1}{9^i}\right) n^3$

$\downarrow$

$\left(\frac{1}{9^2} + \frac{1}{9} + \frac{1}{9^0}\right) \cdot n^3$

$T(1) = 1$, base case so $\frac{n}{3^k} = 1$, $n = 3^k$, $\log_3 n = k \log_3 3$ (1)

$k = \log_3 n$

$T(n) = 3^k T\left(\frac{n}{3^k}\right) + \left(\sum_{i=0}^{k-1} \frac{1}{9^i}\right) n^3$

$T(n) = 3^{\log_3 n} T\left(\frac{n}{3^{\log_3 n}}\right) + \frac{9}{8} n^3 \left(1 - \frac{1}{9^k}\right)$

$T(n) = 3^{\log_3 n} T\left(\frac{n}{3^{\log_3 n}}\right) + \frac{9}{8} n^3 \left(1 - \frac{1}{9^{\log_3 n}}\right)$

$3^k = 3^{\log_3 n} = n$

$T(n) = n\, T\left(\frac{n}{n}\right) + \frac{9}{8} n^3 \left(1 - \frac{1}{n^2}\right)$

$= n\, T(1) + \frac{9}{8}\left(n^3 - \frac{n^3}{n^2}\right)$

$= n + \frac{9}{8}(n^3 - n)$

$= n + \frac{9}{8} n^3 - \frac{9}{8} n$

- $\Theta(n^3)$ is the time complexity.

Geometric series formula:-

$\sum_{i=0}^{k-1} r^i = \frac{1 - r^k}{1 - r} \quad (r \neq 1)$

$\log_3 n \quad n^3 \cdot \frac{1 - (1/9)^k}{1 - 1/9} = n^3 \cdot \frac{1 - 9^{-k}}{8/9}$

$= \frac{9 n^3 (1 - 9^{-k})}{8}$

$= \frac{9}{8} n^3 (1 - 3^{-2k})$

Since $n = 3^k$ so $1 - 3^{-2k}$, $1 - n^{-2}$, $1 - \frac{1}{n^2}$

$= \frac{9}{8} n^3 \left(1 - \frac{1}{n^2}\right)$

$= \frac{9}{8} n^3 - \frac{9}{8} n$

$n^2 = 3^{2k} = 9^k$

$= \frac{9}{8} n^3 \left(1 - \frac{1}{9^k}\right)$

Dated:

Q3) b) $T(n) = 4T\left(\frac{n}{4}\right) + n^2$, Assume $T(1) = 1$.

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

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

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

$$= 4^3 T\left(\frac{n}{4^3}\right) + \underbrace{\frac{4^2 n^2}{256 = 16^2} + \frac{n^2}{4}}_{\frac{n^2}{16^2} + \frac{n^2}{16} + n^2} + n^2$$

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

$$\vdots$$

$$\hookrightarrow \frac{n^2}{4^2} + \frac{n^2}{4} + n^2 = n^2\left(\frac{1}{4^2} + \frac{1}{4} + 1 + \cdots\right)$$

$$= 4^k T\left(\frac{n}{4^k}\right) + n^2\left(1 + \frac{1}{4} + \frac{1}{16}\right) \rightarrow \frac{1}{1-r}$$

Using Geometric Series :-

$$T(n) = 4^k T\left(\frac{n}{4^k}\right) + n^2\left[\frac{1}{1-\frac{1}{4}}\right]$$

$$= 4^k T\left(\frac{n}{4^k}\right) + \frac{4}{3}n^2$$

since $T(1) = 1 \Rightarrow \frac{n}{4^k} = 1 \Rightarrow k = \log_4 n$

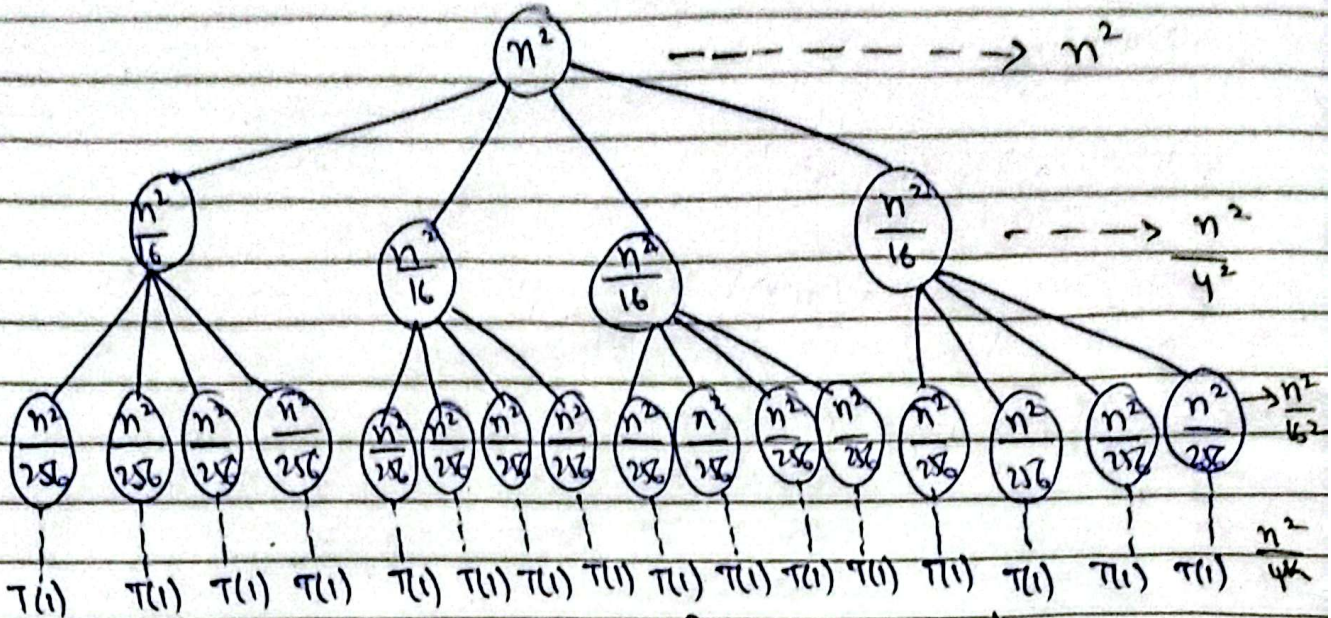
$$T(n) = \underbrace{4^{\log_4 n}}_{4^k = n} T(1) + \frac{4}{3}n^2$$

$$T(n) = n + \frac{4}{3}n^2$$

Hence, $\Theta(n^2)$ is the time complexity.

