Question : 1

Do the detail complexity analysis of the following codes in terms of n.

<table>
<tr><th>S.No</th><th>Code</th><th>Analysis</th></tr>
<tr><td>1</td><td><pre>for(i = n ; i > 0; i++)
{
        for(j = 0; j<n;j++)
{
                cout<<i;
        }
}</pre></td><td>The outer loop runs infinite times.
No time complexity</td></tr>
<tr><td>2</td><td><pre>for(i = n ; i > 0; i++)
{
        for(j = 0; j < n;j * 2)
        {
                cout << i;
        }
}</pre></td><td>The outer loop runs infinite times
NO time complexity</td></tr>
<tr><td>3</td><td><pre>while(low <= high)
{
  mid = (low + high) / 2;
  if (target < list[mid])
    high = mid - 1;
  else if (target > list[mid])
    low = mid + 1;
  else break;
}</pre></td><td>⇒ Binary search
$n = 2^k \Rightarrow \log_2 n = k$
$O(\log_2 n)$</td></tr>
<tr><td>4</td><td><pre>int i, j, imin;
for(i = 0; i<size-1; i++) → n
{
   imin = i;
   for(j = i+1; j<size; j++) → n
        {
                if(array[j] < array[imin])
                        imin = j;
        }
        swap(array[i], array[imin]);
}</pre></td><td>⇒ Both loops run 'n' times so,
$O(n^2)$</td></tr>
</table>

5	int key, j; → (n-1) for(int i = 1; i<size; i++) { key = array[i]; → (n) j = i; while(j > 0 && array[j-1]>key) { array[j] = array[j-1]; j--; } array[j] = key; }	The outer loop runs for 'n-1' times and inner loop for 'n' times: $n(n-1)$ (n^2-n) so, $O(n^2)$
6	for (int i=0; i<n; i++) { → $n-1$ for (int j=i; j<= i*i; j++) { → n^2 if (j%i == 0) { for (int k=0; k<j; k++) → n { printf("*"); } } } }	Loop1 = $n-1$ times Loop2 = n^2 Loop3 = n so, $n \times n^2 \times (n-1) = n^4 - n^3$ $\Rightarrow O(n^4)$
7	for (int i=0; i<n; i++) { for (int j=i; j<= i*i; j++) { print("*"); } }	Loop1 = $n-1$ Loop2 = n^2 so, $n^2(n-1) = n^3 - n^2$ $\Rightarrow O(n^3)$
8	for (int i=0; i<n; i++) { for (int j=i; j=i*2;j*2) { for (int k=1; k< j; k*2) { print("*"); } } }	
9	for (int i=0; i<n; i++) { for (int j=i; j=i*2;j*2) { for (int k=j; k< j; k*2) { print("*"); } } }	

10	void function (int n) { int count = 0; for (int I = n/2; i<n; i++) for (int j =1; j+n/2<=n; j = j++) for (int k =1; k<=n; k = k*2) count++; }	Loop1 = $n/2$ Loop2 = $n/2$ Loop 3 = $\log n$ $n/2 \times n/2 \times \log n = n^2 \log n$
11	for I = 1 to n { // some operations of O(1) for j = 1 to 2*i { // some operations of O(1) k = j; while (k>=0) { // some operations of O(1) k=k-1; } } }	
12	for (int i=0; i<n; i++) for (int j=i; j< i; j+=i) { print("*"); } }	Loop 1 : n times Loop 2 : 0. $\Rightarrow O(n)$
13	for (int i=0; i<n; i++) for (int j=i; j< i*i; j++) { for (int k=j; k< j; k+=j) { print("*"); } } }	$\Rightarrow$ loop1: n times $\Rightarrow$ loop2: n^2 times $\Rightarrow$ loop3: 1 so, $n(n^2)(1) \Rightarrow O(n^3)$
14	for (int i=0; i<n; i*i) for (int j=i; j< n; j++) { print("*"); } }	$\Rightarrow$ The loop runs infinite times No time complexity
15	for (int i=n; i>0; i--) for (int j=n; j > i; j--) { print("*"); } }	$\Rightarrow$ Loop1 = n $\Rightarrow$ Loop2 = n $n \times n \Rightarrow O(n^2)$

