DESIGN AND ANALYSIS OF ALGORITHMS
ASSIGNMENT - 1

Name : Rishita Gaur
Section : CCIS
Roll no : 2014442

Q1) What do you understand by Asymptotic notations. Define — — — — examples.

Ans) Asymptotic notations : are methods / languages using which we can define the running time of the algorithm based on input size. These notations are mathematical tools to represent the complexities.

There are 3 notations that are commonly used :

(i) Big-Oh Notation

- Big-oh (O) notation gives an upper bound of a function f(n) to within a constant factor

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

such that $f(n) \leq c \cdot g(n)$ $\forall$ $n \geq n_0$, some constant $c > 0$

eg: $f(n) = n!_0 = n \cdot (n-1) \cdot (n-2) \cdots 1$

$1 \times 2 \times 3 \times 4 \cdots \times n \leq n \times n \times n \times n \cdots$

$n!_0 \leq n^n$

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

(i) Big Omega Notation

Big Omega (Ω) notation gives a lower bound for a function $f(n)$ to within a constant factor.

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

iff $f_n \geq c \cdot g(n)$

$\forall\ n \geq n_0$ & some constant $c > 0$.

Eg: $f(n) = n!$

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

$$1 \leq n!$$

$$\therefore f(n) = \Omega(1)$$

(ii) Theta Notation

Theta gives both tight upper & lower bound for a function.

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

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

$\forall\ n \geq \max(n_1, n_2)$

$c_1, c_2 > 0$

(Q2) Ans)

Calculate Time Complexity

```
for ( i=1 to n)
  { i = i * 2 ; }
```

Suppose the loop ends in k steps

$$\therefore n = 1 \times 2 + 2 \times 2 + 4 \times 2 + 8 \times 2 - - - - \cdots$$

$$n = 2^1 + 2^2 + 2^3 + 2^4 + \cdots 2^k$$

Sum of increasing GP $= \dfrac{a(r^k - 1)}{r - 1}$

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

SHOT ON REDMI Y3
AI DUAL CAMERA

$$n = 2 \cdot (2^k - 1)$$

$$\frac{n}{2} = 2^k - 1$$

$$\frac{n+2}{2} = 2^k$$

$$\log\left(\frac{n+2}{2}\right) = \log_2 2^k$$

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

$$\therefore O\left(\log\left(\frac{n+2}{2}\right)\right) \simeq O(\log_2 n)$$

Q3) $$T(n) = \begin{cases} 3T(n-1) & , n > 0 \\ 1 & , n \leq 0 \end{cases}$$

Ans) Using Backward substitution,

$$T(n) = 3 \cdot T(n-1)$$

$$T(n-1) = 3 \cdot T(n-2)$$

$$\therefore T(n) = 3 \cdot 3 \cdot T(n-2)$$

$$\rightarrow T(n) = 3^k T(n-k)$$ (let total terms be k)

Given $T(0) = 1$

$$\therefore n - k = 1$$

$$k = n - 1$$

$$T(n) = 3^{n-1} T(n - n + 1)$$

$$= 3^{n-1} T(1) \quad \text{---} ①$$

Putting value of $n = 1$ in equation

$$T(1) = 3T(1-1)$$

$$= 3 \cdot T(0)$$

$$= 3 \quad [T(0) = 1]$$

Eq ① becomes : $3^{n-1} \cdot 3$

$$= 3^n$$

$\therefore$ Time Compl. $= O(3^n)$

SHOT ON REDMI Y3
AI DUAL CAMERA

Q4) $T(n) = \begin{cases} 2T(n-1) - 1 & \text{if } n > 0 \\ 1 & \text{if } n \leq 0 \end{cases}$

Ans) Using Back substitution,

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

$T(n-1) = 2 \cdot T(n-2) - 1$

$\rightarrow T(n) = 2 \cdot (2T(n-2) - 1) - 1$

$= 2 \cdot 2 \cdot T(n-2) - 2 - 1$

$\therefore$ General term $= 2^k \cdot T(n-k) - (1 + 2 + 4 + \cdots 2^{k-1})$ — ①

Given: $T(0) = 1$

$\therefore n - k = 1$

$k = n - 1$

Putting value of k in eqn. ①

$= 2^{n-1} T(n - n + 1) - (1 + 2 + 4 + \cdots + 2^{n-1-1})$

$= 2^{n-1} T(1) - (1 + 2 + 4 + \cdots 2^{n-2})$

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

$= 2^{n-1} T(1) - (2^{n-1} - 1)$ — ②

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

$= 2T(0) - 1 \quad (T(0) = 1)$

$= 2 \cdot 1 - 1 = 1$

$\therefore T(n) = 2^{n-1} - (2^{n-1} - 1)$

$= 2^{n-1} - 2^{n-1} + 1$

$= 1$

$\therefore$ Time complexity $= O(1)$

SHOT ON REDMI Y3
AI DUAL CAMERA

Q5) Find Time complexity

Ans

```
int i=1, s=1;
while (s <= n)
{ i++;
  s = s+i;
  printf ("# ");
}
```

Suppose the loop ends in k steps.

s after 1st iteration = 1+2

" " 2nd " = 1+2+3

" " 3rd " = 1+2+3+4 . . .