Dated:

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



FALL2020 Q4 Substitution Guess & Test Method.

$T(n) = 3T(n/2) + n^2$

Guess 1 :- $T(n) = O(n)$

$T(n) \leq cn$

$T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)$

$T(n) \leq 3c\frac{n}{2} + n^2 \leq 1.5cn + cn^2$ ×

Our guess was wrong, there is no $c > 1$ exist to prove $1.5cn + n^2 = O(n)$.

Guess 2 :- $T(n) = O(n^2)$

$T(n) \leq cn^2$

$T(n/2) \leq c(n/2)^2 \leq cn^2/4$

$T(n) \leq 3c\frac{n^2}{4} + n^2 = 0.75cn^2 + n^2 \leq cn^2$ for $c \geq 4$

So our guess is correct for $c \geq 4$.

$0.75cn^2 + n^2 \leq cn^2$ (find out c)

$n^2 \leq cn^2 - 0.75cn^2$

$n^2 \leq 0.25cn^2$

$\boxed{c \geq 4}$

MASTER'S THEOREM :-

Masters theorem cannot be used if :- o- $T(n)$ is not monotone

e.x :- $T(n) = \sin x$, $f(n)$ is not polynomial, e.x :- $T(n) = 2T\left(\frac{n}{2}\right) + 2^n$; b cannot be expressed as constants ex :- $T(n) = T(\sqrt{n})$

Dated:

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

$T(1) = c$

where $a \geq 1, b \geq 2, c > 0$. If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Ex1:- $T(n) = 1T(n/2) + \frac{1}{2}n^2 + n$

$a = 1, b = 2, \quad f(n) = \frac{1}{2}n^2 + n$

$d = 2 \qquad f(n) \in \Theta(n^d)$

so d is highest power of n in f(n).

$\therefore a < b^d$

$\Rightarrow \boxed{\Theta(n^2)}$

Ex2:- $T(n) = 2T(n/4) + \sqrt{n} + 42$.

$a = 2, b = 4, d = 1/2 \qquad f(n) \to n^{(1/2)} \to d = 1/2$

$2 = 4^{1/2}$

$2 = 2$

so $\Theta(n^{1/2} \log n) \Rightarrow \boxed{\Theta(\sqrt{n} \log n)}$

Ex3:- $T(n) = 3T(n/2) + \frac{3}{4}n + 1$

$a = 3, b = 2, \quad f(n) = \frac{3}{4}n + 1 \quad \hookrightarrow d = 1$

$3 > 2^1$

$\Rightarrow \boxed{\Theta(n^{\log_2 3})} \Rightarrow \boxed{\Theta(n^{\log_2 3})}$

FOURTH CONDITION :-

•- If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$ then

$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$

Ex:- $T(n) = 2T(n/2) + n \log n$

$a = 2, b = 2$, $f(n)$ is not a polynomial, however,

$f(n) \in n \log n$

$T(n) \in \Theta(n^{\log_2 2} \log^{1+1} n) \Rightarrow \boxed{\Theta(n \log^2 n)}$

Dated:

Master's Theorem Past Paper Qs :-

a) $T(n) = 6T\left(\frac{n}{\sqrt{n}}\right) + n + 30$

↳ b cannot be expressed as constant so Master's theorem cannot be used.

b) $T(n) = 9T\left(\frac{n}{3}\right) + \underbrace{3n^2 + 2^3 n}_{f(n)}$

$a = 9, \quad b = 3, \quad d = 2$

$9 = 3^2 \qquad \Theta(n^d \log n)$

$\Rightarrow \boxed{\Theta(n^2 \log n)}$

c) $T(n) = 7T\left(\frac{n}{4}\right) + \underbrace{n^{\log_4 7} \log^{k=1} n}_{k=1}, \qquad f(n) \in \Theta(n^{\log_b a} \log^k n)$

$\hookrightarrow T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$

$T(n) \Rightarrow \boxed{\Theta(n^{\log_4 7} \log^2 n)}$

Substitution Guess and Test Method :-

$\therefore T(n) = 8T\left(\frac{n}{2}\right) + n^2$

Guess 1: $T(n) = O(n^2 \log n)$

$T(n) \leq c \cdot n^2 \log n$ for some constant $c > 0$.

$T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^2 \log\left(\frac{n}{2}\right) = \frac{cn^2}{4} \log\left(\frac{n}{2}\right)$

$T(n) = 8T\left(\frac{n}{2}\right) + n^2$

$= 8 \frac{cn^2}{4} \log\left(\frac{n}{2}\right) + n^2 = 2 \cdot cn^2 \log\left(\frac{n}{2}\right) + n^2$

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

$= 2cn^2 (\log n - \log 2) + n^2 \qquad \log\left(\frac{a}{b}\right) = \log a - \log b$

$= 2cn^2 \log n - 2cn^2 \log 2 + n^2$

$\vdots \quad 2cn^2 \log n - 2cn^2 + n^2 = 2cn^2 (\log n - 1) + n^2$

$\log n - 1 \approx \log n$

$2cn^2 \log n$ is not $\leq cn^2 \log n$. so Guess was wrong.

Guess 2 :- $T(n) = O(n^3 \log n)$.

$T(n) \leq cn^3 \log n$

Dated:

$T\left(\frac{n}{2}\right) = c\left(\frac{n}{2}\right)^3 \log\left(\frac{n}{2}\right) = c\cdot\frac{n^3}{8}\log\left(\frac{n}{2}\right)$

$T(n) = 8T\left(\frac{n}{2}\right) + n^2$

$= 8c\frac{n^3}{8}\log\left(\frac{n}{2}\right) + n^2$

$= cn^3\log\left(\frac{n}{2}\right) + n^2$

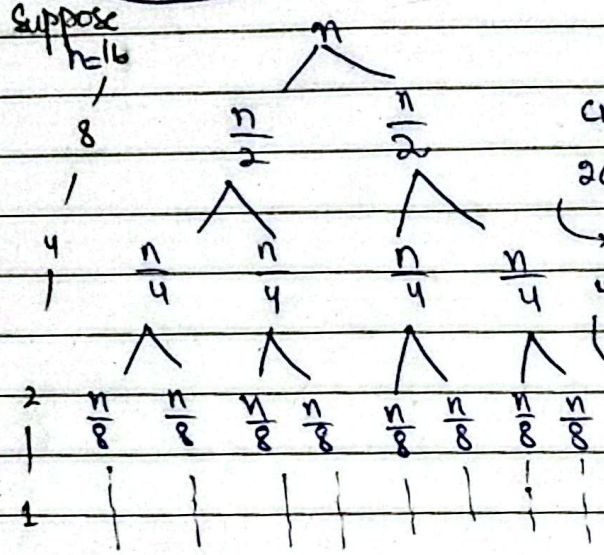
$= cn^3(\log n - \log 2) + n^2 = cn^3\log n - cn^3 + n^2$