16	for (int i=0; i<n; i*2) { for (int j=0; j< i*2; j++) { print("*"); } }	⇒ The loop runs infinite time ⇒ No time complexity
17	for (int i=0; i<n; i++) for (int j=i; j< i*i; j++) { for (int k=j; k< j; k+=j) { print("*"); } } }	Loop1 : n times Loop2 : n^2 So, $n \times n^2$ $O(n^3)$
18	for (int i=0; i<n; i++) for (int j=i; j< n/2; j++) { for (int k=j; k< j; k+=j) { print("*"); } } }	Loop1 = n times Loop2 = $n/2$ times loop3 = 1 So, $n \times n/2 \times 1 \Rightarrow O(n^2)$
19	for (int i=0; i<n*2; i++) for (int j=i; j< n*2; j++) { print("*"); } }	Loop1 = n times Loop2 = n times So, $n \times n \Rightarrow O(n^2)$
20	for (int i=0; i<n*2; i++) for (int j=i; j< n*2; j++) { for (int k=j; k< j; k+=j) { print("*"); } } }	Loop1 : n times Loop2 : n times Loop3 = 1 times So, $n \times n \times 1 \Rightarrow O(n^2)$

21	``` for (int i=0; i<n; i*2) for (int j=i/2; j< i; j*2) { for (int k=j; k< j; k+=j) { print("*"); } } } ```	Loop runs in finite times So, no time complexity
22	``` void fun (int n, int k) { for (int i=1; i<=n; i++) { int p = pow (i, k); for (int j=1; j<=p; j++) { // Some O(1) operation } } } ```	
23	``` for (int i = 1; i <= n; i++) { for (int j = 1; j < n; j += i) { // Some O(1) operations } } ```	Loop1: n Loop2: n So, $n \times n \Rightarrow O(n^2)$
24	``` for (int i = 1; i <= n; i++) { for (int j = 1; j < i*i; j++) { // Some O(1) operations } } ```	Loop 1: n Loop 2: n^2 So, $n \times n^2 \Rightarrow O(n^3)$
25	``` for (int i = 1; i <= n; i++) { for (int j = 1; j < i*i; j*=2) { // Some O(1) operations } } ```	Loop1: n Loop2:

Question : 2

Solve the following recurrence relation using Iteration method.

a) $T(n) = 2 + T(n-1)$ for $n > 0$, $T(0) = 2$ for $n > 0$

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

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

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

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

Base case :

$n - k = 0$

$n = k$

$T(n) = T(0) + 2 \times n$

$= 2 + 2n$

$\Rightarrow O(n)$

b) T(n) = 2T (n / 2) + n, T (1) = 1

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

$$= 2\left[2T(n/4) + n/2\right] + n$$

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

$$= 2^2\left[2T(n/8) + n/4\right] + n + n$$

$$= 2^3 T(n/8) + n + n + n$$

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

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

$$\Rightarrow 2^k T(n/2^k) + kn$$

$$\boxed{\begin{aligned} T(n/2) &= 2T(n/4) + n/2 \\ T(n/4) &= 2T(n/8) + n/4 \end{aligned}}$$

$$n/2^k = 1 \Rightarrow n = 2^k \Rightarrow \log_2 n = k$$

$$= nT(1) + (\log_2 n) \times (n)$$

$$= n \times 1 + n \log_2 n$$

$$= n + n \log_2 n$$

$$\Rightarrow O(n \log_2 n)$$

$$\Rightarrow T(n/4) = 3T(n/16) + n/4$$

$$\Rightarrow T(n/16) = 3T(n/64) + n/16$$

c) T(n) = 3T (n / 4) + n, T (1) = 1

$$T(n) = 3T(n/4) + n \quad , T(1) = 1$$

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

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

$$T(n) = 3^2\left[3T\left(\frac{n}{64}\right) + \frac{n}{16}\right] + \frac{3n}{4} + n.$$

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

$$T(n) = 3^k T\left(\frac{n}{4^k}\right) + \underbrace{\left(\frac{3}{4}\right)^{k-1} n + \left(\frac{3}{4}\right)^{k-2} n + \cdots \left(\frac{3}{4}\right)^0 n}_{\text{geometric series}}$$

