DAA - ASSIGNMENT-1

Q1 Asymptotic Notation - These are the methods used to define the complexity/ running time of an algorithm based on input size.

Types of Asymptotic Notations -

1) Big-O - It is used for worst case or ceiling of growth for a given function i.e. function's complexity will not cross the growth of asymptotic notation in any case.

$$\boxed{f(n) = O(g(n)) \text{ iff } f(n) \leq c\,g(n) \ \forall\ n \geq n_0 \text{ and } c > 0}$$

Here $g(n)$ is tight upper bound of $f(n)$.

Eg: $(n^2+2n) = O(n^3)$, $n + \log n = O(n)$

2) Big-Omega (Ω) - It is used for best case or floor growth rate of a given function. It provides us with asymptotic lower bound for growth rate of an algorithm

$$\boxed{f(n) = \Omega(g(n)) \text{ iff } f(n) \geq c\,g(n) \ \forall\ n \geq 0 \ \& \ c > 0}$$

Eg: $n^2+n = \Omega(n)$, $n + \log n = \Omega(\log n)$

3) Theta (Θ) - It denotes the asymptotic tight bound of growth rate of runtime of an algorithm.

$$\boxed{f(n) = \Theta(g(n)) \text{ iff } c_1 g(n) \leq f(n) \leq c_2 g(n) \ \forall\ n \geq \max(n_1, n_2),\ c_1, c_2 > 0}$$

Eg: $2n^2+n = \Theta(n^2)$, $n + \log n = \Theta(n)$

4) Small-Oh (o) - It is used to denote upper bound (not asymptotically tight) on growth rate of runtime of algorithm.

$$\boxed{f(n) = o(g(n)) \text{ iff } f(n) < c \cdot g(n) \ \forall\ n > n_0 \text{ and } c > 0}$$

5) Small-omega (ω) - It denotes the lower bound (not asymptotically tight) on growth rate of runtime of an algorithm.

$$\boxed{f(n) = \omega(g(n)) \text{ iff } f(n) > c \cdot g(n) \ \forall\ n > n_0,\ c > 0}$$

Eg: $2n^2+n = \omega(n)$, $n + \log n = \omega(\log n)$

Q2 for(i=1 to n)
{ i = i*2 }

$\boxed{\text{Time Complexity} = O(\log n)}$

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

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

$T(1) = 3T(0) = 3$

$T(2) = 3T(1) = 3\cdot 3 = 3^2$

$T(3) = 3T(2) = 3\cdot 3^2 = 3^3$

$\vdots$

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

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

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

Q4 $T(n) = \begin{cases} 2T(n-1)-1 & , n>0 \\ 1 & , n\le 0 \end{cases}$

$T(n) = 2T(n-1) - 1$ — ①

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

① ⇒ $T(n) = 2^2 T(n-2) - 2 - 1$ — ②

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

② ⇒ $T(n) = 2^3 T(n-3) - 2^2 - 2 - 1$

$\vdots$

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

$n-k=0$
$n=k$

$= 2^n T(0) - 2^{n-1} - \dots - 2^2 - 2^1 - 1$

$= 2^n - 2^{n-1} - \dots - 2^2 - 2^1 - 1$

$= 2^n - [2^{n-1} + 2^{n-2} + \dots + 2^2 + 2 + 1]$

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

$= 1$

$\boxed{\therefore T(n) = O(1)}$

Q5
```
int i=1, s=1;
while (s<=n)
{ i++;
  s=s+i;
  printf("#");
}
```

~~$T(n) = 1+3+6+10+15+$~~

$i = 1, 2, 3, 4, 5, \dots, k$

$s = 1, 3, 6, 10, 15, \dots, n$

$\Rightarrow 1+2+3+\dots+k = n$

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

$\Rightarrow k = \frac{-1 \pm \sqrt{1+8n}}{2} \Rightarrow \boxed{T.C. = O(\sqrt{n})}$

Q6
```
void function (int n)
{ int i, count = 0;
  for (i = 1; i*i <= n; i++)
    count ++;
}
```