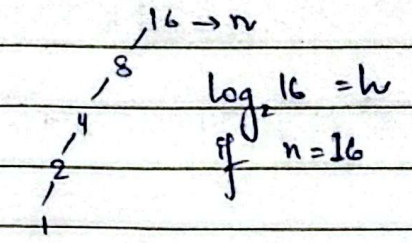
Proved | Guess correct

$cn^3\log n = O(n^3\log n)$

Recurrence Tree.

Ex: $T(n) = 2T(n/2) + cn$





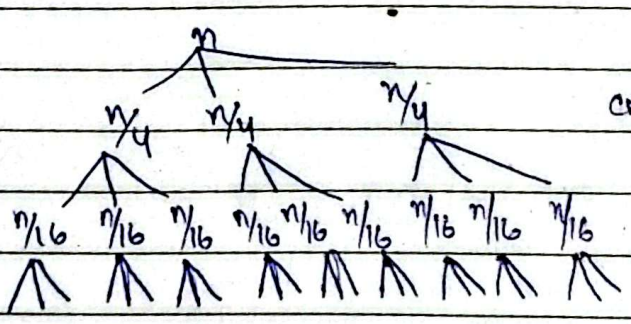
3cn → depends on height of the tree. $\frac{n}{2^i} = 1$, $n = 2^i$, $\log n = i$

$h = cn\log n$

$O(n\log_2 n)$

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

This 3 tells that n will divide into n/4 (3 times).



cost :-

$3c\left(\frac{n}{4}\right)^2 = \frac{3cn^2}{16}$

$9\cdot\left(\frac{n}{64}\right)^2 = \left(\frac{3}{16}n\right)^2$

$\left(\frac{3n}{4}\right)^3$

Dated:

G.P series $cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \left(\frac{3}{16}\right)^3 cn^2 + \cdots$

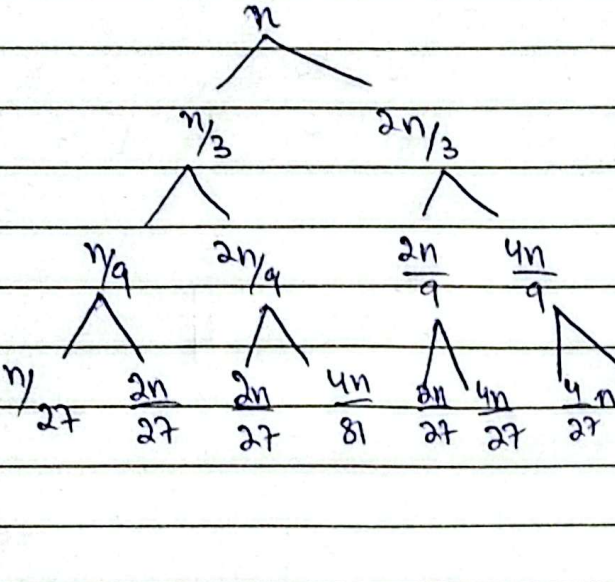
$= cn^2 \left[1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \left(\frac{3}{16}\right)^3 + \cdots + \cdots\right]$

$= cn^2 \left[1 + r + r^2 + r^3 + \cdots\right]$

$\frac{1}{1-r}, \quad r < 1$

$= cn^2 \left[\frac{1}{1 - \frac{3}{16}}\right] = cn^2 \frac{1}{13/16} = \boxed{\frac{cn^2\,16}{13}}$

Q. $T(n) = T(n/3) + T(2n/3) + cn.$



$c(n/3) + c(2n/3) = \frac{cn}{3} + \frac{2cn}{3} = \frac{3cn}{3}$

$= cn$

$c(n/9) + c(2n/9) + c(2n/9) + c(4n/9)$

$= cn/9 + 2cn/9 + 2cn/9 + 4cn/9$

$= 9cn/9 = cn$

$cn + cn + cn$ ——— ↑ depends on height of tree.

$\frac{n}{3^i} = 1$

$n = 3^i$

$\log_3 n = \log_3 3^i$

$i = \log_3 n$

$\left(\frac{2}{3}\right)^i n = 1$

$\frac{2^i}{3^i} n = 1 \quad \Rightarrow n = \frac{3^i}{2^i} \quad \Rightarrow n = \left(\frac{3}{2}\right)^i$

$\log_{3/2} n = i \log_{3/2} \frac{3}{2}$

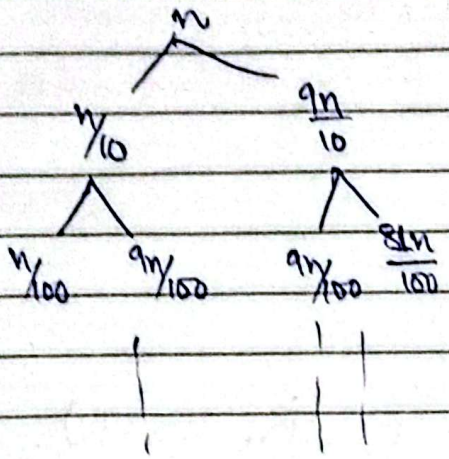
$i = \log_{3/2} n$

Dated:

since $n/3 < 2n/3$
↳ this will take more time to reach base case.
so, that's why we used $\log_{3/2} n$.

- $O(n \log_{3/2} n)$

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



Cost

cn

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

$= cn$

$n/10^i = 1$, $(9/10)^i n = 1$

$n = 10^i$, $n = (10/9)^i$

$\log_{10} n = i$, $\log_{10/9} n = i \log_{10/9} 10/9$

$i = \log_{10/9} n$

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

$n/10 = 0.1$

$9n/10 = 0.9$

↳ will take more time to reach base case.