$$\frac{n}{4^k} = 1$$

$$n = 4^k$$

$$\log_4 n = k.$$

as $\frac{3}{4} < 1$

So, $\frac{1}{1 - 3/4} = 4$

$$T(n) = 3^{\log_4 n} + 4(n)$$

$$\Rightarrow O(n).$$

Base Case

$* \; T(0) = 0$

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

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

$* \; T(n-1) = 2T(n-2)+1$

$* \; T(n-2) = 2T(n-3)+1$

$* \; T(n-3) = 2T(n-4)+1$

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

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

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

$= 2^3T(n-3)+4+2+1$

$= 2^3[2T(n-4)+1]+4+2+1$

$= 2^4T(n-4)+8+4+2+1$

$= 2^kT(n-k)+2^k-1$

$\Rightarrow$ Base case $T(0)=0$

So, $n-k=0$

$n=k$

$= 2^n \times T(0) + 2^n - 1$

$= 0 + 2^n - 1$

$\Rightarrow O(2^n)$

Base case
$*T(1)=1$

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

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

$T(n/4) = 4T(n/16)+4$

$T(n/16) = 4T(n/64)+4$

$T(n) = 4[4T(n/16)+4]+4$

$= 4^2T(n/16)+16+4$

$T(n) = 4^2[4T(n/64)+4]+16+4$

$T(n) = 4^3T(n/4^3)+4^3+4^2+4^1$

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

$\frac{n}{4^k}=1 \Rightarrow n=4^k$

$$S_k = \frac{a(r^n-1)}{r-1} = \frac{4(4^k-1)}{4-1}$$

$$= \frac{4(4^k-1)}{3} = \frac{4(n-1)}{3}$$

$$T(n) = nT(1) + \frac{4(n-1)}{3}$$

$$= n + \frac{4n-4}{3}$$

$\Rightarrow O(n)$

Question 4:

Solve the following recurrence relation using Master Theorem.

a) $T(n) = 9T(n / 3) + n$

$a = 9 \geq 1$

$b = 3 \geq 1$

$f(n) = n$

$f(n) = O(n^{\log_b a})$

$n = O(n^{\log_b a})$

$n = O(n^{\log_3 9})$

$n = O(n^2)$ True

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

$T(n) = \Theta(n^{\log_3 9})$

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

b) T(n) = 4T (n / 4) + n / log² n

This cannot be solved by master theorem because we can see that '$\log_2 n$' is in denominator which means it is not a polynomial.

Master Theorem can not be applied:

① T(n) is not monotone
② f(n) is not polynomial
③ b can not be expressed as a constant

c) $T(n) = 4T(n/3) + n^3$

Case i)

$f(n) = O(n^{\log_b a})$

$n^3 = O(n^{\log_3 4})$

$\underbrace{n^3 = O(n^{1.26})}$

$\Rightarrow$ false

case (ii)

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

$n^3 = \Theta(n^{\log_3 4} \cdot \log^0 n)$

$n^3 = \Theta(n^{1.26} \cdot 1)$

$\underbrace{n^3 = \Theta(n^{1.26})}$

false

case (iii)

$f(n) = \Omega(n^{\log_b a})$

$n^3 = \Omega(n^{\log_3 4})$

$\underbrace{n^3 = \Omega(n^{1.26})}$

True

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

$\boxed{T(n) = \Theta(n^3)}$

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

This cannot be solved by master theorem because '1/n' is not a polynomial.

Master Theorem can not be applied:.

① T(n) is not monotone

② f(n) is not polynomial

③ b cannot be expressed as a constant

e) T (n) = 2T (n/2) + n log n

$a = 2 \geq 1$

$b = 2 > 1$

$f(n) = n \log n$

case (i)

$f(n) = O(n^{\log_b a})$

$n\log n = O(n^{\log_2 2})$

$\underbrace{n \log n = O(n)}_{\text{false}}$

case (ii)

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

$n \log n = \Theta(n^{\log_2 2} \cdot \log^1 n)$

$\underbrace{n \log n = \Theta(n \log n)}_{\text{true}}$

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

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

f) T (n) = 64T (n/8) – n 2 log n

$a = 64 \geq 1$

$b = 8 \geq 1$

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

<u>case(i)</u> $f(n) = \Theta(n^{\log_b a})$

$n^2 \log n = O(n^{\log_8 64})$

$\underbrace{n^2 \log n = O(n^2)}_{\text{false}}$

<u>case(ii)</u>

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

$n^2 \log n = \Theta(n^{\log_b a} \log^k n)$

$\underbrace{n^2 \log n = \Theta(n^2 \log n)}_{\text{True}}$

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

g) T (n) = T (n/2) + n (2 – cos n)

This can not be solved by master theorem because f(n) is not monotone function

Master theorem can not be applied:

① T(n) is not monotone

② f(n) is not polynomial

③ b can not be expressed as constant

$a \in 0.5 \le 1$ (master theorem can't be applied).

h) T (n) = 0.5T (n/2) + 1/n

This can not be solved by master theorem because '$1/n$' is not polynomial.

master theorem cannot be applied:-

① T(n) is not monotone

② f(n) is not polynomial

③ b can not be expressed as a constant