$i^2 = n$
$i = \sqrt{n}$

$T.C. = O(\sqrt{n})$

Q7
```
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 ++;
}
```

$T.C = O\left((n/2+1)(\log_2 n)(\log_2 n)\right)$
$\simeq O(n(\log_2 n)^2)$

Q8
```
function (int n)
{  if (n == 1)
     return;
   for (i = 1 to n)
     for (j = 1 to n)
       printf ("*");
   function (n-3);
}
```

$T(n) = n^2 + T(n-3)$
$T(1) = O(1)$
$T(n-3) = (n-3)^2 + T(n-6)$
$T(n) = n^2 + (n-3)^2 + T(n-6)$
$T(n-6) = (n-6)^2 + T(n-9)$
$T(n) = n^2 + (n-3)^2 + (n-6)^2 + T(n-9)$
$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \dots + (n-k)^2 + T(n-k-3)$

$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \dots + 4^2 + T(1)$
$= n^2 + (n-3)^2 + (n-6)^2 + \dots + 7^2 + 4^2 + 1 = \sum_{k=1}^{(n+2)/3} (3k-2)^2 = \sum (9k^2 + 4 - 12k)$

$\boxed{T.C \simeq \Theta(n^3)}$

Q9
```
void function (int n)
{  for (i = 1 to n)
     for (j = 1; j <= n; j = j+i)
       printf ("*");
}
```

$i = 1, 2, 3, \dots, n$
$j = \frac{n}{1}, \frac{n}{2}, \frac{n}{3}, \dots, \frac{n}{n}$

$\therefore T(n) = \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \dots + \frac{n}{n}$

$= n\left[1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}\right]$

$= n \log n$

$\boxed{T.C. = O(n \log n)}$

Q10 $n^k \leq ca^n$

$a^n + n^k \leq ca^n$

$a^n + n^k \leq a^n(c-1)$

$\frac{a^n + n^k}{a^n} \leq (c-1)$

$c \geq \frac{1 + n^k}{a^{n_0}} + 1$

$c \geq 2 + \frac{n_0^k}{a^n}$

$c \geq 2 + \frac{n_0^1}{1.5^n}$

$\boxed{n_0 = 1}$

$c \geq 2 + \frac{1}{1.5}$

$c \geq 3 + 1$

$\boxed{c \geq 4}$

Q11
```
void fun (int n)
{ int j = 1, i = 0;
  while (i<n)
  { i = i+j;
    j++;
  }
}
```

$i = 0 \quad 1 \quad 3 \quad 6 \quad 10 \quad \cdots \quad i$

$j = 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad \cdots \quad k$

$T.C = n \quad n-1 \quad n-3 \quad n-6 \quad n-10$

Q12 Recursive Fibonnaci Series.

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

~~int fib(int n)~~
~~{ if (n==0)~~
~~{ cout << "0" << endl;~~
~~return 0;~~
~~}~~
~~if(n==1)~~
~~{ cout << "1"~~ ~~fib (0);~~
~~cout <<~~

①
```
void fib (int n, int a, int b)
{
    if (n==0)
        return;
    int c = a+b;
    cout << c << " ";
    fib (n-1, b, c);
}
```

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

$T(n) = T(n-1) + 1$
$T(1) = T(0) + 1 = 1+1 = 2$
$T(2) = T(1) + 1 = 2+1 = 3$
$\vdots$
$T(n) = n+1$

$\boxed{\therefore T(n) = O(n)}$

~~fib(5)~~

③ Space Comp. = $O(n)$ bcz it uses n-1 calls in stack.

Q13 $T(n) = O(n \log n)$

```
for (int i=0; i<n; i++)
    for (int j=0; j<n; j=j*2)
        cout<<"*";
```

$T(n) = n^3$

```
for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
        for (int k=0; k<n; k++)
            cout<<i<<j<<k<<endl;
```

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

```
for (int i=0; i< log(n); i=i*2)
    cout << i <<" ";
```

Q14 $T(n) = T(n/4) + T(n/2) + c \cdot n^2$

$T(n/2) = T(n/8) + T(n/4) + c \cdot (n/2)^2$

$T(n) = T(n/4) + T(n/8) + T(n/4) + c \cdot \frac{n^2}{2^2} = 2T(n/4) + T(n/8) + c\frac{n^2}{2^2} + cn^2$