$\therefore 1+1+2+3+4 \cdots k = n$

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

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

Eliminating lower powers of k & constants

$$k^2 = n$$

$$k = \sqrt{n}$$

$\therefore$ Time complexity = $O(\sqrt{n})$

Q6)

Ans)

```
void function (int n)
{ int i, count = 0;
  for (i=1; i*i <= n; i++)
      count ++;
}
```

loop ends when $i^2 > n$

$i > \sqrt{n}$

$\therefore$ Time complexity = $O(\sqrt{n})$

Q7)
Ans)

```
void function (int n) {
    int i, j, k, count = 0;
    for (i = n/2 ; i <= n ; i++)
        for( j=1; j<=n; j = j*2)
            for (k=1; k<=n; k = k*2)
                count++
}
```

T.C. for 1st loop = $O(\frac{n}{2}) \simeq O(n)$
T.C. for 2nd loop = $O(\log n)$
T.C. for 3rd loop = $O(\log n)$

Total Time complexity = $O(n \times (\log n)^2)$

Q8)
Ans)

```
function (int n) {
    if (n==1) return;          ——— O(1)
    for (i=1 to n) {
        for (j=1 to n) {
            printf ("*");      } O(n^2)
        }
    }
    function (n-3);            ——— ø T(n-3)
}
```

$$T(n) = \begin{cases} T(n-3) + n^2 & , \text{otherwise} \\ 1 & n=1 \end{cases}$$

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

SHOT ON REDMI YS
AI DUAL CAMERA

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

General term;

$T(n) = T(n-(k+3)) + n^2 + (n-1)^2 + \cdots + (n-k)^2$

Given: $T(1) = 1$

$n-(k+3) = 1$

$n-1-3 = k$

$n-4 = k$

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

$= T(1) + 4^2 + 5^2 + \cdots + n^2$

Sum of squares of n natural nos $= \frac{n(n+1)(2n+1)}{6}$

" " " " numbers starting from 4 $= \frac{n(n+1)(2n+1) - 1^2 - 2^2 - 3^2}{6}$

$= \frac{n(n+1)(2n+1)}{6} - 14$

$\because T(n) = T(1) + \frac{n(n+1)(2n+1)}{6} - 14$

$T(n) = 1 + \frac{2n^3 + n^2 + 2n^2 + n}{6} - 14$

Eliminating lower order terms & constants

$T(n) = O(n^3)$

⟶

(Q9)

Ans

```
void function (int n)
{  for (i=1 to n)
   {  for (j=1; j<=n; j=j+i)
         printf ("*");
   }
}
```

i j

Suppose loop for j runs k times

After 1st iteration, $j = 1 + i$

" 2nd " , $j = 1 + i + i$

$\therefore j = 1 + k \cdot i$

After loop ends,

$1 + k \cdot i = n$

$k = \frac{n-1}{i}$

$\therefore$ T.C. for inner loop $= O\left(\frac{n-1}{i}\right)$

$\simeq O(n)$

Overall complexity = innerloop × outerloop

$= O(n^2)$

→

(Q10)

Ans)

$f(n) = n^k$

$g(n) = a^n$, $k >= 1$, $a > 1$

For $a > 1$, the exponential value a^n far outgrows any polynomial term (i.e. n^k) $\therefore$

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

or $n^k = O(a^n)$

value of $c \geq \frac{n^k}{a^n}$, n_0

(Q11)

Ans)

```
void fun(int n)
{ int j=1; i=0;
  while (i<n) {
    i = i+j; j++;
  }
}
```

SHOT ON REDMI
AI DUAL CAMERA

For 1st iteration, $i = 0+1$

" 2nd " , $i = 0+1+2$

" 3rd " , $i = 0+1+2+3$

" kth " , $i = 0+1+2+3+\dots+k$

After loop $i = \frac{k(k+1)}{2}$

After loop terminates,

$i >= n$

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

$k^2 >= n$ (ignoring other terms)

$k = \sqrt{n}$

∴ Time complexity = $O(\sqrt{n})$

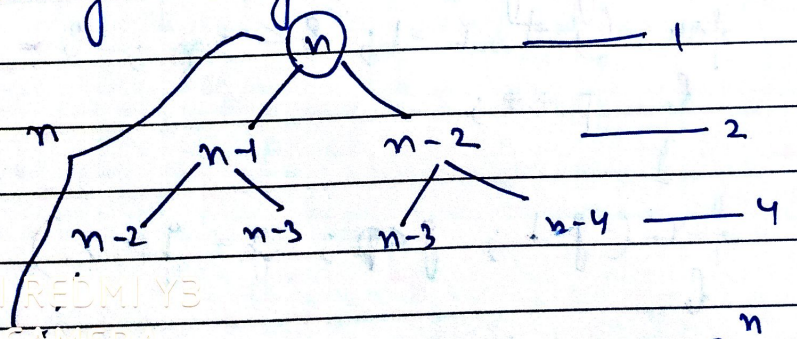
(Q12)
Ans)

Fibonacci series :

```
int fib(int n)
{ if (n<=1)
    return n;
  return fib(n-1) + fib(n-2);
}
```

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

Solving using tree method.