Dated:

LOOP INVARIANT :-

Precondition :- true before the method execution.
Post condition :- true after method execution.

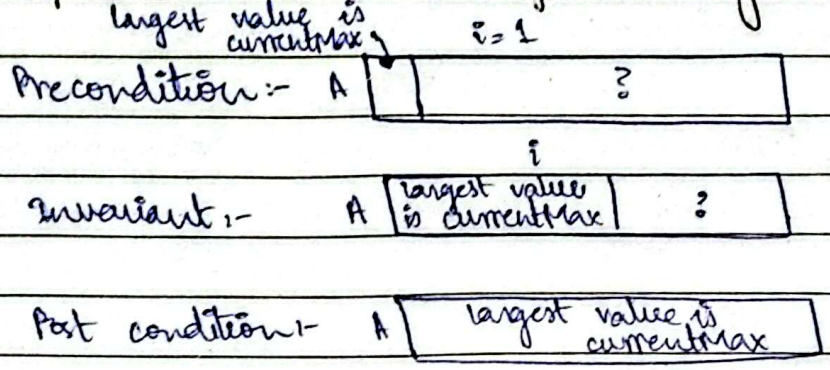
FINDING MAXIMUM ELEMENT :-

- precondition: A is non empty.
- postcondition: max value of A is returned.

```
int findmax (int [] A){
    int currentMax = A[0];
    // invariant : currentMax is the max of A[0..i]
    for (int i=1 ; i< A.length() ; i++){
        if (A[i] > max){
            max = A[i];
        }
    }
    return max;
}
```

A[0..1] A[0..i] A[0.. a.length)

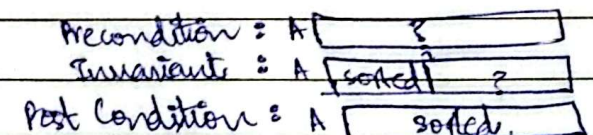
- A loop invariant is true before, during, & after the loop.



Four Concerns of Loop Invariants :-

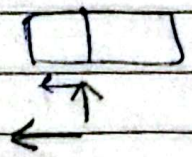
1. Initialization :- Make the invariant true at the start.
2. Termination :- Make the loop end when the post condition is true).
3. Progress :- Make progress towards the post condition.
4. Maintenance :- Make the invariant true after each iteration.

INSERTION SORT :-



Dated:

```
insertionSort(A):
for i = 1; i < A.length; i++;
    j = i;
    while j > 0 and A[j] < A[j-1]
        swap(A[j], A[j-1]);
        j--;
```



Working & Dry-Run :-

Example :- A = [5, 3, 4, 1, 2]

i = 1, j = 1, A[j] = 3, A[j-1] = 5

A = [5, 3, 4, 1, 2]

swap

After swapping :- A = [3, 5, 4, 1, 2]

i = 2, j = 2, A[j] = 4, A[j-1] = 5

A = [3, 4, 5, 1, 2]

compare, no swap

i = 3, j = 3, A[j] = 1

A = [3, 4, 5, 1, 2]

A = [3, 4, 1, 5, 2]

A = [3, 4, 5, 2]

A = [1, 3, 4, 5, 2]

i = 4, j = 4, A[j] = 2

A = [1, 3, 4, 5, 2], A = [1, 3, 4, 2, 5], A = [1, 3, 2, 4, 5]

A = [1, 2, 3, 4, 5]

j < 0, while loop terminated.

i = i++ A.length for loop terminates. ∴ Array is sorted.

Dated:

INSERTION SORT COMPLEXITY:-

- WORST CASE :- When array is in reverse order. Every new element has to be compared with all previous elements.

$1+2+3+\dots+(n-1) = \boxed{O(n^2)}$

- BEST CASE :- When the array is already sorted. Each new element is compared once, sees that its already in place, & stops.

$= \boxed{O(n)}$

WHY IS THE BEST-CASE RUNTIME INTERESTING?

- Because :- Unlike many other quadratic algorithms (like selection sort, or bubble sort), insertion sort actually runs linearly on sorted input.
- This means insertion sort is adaptive → it takes advantage of existing order in the data.
- That makes it efficient for nearly sorted arrays & real-world cases where data is often close to sorted.

Example :- A = [1, 2, 3] already sorted

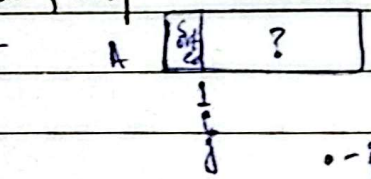
so it skips inner loop swapping.

```
while j>0 & A[j] < A[j-1]
{ swap(A[j],A[j-1]); this becomes false.
  j--;
}
```

∴ j only decrements when j & j-1 are out of order.

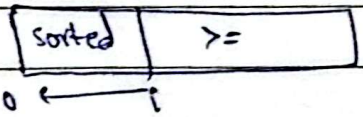
Q) Precondition, loop invariant, Postcondition of inner loop of Insertion sort?

Precondition :-



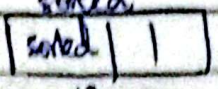
- The subarray A[0 ... i-1] is sorted (that is guaranteed by outer loop).
- j=i, meaning we're trying to insert A[i] in the correct position wrt to already sorted part.

Loop invariant :-



Within A[0...i], the element A[0..j-1] are sorted, & the elements A[j..i] are >= to everything in A[0..j-1].

Dated:

Postcondition:- 

after inner loop finishes,
- the element that started at A[i] has been inserted into its correct position among A[0...i].
- As a results, the subarray A[0..i] is sorted.

SELECTION SORT :-

```
selectionSort (A):
    for i = 0 to A.length - 1:
        minIndex = i
        for j = i+1 to A.length-1:
            if A[j] < A[minIndex]:
                minIndex = j;
        swap (A[i], A[minIndex]);
```

Initialization :- Before the first iteration (i=0):
A[0... -1] is an empty subarray, which is trivially sorted.

Maintenance :-
- Assume before iteration i, the invariant holds A[0...i-1] is sorted & contains the i smallest elements.
- During iteration i:
  - The inner loop finds the smallest element in A[i...n-1].
  - After swapping it into position i, the subarray A[0..i] becomes sorted & contains the i+1 smallest elements.

Progress :-
- On each iteration i, i increases by 1.
- This means the sorted subarray A[0...i-1] grows larger by one element at each step.