$T(n/4) = T(n/16) + T(n/8) + c\left(\frac{n}{4}\right)^2$

$T(n) = 2T\left(\frac{n}{16}\right) + 2T\left(\frac{n}{8}\right) + 2c\frac{n^2}{4^2} + T(n/8) + c \cdot \frac{n^2}{2^2} + c \cdot n^2$

$= 2T\left(\frac{n}{16}\right) + 3T\left(\frac{n}{8}\right) + 2c \cdot \frac{n^2}{4^2} + c \cdot \frac{n^2}{2^2} + c \cdot n^2$

cn^2

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

$T(n/8)$ $T(n/16)$ $T(n/4)$ $T(n/8)$

$T(n) = cn^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \cdots$

This is GP with ratio 5/16

$\therefore T(n) = \frac{n^2}{1 - 5/16} \Rightarrow \boxed{T.C. = O(n^2)}$

Q15
```
int fun(int n)
{ for(int i=1; i<=n; i++)
    for(int j=1; j<n; j+=i)
      O(1)
```

$i = 1, 2, 3, 4, \ldots, n$

$j = n, n/2, n/3, n/4, \ldots, n/n$

$\therefore T.C. = n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$

$= n\left[1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right]$

$= n \log n$

$\boxed{T.C. = O(n \log n)}$

Q16 for(int i=2; i<=n; i=pow(i,k))
{ O(1) }

$i = 2, 2^k, 2^{k^2}, 2^{k^3}, \ldots \ldots 2^{k^x}$ i.e. $(x+1)$ terms

$2^{k^x} = n$

$k^x = \log_2 n$

$x = \log_k(\log_2 n)$

$\therefore \boxed{T.C. = O(\log_k \log_2 n)}$

Q17 $T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right) + cn$

n

$n/100 \quad 99n/100$

$\frac{n}{10000} \quad \frac{99n}{10000} \quad \frac{99n}{100000} \quad \frac{99 \times 99n}{1000000}$

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

$< n^2 < 2^n < 4^n < n! < 2^{2^n}$

Q18 a) $100 < \log(\log n) < \log n < \sqrt{n} < n < \log(n!) < n\log n$

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

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

Q19
```
int linearSearch(int *arr, int n, int key)
{ int i;
  for(i=0; i<n; i++)
    if(arr[i]==key)
      return i;
    else if(arr[i]>key)
      return 1;
}
```

$\boxed{\begin{array}{l} T.C. = O(n) \\ S.C. = O(1) \end{array}}$

Q20 Iterative Insertion Sort

```
void insertionSort (int *a, int n)
{ int i, j, temp ;
    for ( i ← 1 to n)
        temp ← a[i] ·
        j ← i-1
        while (j >=0 && a[j] > temp)
            a[j+1] = a[j]
            j ← j-1
        a [j+1] ← temp ·
}
```

Recursive Insertion Sort

```
void insertionSort (int *a , int n)
{       if (n<2)
            return ;
        insertionSort (a, n-1) ·
        int last ← a [n-1]
        int j ← n-2
        while (j>=0 && a[j] > last)
            a[j+1] ← a[j]
            j ← j-1
        a [j+1] ← last
}
```

Online sorting is one that will work if elements to be sorted are provided 1 at a time with understanding that algo must keep sequence sorted as more & more elements are added. Insertion Sort is online.

Q 21, 22

Algo	Best	Avg	Worst	Worst Space	Inplace	Stable	Online
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	✗
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✗	✗
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	✓
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	✗	✓	✗
Quick	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(n)$	✗	✗	✗
Heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	✓	✗	✗

Q 23 Iterative Binary Search

```
int binarySeach (int *a, int l, int r, int key)
{
    while (l<=r)
        m = (l+r)/2
        if (a[m]==key)
            return m;
        if (a[m] < key)
            l <- m+1
        else
            r <- m-1

    return -1
}
```

Binary Search

T.C. = $O(\log n)$ Avg, worst

$O(1)$ Best

S.C. = $O(1)$

Q24 Binary Recursive Search

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