ASSIGNMENT 1

Q1. Solve the recurrance relation using repeated substitution method.

a) $S(n) = \begin{cases} S(n-1)+n & \text{for } n>0 \\ 0 & \text{for } n=0 \end{cases}$

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

$= [S(n-2)+n-1]+n$

$= [S(n-3)+(n-2)]+(n-1)+n$

$= [S(n-i)+(n-(i-1)] + (n-(i-2))+i$

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

$= S(0)+(n-n+1)+(n-n+2)+\cdots+n$

$= 0+1+2+\cdots+n$

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

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

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

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

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

replace i by n

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

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

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

$= m(n-3)+1+1+1$

$= m(n-k)+k$

$M(0) = 0$

let $n - k = 0$

$\therefore n = k$

$m(n) = M(0) + k$

$= 0 + n$

$= n$

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

$T(1) = 1$

$T(n) = T(n/4) + 1 + 1$

$= T(n/8) + 1 + 1 + 1$

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

$T(1) = 1$

let $n/2^k = 1$

$n = 2^k$

$k = \log_2 n$

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

$= 1 + \log_2 n$

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

$T(1) = 2$

$T(n) = 2T(n/4) + 2(n/2) + n$

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

$= T(n/8) + 4 \times (n/4) + n + n$

$= 8T(n/8) + n + n + n$

$= 2^k T(n/2^k) + k \times n$

$T(1) = 2$

let $n/2^k = 1$

$n = 2^k \Rightarrow k = \log_2 n$

$T(n) = 2^k T(1) + kn$

$= n(2) + n(\log_2 n)$

$= 2n + n\log_2 n$

~~$= 2n + n\log_2 n$~~

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

$T(n) = 1 + t(n-2)$ $\quad t(n-2) = 1 + t(n-4)$

$= 1 + 1 + t(n-4)$ $\quad t(n-4) = 1 + t(n-6)$

$= 1 + 1 + 1 + t(n-6)$ $\quad t(n-6) = 1 + t(n-8)$

$= 1 + 1 + 1 + 1 + t(n-8)$

$= 4 + t(n-8)$

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

@

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

i) Replace $i = \frac{n}{2}$ $\quad = T\left(n - \frac{2n}{2}\right) + \frac{n}{2}$

$= T(0) + \frac{n}{2}$

$= \frac{n}{2}$

ii) Replace $i = \frac{n+1}{2}$ $= T\left(n - 2\left(\frac{n+1}{2}\right)\right) + \frac{n+1}{2}$

$= T(n - (n+1)) + \frac{n+1}{2}$

$= T(n - n + 1) + \frac{n+1}{2}$

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

$= 1 + \frac{n-1}{2}$

$\frac{2+n-1}{2} = \frac{n+1}{2}$

Q3) Prove

a) $f(n) = 5n^3 + 2n^2 - 5$

S.T. $F(n) = O(n^3)$

let $f(n) = 5n^3 + 2n^2 - 5$

$f(n) \leq C * g(n) \quad \forall \; n \geq n_0$ $\quad \forall$ = for all

$g(n) = n^3$

$5n^3 + 2n^2 - 5 \leq C * n^3$

$C = 1 + 5 = 6 \Rightarrow C = 6$

$5n^3 + 2n^2 - 5 \leq 6n^3$

let $n = 1$

$5 + 2 - 5 \leq 6$

$2 \leq 6$

Condition is true.

$\therefore 5n^3 + 2n^2 - 5 = O(n^3)$ for $C = 6$ and $n_0 \geq 1$

b) $6 * 2^n + n^2 = \Omega(2^n)$

let $f(n) = 6 * 2^n + n^2$

$f(n) \geq C * g(n)$ when $n \geq n_0$

$g(n) = 2^n$

$C = 6 - 1 = 5 \Rightarrow 6 = 5$

$6 \times 2^n + n^2 \geq 5 * 2^n$

let $n = 1$

$6 \times 2 + 1 \geq 5 \times 2$

$13 \geq 10$

Condition is true.

let $n = 0$

$6 \times 2^0 + 0 \geq 5 \times 2^0$

$6 \geq 5$

condition is true.