Dated:

Termination :-

- The outer loop ends when $i = n$ (after n iterations).
- At this point, the invariant tells us that $A[0..n-1]$ is sorted & contains all n elements of the array.

Working & Dry-Run :-

A = [3, 1, 2]

i = 0 , j = i+1 → n-1 minIndex = 1

A[1..2] look for smaller element than A[0]

minIndex = 1

swap A[0] & A[1] , A = [1, 3, 2]

A = [1, 3, 2]

j++ → n-1

minIndex = 2

swap A[1] & A[2]

A = [1, 2, 3]

i = n-1

Time Complexity Analysis Of Selection Sort :-

- Best case :- $O(n^2)$
- Worst case :- $O(n^2)$
- Average case :- $O(n^2)$

- Not efficient for large n.

Maximum Sub-Array Problem :-

Problem :- Find a contiguous subarray with the largest sum.

Step 1 - Brute Force Approach :- $O(n^2)$.

- Try all possible subarrays.
- for each subarray $A[i..j]$, compute its sum.
- Keep track of the maximum sum seen so far.

Pseudocode:- **Dated:**

```
maxSum = -Infinity ——→ 1
for i = 0 to n-1 :  ——— n+1 or n
{
    for j = i to n-1 : ——— n * n
    {
        sum = 0 ;  ——— n * n
        for k = i to j :  —— n * n * n
        {
            sum += A[k] ;  —— n * n * n
        }
        if sum > maxSum :  —— n * n
        {
            maxSum = sum  —— n * n
        }
    }
}
```

Total Time Complexity:- $O(n^3)$

Outer loop runs n times
Inner loop runs n-i times for each i.
Innermost loop runs:- (j-i+1) times for each (i,j).

Optimized $O(n^2)$ Approach:-

```
int maxSum = INT_MIN;
for (int i = 0 ; i < n ; i++) {  — n
    int sum = 0 ;  — n
    for (int j = i ; j < n ; j++) {  — n-i
        sum += A[j] ;  — n * n-i
        if (sum > maxSum) {  —
            maxSum = sum ;  — n * n-i
        }
    }
}
```

$\approx O(n^2)$

DIVIDE N CONQUER APPROACH:-

- Divide the array into two halves.
  - Left half → A [left .. mid]
  - Right half → A [mid+1 .. right].

Dated:

- Recursively find the maximum subarray sum in :-
  - The left half.
  - The right half.
- Combine :- The maximum subarray might cross the middle, so we also calculate the maximum sum that crosses mid.

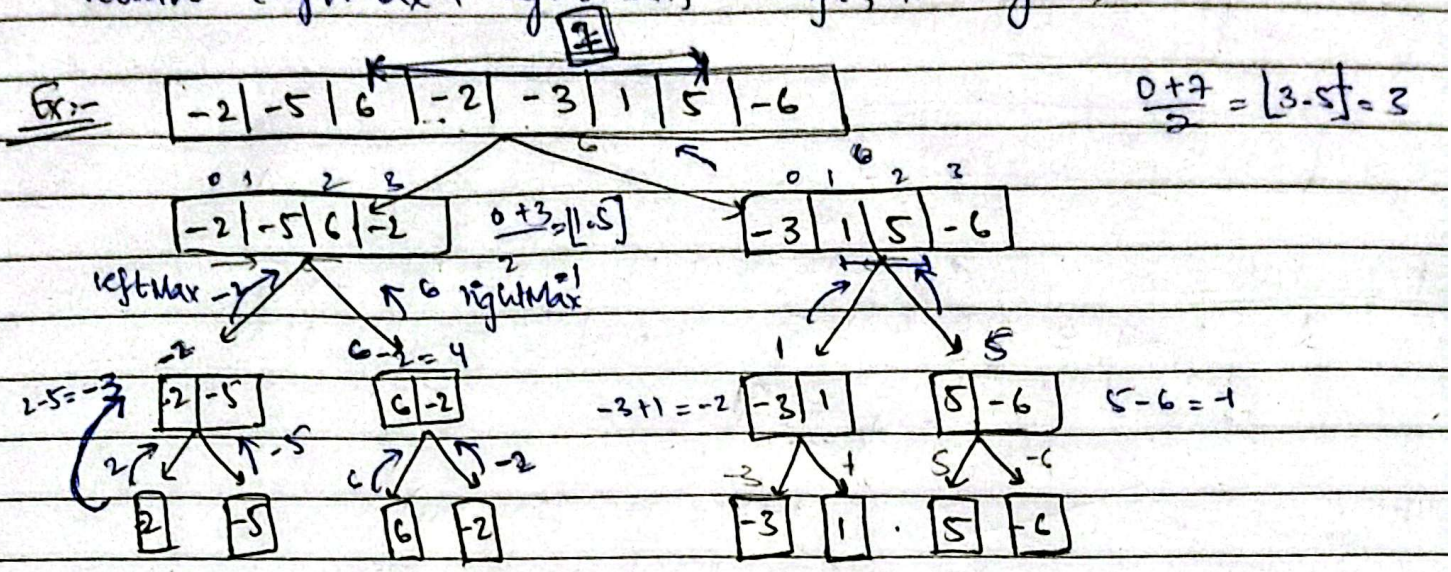
Pseudocode :-

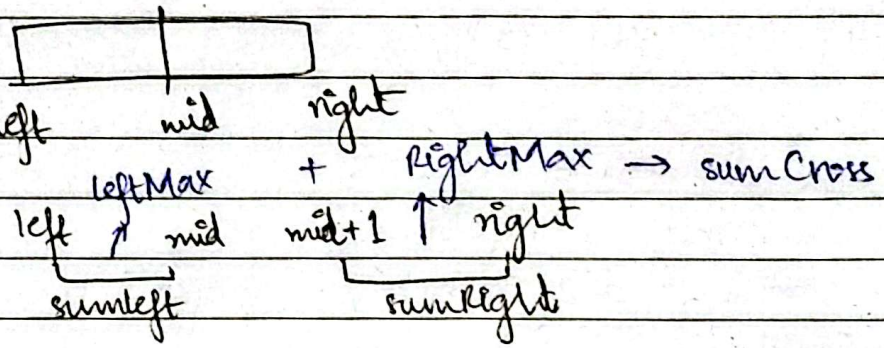
```
FindMaxSubArray (A, left, right):
    if (left == right)  // single element left = base case
        return (A[left], left, right)
    mid = floor((left+right)/2)

    (sumCross, startCross, endCross) = FindMaxCrossingSubarray (A, left,
                                                    mid, right)
    (sumleft, startleft, endleft) = FindMaxSubarray(A, left, mid)

    (sumright, startright, endright) = FindMaxSubarray(A, mid+1, right)

    return the maximum of (sumleft, sumRight, sumCross)

FindMaxCrossingSubarray (A, left, mid, right):
    sum = 0
    leftMax = -infinity
    for i = mid to left:
        sum += A[i]
        if sum > leftMax:
            leftMax = sum
            maxleft = i

    sum = 0
    rightMax = -infinity
    for j = mid+1 to right:
        sum += A[j]
```