SHOT ON REDMI Y3
AI DUAL CAMERA

∵ $T(n) = 1+2+4+\ldots+2^n$

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

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

$\simeq O(2^n)$

Here there is one entry at stack at every function call. Maximum entry at any instance is always less than or equal to n.

∵ space complexity = $O(n)$

(Q13)

Ans)

(i) $O(n \log n)$

```
for (int i=1; i<=n; i++)
    for (int j=1; j<=n; j=j*2)
        count++
```

(ii) $O(n^3)$

```
for (int i=1; i<=n; i++)
    for (int j=1; j<=n; j++)
        for (int k=1; k<=n; k++)
            count++
```

(iii) $O(\log \log n)$

```
for (int i=1; i<=n; i=i*2)
{   p++;
}
for (j=1; j<p; j=j*2)
{
    count++;
}
```

SHOT ON REDMI 9
AI DUAL CAMERA

Q14)

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

Using master's theorem

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

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

$a = 1, \; b = 4$

$\because$ constant $p = \log_b a = \log_4 1 = 0$

$\because n^p = n^0 = 1$

~~Case~~ As if $f(n) > n^c$, $T(n) = \Theta(f(n))$

$\because T.C = \Theta(cn^2)$

$\simeq \Theta(n^2)$

Q15)
Ans)

```
int f fun (int n) {
    for (int i=1; i <= n; i++) {
        for (int j=1; j<n; j += i)
            { O(1) }
    }
}
```

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

Q16)

```
for (int i = 2; i <= n; i = pow (i, k))
    { O(1); }
```

$i = 2^k, (2^k)^k, ((2^k)^k)^k \ldots\ldots$

$= 2^k, 2^{k^2}, 2^{k^3} \ldots\ldots 2^{k^p}$

Loop terminates when

$2^{k^p} > n$

To find when $2^{k^p} = n$, take $\log_2$ on both sides

$\log_2 2^{k^p} = \log_2 n$

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

$k^p = \log_2 n$

Taking log again,

$p \log_k k = \log_k \log_2 n$

$p = \log_k \log_2 n$

$\therefore O(\log_k \log_2 n)$

or $O(\log \log n)$

SHOT ON REDMI Y3
AI DUAL CAMERA

Q18) (a) n, $n!$, $\log n$, $\log\log n$, $\text{root}(n)$, $\log(n!)$, $n\log n$, 2^n, 2^{2n}, 4^n, n^2, 100

Order: $100 < \log\log n < \log n < \sqrt{n} < n < n\log n < \log(n!) < n^2 < 2^n = 4^n < n!$

(b) $1 < \log\log(n) < \sqrt{\log(n)} < \log(n) < 2\log(n) < \log(2n) < n < 2n < 4n < \log(n!) < n^2 < 2\cdot(2^n) < n!$

(c) $96 < \log_8(n) < \log_2(n) < n\log_6(n) < n\log_2(n) < 5n < \log(n!) < 8n^2 < 7n^3 < 8^{2n}$

Q19)

Ans)

```
for ( int i=0; i <= n-1 ; i++)
  {
    if ( arr[n-1] < key || arr[i] >= key)
      {
        if ( arr[i] == num)
          printf (" present");
        else
          printf (" Not present ");
        break;
      }
  }
```

Q20)

Ans) Iterative insertion sort

```
insertionsort (int a[], int n) int key)
{   for (int i = 1; i < n; i++) {
      int key = i
      int j = i-1;
      while (j >= 0) {
```

SHOT ON REDMI Y3
AI DUAL CAMERA

```
        if ( a[j] > key)
          {  a[j+1] = a[j] }
        j--;
      }
      a[j+1] = key
    }
```

→ Recursive approach :

```
void insertionSort (int arr[], int n)
  {  if (n<=1)
       return;
     insertionSort ( arr, n-1);
     int last = arr[n-1]
     int j = n-2;

     while (j>=0  && arr[j] > last)
     {
          arr[j+1] = arr[j];
          j--;
     }
     arr[j+1] = last;
  }
```

Q22)

	stable?	inplace?
selection sort	no	yes
insertion "	yes	yes
shellsort	no	yes
quicksort	no	yes
mergesort	yes	no
heapsort	no	yes

SHOT ON REDMI Y3
AI DUAL CAMERA

Q4) int BinarySearch (int a[], int l, int r, int key)

T(n)

```
{
    if (l <= r)
    { int mid = l + (r-1)/2 ;
      if (a[mid] == key)
          return mid;
    }                                   O(1)
    else if (a[mid] > key)              T(n/2)
        return BinarySearch(a, l, mid-1, key);
    else                                T(n/2)
        return BinarySearch(a, mid+1, r, key);
  }
  return -1;
}
```

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

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

$\therefore$ T.C = $O(\log n)$