$\therefore 6 \times 2^n + n^2 = \Omega(2^n)$ for $C = 5$ and $n \geq 0$

c) $3n+3 = \Theta(n)$

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

let $f(n) = (3n+3)$
$\therefore g(n) = (n)$

$f(n) \geq c_1 * g(n)$
$3n+3 \geq c_1 * n$
$\therefore c_1 = 2$

$3n+3 \geq 2n$
for $n=1$ $6 \geq 2$
for $n=3$ $12 \geq 6$
for $n=2$ $9 \geq 4$
} condition is true

$f(n) \leq c_2 * g(n)$
$3n+3 \leq c_2 * n$
$\therefore c_2 = 4$
$3n+3 \leq 4 * n$
for $n=1$ $6 \leq 4$ $\Rightarrow$ condition is false.
for $n=3$ $12 \leq 12$ $\Rightarrow$ condition is true
for $n=2$ $9 \leq 8$ $\Rightarrow$ condition is false

$\therefore 2n \leq 3n+3 \leq 4n$ for $c_1 = 2$, $c_2 = 4$ and $n_0 = 3$

d) $5n^2 - 6n = \Theta(n^2)$

$f(n) = 5n^2 - 6n$

$C_1 * g(n) \leq f(n) \leq C_2 * g(n)$

$g(n) = n^2$

$f(n) \geq C_1 * g(n)$

$C_1 = 5 - 1 = 4$

$\therefore 5n^2 - 6n > 4n^2$

for $n = 6$

$5(6^2) - 6^2 \geq 4(6^2)$

$144 \geq 144$

condition is true

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

$C_2 = 6$

$\therefore 5n^2 - 6n \leq 6n^2$

for $n = 6$

$144 \leq 216$

condition is true.

$\therefore 4n^2 \leq 5n^2 - 6n \leq 6n^2$ for $C_1 = 4, C_2 = 6, n_0 = 6$

Q3) Determine the frequency counts for all statements in the following 2 algorithm statement.

Statements	S/E	frequency	total Steps
for i=1 to n do	1	1+n	1+n
for j=1 to i do	1	1+i	n(i+1)
for k=1 to j do	1	1+j	ni(j+1)
x = x+1;	1	n	nij
			1 + 2n+2ni+2nij
i = 1	1	1	1
while (i<n) do	1	n+1	n+1
{	0		
x = x+1;	1	n	n
i = i+1;	1	n	n
}	0	-	
			3n+2

Q4) Average following growth rate in increasing order:
$(\log n)^3, O(1), O(n^2), O(n \log n), O(n^2 \log n), O(n^{0.5}), O(n^3)$

~~$O(1), O(\log n)$~~

$O(1) < O(n^{0.5}) < O(\log n)^3 < O(n \log n) < O(n^2) < O(n^2) < O(n^2 \log n) < O(n^3)$

Q3) What are Asymptotic Notations? Explain Big-Oh, Omega, Theta

- The running time of a algorithm depends on various characteristics and slight variation in the characteristics varies the running time.

- The algorithm efficiency and performance in comparision to alternate algorithm is best described by order of growth of running time of an algorithm.

- Suppose one algorithm for a problem has time complexity $C_3 n^2$ and another has $C_1 n^3 + C_2 n^2$ then it can be easily observed that algorithm with complexity $C_3 n^2$ will be faster that one with complexity $C_1 n^3 + C_2 n^2$ for sufficiently larger values of n.

- Whether be the values of C_1, C_2 and C_3 there will be an n beyond which algorithm with complexity $C_3 n^2$ is faster, we refer this n has a break down point.

- It is difficult to measure the correct break down point analytically so asymptotic notation are introduced that describe the algorithm efficiency and performance in meaningful way.