Dated:

```
if sum > rightMax :
    rightMax = sum
    maxRight = j

return (leftMax + rightMax, maxleft, maxRight)
```

Ex:-





Time Complexity :-

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

n —split $n/2$ parts→ cn

$n = 2^i$

$n/2$ $n/2$ ——→ $c(n/2) + c(n/2) = 2cn/2$

$n/4$ $n/4$ $n/4$ $n/4$ — cost for this $2c(n/4) = 4cn/4 = cn$

$n/8$ $n/8$ $n/8$ $n/8$ $n/8$ $n/8$ $n/8$ $n/8$ cost for this

$h \to n \dots 1$ $\therefore O(n)$

Dated:

$\frac{n}{2^i} = 1$

$n = 2^i$

$\log_2 n = i \log_2 2$

$i = \log_2 n$

$T(n) = O(n \log_2 n)$

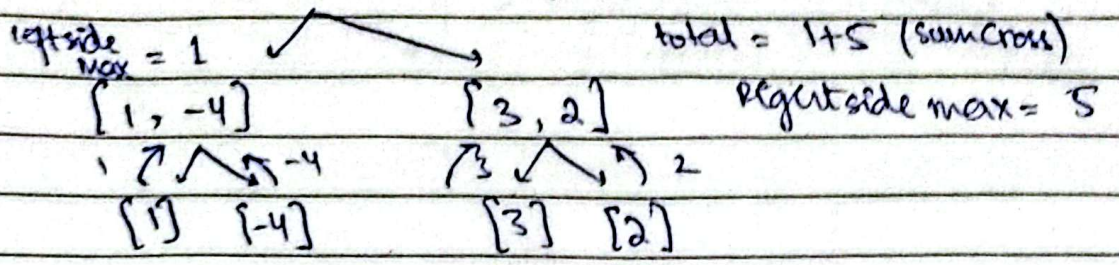
Dry Run :-

$A = [1, -4, 3, 2]$ total = 5

left = 0 right = 3

return max of (1, 5, 4) → 5

mid = $\frac{0+3}{2} = [1.5] = 1$



Binary Search Problem :-

- Given sorted array, target
- Input :- arr[] = {3, 4, 6, 7}, target = 4

linear search :- O(n).

```
int binarySearch (int arr[], int left, int right, int target){
    if while
    if (left > right)
        return -1;
    int mid = left + (right - left)/2;

    if (arr[mid] == target)
        return mid;
```

Dated:

```
else if (arr[mid] < target)
    return binarySearch(arr, (mid+1), right, target);
```

left → mid+1 index. now becomes left index.

```
else
    return binarySearch(arr, left, (mid), target);
}
```

right → mid index. now becomes right index.

Ex:-

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

left → 0, right → 8, arr[mid] = 7

$\left\lfloor \frac{0+8}{2} \right\rfloor = 4$

Case 1: target = 7

arr[mid] == target

simply return found @ index mid (4).

Case 2: target = 10

arr[mid] = 7

7 < 10

1	3	4	6	7	8	10	13	14
0	1	2	3	4	5	6	7	8

mid+1 → 5, right → 8

7 < 10

search in this

5	6	7	8
8	10	13	14

left → 5, right → 8

$\left\lfloor \frac{5+8}{2} \right\rfloor = 6.5 = 6$

mid = 6

arr[6] = 10 found @ index 6.

Case 3: target = 5

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

arr[mid] = 7

left → 0, right → 8

7 > 5 so search left → mid.

0	1	2	3	4
1	3	4	6	7

left → 0, right → 4, mid → 2

arr[mid] = 4 < 5 so search mid+1 → right

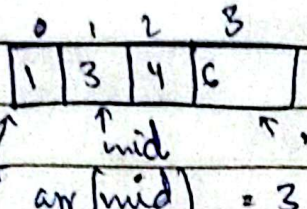
3	4
6	7

mid → 3, left → 3, right → 4

$\left\lfloor \frac{3+4}{2} \right\rfloor = 3$

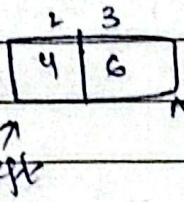
6 > 5 search in left → mid

Dated:



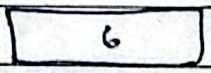
arr[mid] = 3 < 5

so search in mid + 1 to right



left = 2, right = 3

mid = 2, arr[2] = 4 < 5 search ~~left → mid~~ mid+1 → right

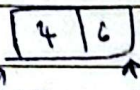


left = 3, right

mid = 3

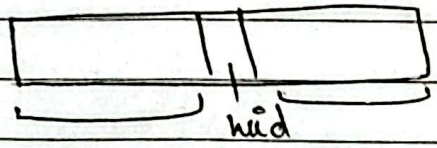
6 > 5 search in left half

arr[3...2]



right left

left > right return -1



left mid+1 to right

- when searching left → mid, we donot include mid element itself. left → mid - 1

Time Complexity :-

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

$n/2^i = 1$

$n = 2^i$

$\log_2 n = i \log_2 2$

$i = \log_2 n$

$\Rightarrow O(\log n)$

Loop Invariant for Binary Search :-

Dated:

Three Conditions :-

- Array arr is sorted in ascending order.
- left <= right
- target belongs to arr[left.....right]

Initialization :-

- Array is sorted due to precondition of the method.
- since arr.length is atleast 1 so left <= right.
- Precondition guarantees that target is in arr.

Maintenance :-

Case 1 :- If arr[mid] == target

Case 2 :- arr[mid] > target

left → mid-1 arr[left.... mid -1] or arr[left...mid]

so right = mid or right = mid-1

subarray arr[left... right] still contains target.

Case 3 :- If arr[mid] < target → arr[mid+1 ... right]

left = mid+1

- subarray arr[left ... right] still contains target.

After each iteration :-

- Array is still sorted (unchanged).
- New bounds still satisfy left <= right
- If target exists its still in range arr[left... right].

Termination :-

- arr[mid] == target so return mid index.
- If left > right so terminates if target doesnot exists in arr[left .... right].

DIVIDE N CONQUER : MERGE SORT

Iterative Merge sort :-

Compare A[i] & B[j]

A	B	C (final sorted array)
i → 2	5 ← j	2
i → 8	9 ← j	5
i → 15	12 ← j	8
		9
		12, 15

$\Theta(m+n)$

Dated:

```
Algorithm Merge (A, B, m, n)
{   i = 1, j = 1, k = 1
while (i <= m && j <= n)
    if (A[i] < B[j])
        C[k++] = A[i++];
    else
        C[k++] = B[j++];
}
for ( ; i <= m ; i++)
    C[k++] = A[i];

for ( ; j <= n ; j++)
    C[k++] = B[j];
}
```

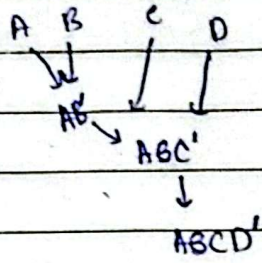
M-way Merging

A B C D

Merging 3 arrays
↳ 3-way merge

A B C D

Now if I want to merge 4 arrays using 2-way merge

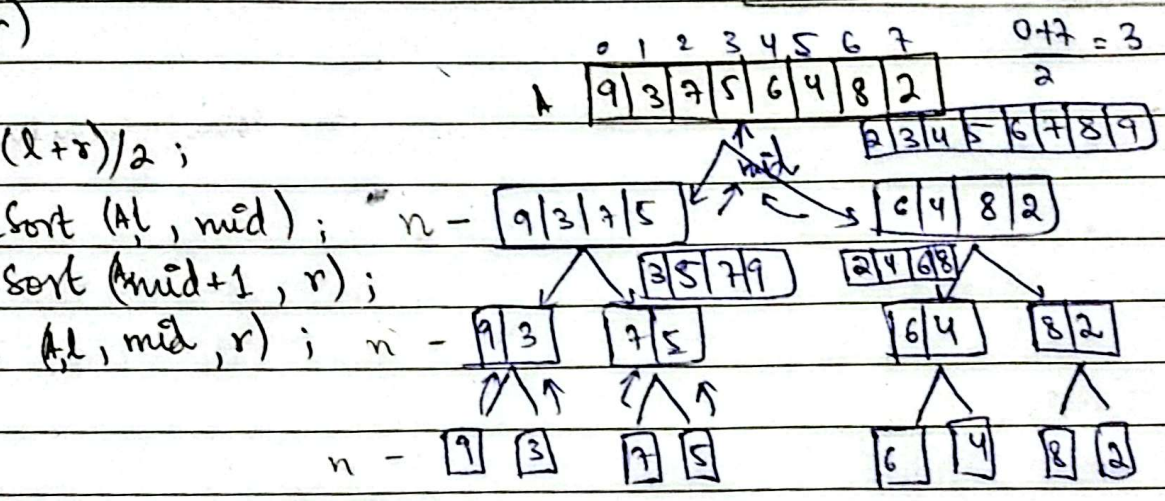


MERGE SORT ALGORITM: DIVIDE AND CONQUER :-

```
Alg. MergeSort (A, l, r)
{
    if ( l < r)
    {
1 — mid = (l+r)/2 ;
T(n/2) — MergeSort (A, l, mid) ;
T(n/2) — MergeSort (A, mid+1, r) ;
n  — Merge (A, l, mid, r) ;
    }
}
```

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

$n \log n$



$\frac{n}{2^i} = 1$ ↳ base case :- only one element left.

$i = \log n$

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

↳ Master theorem

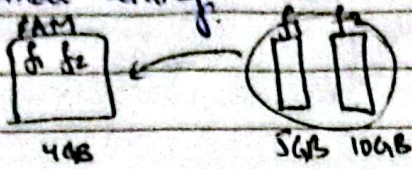
$a = 2, b = 2, d = 1 \quad 2 = 2^1 \Rightarrow \boxed{O(n \log n)}$

Pros n Cons of Merge Sort :- **Dated:**

Pros :- •- large size list (works effectively on that).

•- Linked list

•- External sorting



•- Stable

Cons :- •- Extra space (not inplace sort)

•- No small problem (for small sized lists it's slower).

•- we use insertion sort (better for small sized arrays) also stable.

Stable algorithms	→ Insertion sort
	→ Merge sort
	→ Bubble sort.

•- Recursive, Recursive call (Call stack) / $h = \log n$

Space $(n + \log n)$

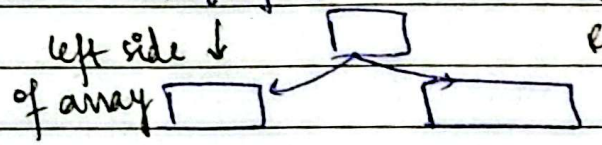
extra arr for merging — call stacks

~~Loop Invariant~~ :- left → mid sorted ↔ merge ↔ mid+1 → right sorted.

Initialization :- $i = j = 1$, $k = 1$ left subarray arr[l...m] right subarray arr[m+1 ...r]

temp[1, k-1] is empty. This empty subarray contains $k-1 = 0$ smallest elements of l and R since $i = 1 = j = 1 = k = 1$.

L[i], R[j] or arr[left] arr[mid+1]



Maintenance :-

•- In each iteration, we compare arr[i] (from left) and arr[j] (from right).

•- The smaller of two is appended to temp[k].

•- This guarantees that temp[0..k] contains the k+1 smallest elements of combined subarrays in sorted order.

Dated:

- do i++ until it reaches m (mid) i <= m

j++ → j <= r → right

Termination :-

- The loop ends when one subarray is fully scanned.
- At this point, temp[0..k-1] contains the smallest k elements in sorted order.
- The remaining elements from unfinished subarray are greater than or equal to the ones already copied so they can simply be appended.
- Finally arr[l...r] is overwritten with temp making it sorted.

Loop invariant for mergesort function

Recursive structure :-

- mergeSort(arr, l, m) sorts arr[l..m]
- mergeSort(arr, m+1, r) sorts arr[m+1..r]
- Then merge(arr, l, m, r) merges them.