- These notation defines the Time Space Complexity for large instance characteristics some commonly used asymptotic notation are.

* **Big Oh notation (O)**

The upper bound for the function 'f' is provided by big Oh notation. Considering 'g' to be a function from non negative integer to positive real numbers, we defined(g) as set of function f. For some real constant $c > 0$ and some non-negative integer constant n_0, $f(n) \leq c g(n)$ for all $n \geq n_0$, then $F(n) = O(g(n))$

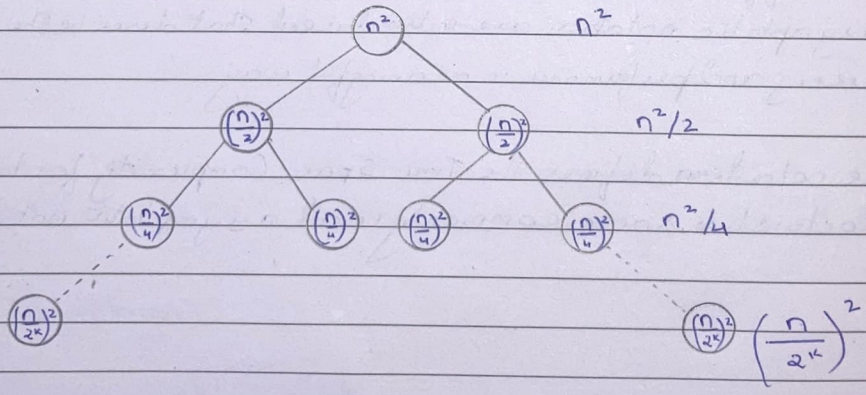
* **Big Omega Notation (Ω)**

The function $f(n) = \Omega(g(n))$ if only there exists positive constant C and n_0 such that $f(n) \geq C * g(n)$ for all n, $n \geq n_0$

* **Big ~~Theta~~ Theta Notation (Θ)**

The function $f(n) = \Theta(g(n))$ if there exits positive constants c_1, c_2 and n_0 such that $c_1 g(n) < f(n) < c_2 g(n)$ $\forall$ n, $n > n_0$

Q6) Solve the following using recursive ~~tree~~ tree method



Total time $= n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \cdots \frac{n^2}{2k^2}$

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

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

$= 2n^2$

$\approx n^2$

$\approx \underline{\underline{O(n^2)}}$

Q5) Solve the following using Master's Theorem:

a) $T(n) = 4T(n/2) + n$

$a = 4 \quad b = 2 \quad f(n) = n$

$T(n) = n^{\log_2 4} * u(n)$ $\because T(n) = n^{\log_b a} * u(n)$

$u(n) \Rightarrow n(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n}{n^{\log_2 4}} = \frac{n}{n^2} = \frac{1}{n} = n^{-1}$

$\therefore u(n) = O(1)$ $\because r = -1 \quad r < 0$

$T(n) = n^2 * u(n)$

$= n^2 * O(1)$

$= O(n^2)$

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

$a = 4 \quad b = 2 \quad f(n) = n^3$

$T(n) = n^{\log_b a} * u(n) = n^{\log_2 4} * u(n) = n^2 * u(n)$

$u(n) = h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n^3}{n^{\log_2 4}} = \frac{n^3}{n^2} = n^1$

$\because x = 1 \quad x > 0$

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

$\therefore T(n) = n^2 * u(n)$

$= n^2 * O(n)$

$= O(n^3)$

Q8) Solve the following matrix multiplication using strassen matrix multiplication method

$C = A * B$

$$A = \begin{bmatrix} 11 & 12 \\ 1 & 2 \\ 21 & 22 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 11 & 12 \\ 5 & 6 \\ 21 & 22 \\ 7 & 8 \end{bmatrix}$$