At the beginning of each call to merge(arr, l, m, r), the subarrays arr[l..m]

Mergesort merge function code :-

```
void merge(int arr[], int l, int m, int r)
{ int i = l; int j = m+1; int k=0;
  int temp[r-l+1];
  while (i<=m && j<=r) {
    if (arr[i] <= arr[j])
      temp[k++] = arr[i++];
    else
      temp[k++] = arr[j++];
  }
  while (i<=m)
    temp[k++] = arr[i++];
  while (j<=r)
    temp[k++] = arr[j++];
```

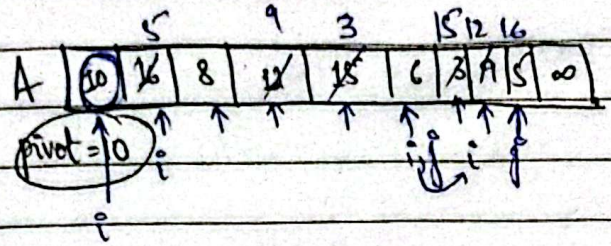
```
for(int p=0; p<k; p++)
{ arr[p] = temp[p];
}
}
```

Dated:

QUICKSORT :-

[6 3 5 4 2 1 (7)] ← this is sorted we're sure since usse pehle waale usse greater hain.

An element is in sorted position if elements right to it are greater & elements left to it are lesser.



- Start i from pivot and j from infinity.
- i will look for elements greater than pivot and j will search for elements smaller than pivot element.
- Then exchange its positions.
- increment i until larger element than pivot found and decrement j until smaller element than pivot found then swap.
- Swap pivot element & A[j].
- Now that pivot element is sorted.

```
Partition (l, h)
{
  pivot = A[l];
  i=l  j=h;
  do {
    i++;
  } while (A[i] ≤ pivot);
  do {
    j--;
  } while (A[j] > pivot);
  if (i<j)
    swap (A[i], A[j]);
  swap (A[low], A[j]);
  return j;
}
```

```
Quicksort (l,h)
{ if (l<h)
  { j = partition (l,h)
    Quicksort (l, j);
    Quicksort (j+1, h);
  }
}
```

Dated:

```
Algorithm Quicksort (arr, start, end):
    if start < end:
        pivotIndex = partition(arr, start, end);
        quicksort (arr, start, pivotIndex-1);
        quicksort (arr, pivotIndex+1, end);

Algorithm Partition (arr, start, end){
    mid = start + (end - start)/2;
    pivot = arr[mid]
    swap (arr[pivot], arr[start]);
    i = start + 1;
    j = end

    while True{
        while i <= j and arr[i] <= pivot {
            i++; }
        while i <= j and arr[i] > pivot{
            j--; }
        if i >= j {
            break}
        swap (arr[i], arr[j]);
    }
    swap (arr[start], arr[j]);
    return j;
}
```

Algorithms	Worst Case	Best Case
Insertion sort	$O(n^2)$	$O(n)$
Merge sort	$O(n \log n)$	$O(n \log n)$
Heap sort	$O(n \log n)$	$O(n \log n)$
Quick sort	$O(n^2)$	$O(n \log n)$
Bubble sort	$O(n^2)$	$O(n)$

Date:

Calculating Complexity of Algorithms :-

1- sum = A[0] ∴ - Assignment ∴ - fetch A[0]
2- for i = 1 to i = n-1 ∴ - fetch A[i]
3- sum = sum + A[i] ∴ - Add ∴ - Update
4- return sum

Step 2 - can also be written as :-

for (i = 1 ; i <= n - 1 ; i++) ∴ - Add ∴ - Update
(2a) (2b) (2c)

Statement	Operations	Iterations	Subtotal
1	2	1	2×1 = 2
2a	1	1	1×1 = 1
2b	1	n	1×n = n
2c	2	n-1	2(n-1) = 2n-2
3	3	n-1	3(n-1) = 3n-3
4	1	1	1×1 = 1

= 2+1+n+2n-2+3n-3+1
= 6n-1

1- sum = 0
2- for i = 1 to i = n ——— n+1
3- for j = 1 to j = n ——— n
4- sum++
5- return sum

Statement	Operations	Iterations	Sub-total
1	1	1	1×1 = 1
2a	1	1	1×1 = 1
2b	1	n+1	1×n+1 = n+1
2c	2	n	2n
3a	1	n	n
3b	1	n*(n+1)	n^2+n
3c	2	n*n	$2n^2$

Date:

4	2	n * n	$2n^2$
5	1	1	1
			$5n^2+5n+4$

```
int divide_sum (int n)
{
    int sum = 0
    while (n > 1)
    {
        sum += n
        n /= 2
    }
    return sum
}
```

Iteration	n=8
1	8
2	4
3	2
4	1

Pg No.

Date:

Mathematical formulas :-

- $1 + 2 + 2^2 + 2^3 + \dots + 2^k$

  Sum $= \sum_{i=0}^{k} \frac{r^k - 1}{r - 1}$

  where $r = \frac{T_2}{T_1}$

- $1 + \frac{3}{4}n + \frac{9}{16}n + \dots$

  $S_\infty = n\left(\sum_{i=0}^{\infty} \frac{1}{1-r}\right)$

  $r = \frac{T_2}{T_1}$

- $q + q^2 + q^3 + \dots + q^{k-1}$

- $\sum_{i=0}^{i=k-1} r^i = \frac{1 - r^k}{1 - r}$

Binary Search :-

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

Merge Sort :-

$$T(n) = 2T(n/2) + \cancel{O(n)}$$

Maximum Subarray Sum :-

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

$$\sum_{n=1}^{\infty} a_1 (r)^{n-1} = \frac{a_1}{1-r}$$

$$\sum_{i=0}^{k-1} a^i = \frac{1 - a^i}{1 - a}$$

O - notation

$O(g(n)) = \{ f(n) : \forall c > 0$, $\exists n_0 > 0$ such that for $\forall n \geq n_0$, we have $0 \leq f(n) \leq cg(n) \}$

$$\lim_{n \to \infty} \left[\frac{f(n)}{g(n)}\right] = 0$$

ω - notation

$\omega(g(n)) = \{ f(n) : \forall c > 0$ $\exists n_0 > 0$ such that $n \geq n_0$ we have $0 \leq cg(n) < f(n) \}$

$$\lim_{n \to \infty} \left[\frac{f(n)}{g(n)}\right] = \infty$$