$P = (1+4)(5+8) = 5 \times 13 = 65$

$Q = (3+4)5 = 7 \times 5 = 35$

$R = 1(6-8) = 1 \times -2 = -2$

$S = 4(7-5) = 4 \times 2 = 8$

$T = (1+2)8 = 3 \times 8 = 24$

$U = (3-1)(5+6) = 2 \times 11 = 22$

$V = (2-4)(7+8) = -2 * 15 = -30$

$C_{11} = 65 + 8 - 24 - 30 = 19$

$C_{12} = -2 + 24 = 22$

$C_{21} = 35 + 8 = 43$

$C_{22} = 65 + (-2) - 35 + 22 = 50$

$$C = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

$$\therefore C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Q9) Explain the best case, worst case and average case scenario for quicksort algorithm shows its recurrence formulation and complexities obtained.

* Analysis of Quicksort

- Time taken by quick sort, in general can be written as:

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

- The first two terms are for two recursive called, the last term is for partition process k, is the number element which are smaller than Pivot.

- The time taken by quicksort depends on the input array and partition strategy

- Following are 3 cases:

* Worst Case

- This case occurs when partition process always picks greatest or smallest element as pivot.

- If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when array is already sorted in increasing or decreasing order.

- Following is the recurrance for worst case:

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

- which means only 1 element in LHS and all others in RHS
- which is equivalent To:

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

let $n-k = 0$

$n = k$

$T(0) = 1$

$T(n) = 1 + n \cdot n$

$= O(n^2)$

* Average case

- We can get an idea of average case by considering case when partition is at P

$0 <= P <= n-1$ where P indicating pivot point

$(0 - to\ P-1) \leq P < (P+1) + to\ (n-1)$

No of elements on LHS = UB - LB + 1
= P(-1) - 0 + 1
= P
= T(P)

No. of element of RHS = UB - LB + 1
= (n-1) - (P+1) + 1
= n - P - 1
= T(n-1-P)

Average case Time = T(P) + T(n-1-P)
where T(n) = O(nlogn)
∴ Avg time case = O(nlogn)

* Best case

- The best case occurs when partition process always picks middle element as pivot.

- Following is the recurrence relation

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

$a = 2, b = 2$ $f(n)$

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

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

$$= n\, u(n)$$

$u(n)$ depends on $h(n)$

$$h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n}{n} = 1$$

$$k = 0, \quad i = 0$$

$$\therefore h(n) = \left(\frac{\log n}{i+1}\right)^{i+1} = \log(n)$$

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

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

Page No :
Date : / /

Q10) Write and explain with example the k^{th} smallest element algorithm, also state its complexity.

```
Algorithm select 1 (a, n, k)
// Select the k^th smallest element in a[1:n] and place it in
// the k^th position of a[]. The remaining elements are rearranged
// such that a[m] ≤ a[k] for 1 ≤ m < k & a[m] ≥ a[k]
// for k < m ≤ n

{
    low := 1; up := n+1;
    a[n+1] := ∞; // a[n+1] is set to infinity
    repeat
    {
        // each time the loop is entered
        // 1 ≤ low ≤ k ≤ up ≤ n+1
        j := partition(a, low, up);
        // j is such that a[j] is the j^th smallest value in a[]
        if (k = j) then return;
        else if (k < j) then up := j;
        // j is the new upper limit
        else low := j+1; // j is the new lower limit
    } until(false);
}
```

- The space needed by select 1 is O(1)
- worst case complexity of select 1 is $O(n^2)$
- The time is $\Omega(n^2)$
- The average computing time of select 1 is O(n)

* Example

- Assume a array of 9 elements, 65, 70, 75, 80, 85, 60, 55, 50 & ∞ we mark a[10] = ∞ if k = 5 then 1st call will be sufficient as 65 is placed at a[5] assume that we are looking for 7th element

k = 7, next invocation of partition is (6, 10)

	(5)	(6)	(7)	(8)	(9)	(10)
i:	65	85	80	75	70	+∞
	65	70	80	75	85	+∞
	65	70	80	75	85	+∞
	65	70	75	80	85	+∞

∴ The correct k^{th} smallest element is found after an interchange between a[7] and a[8] and return it