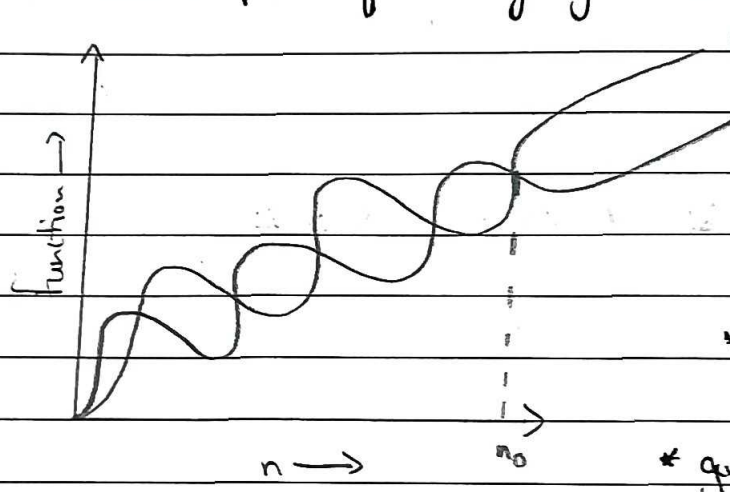
Date
Page No.

DAA assignment - 1

1) Asymptotic notation is used to describe the behaviour of a function as its input grows without bound. It is commonly used in the analysis of algorithms to describe their time and space complexity when the i/p is very large.

i) Big O(o): It represents the upper bound of a function. It is used to describe the worst case scenario for an algo. It represents the ~~upper~~ maximum time an algo. will take to complete for any given input.



$f(n) = O(g(n))$
iff $f(n) \leq c\,g(n)$
$\forall\ n \geq n_0$, some consta
$c > 0$

* g(n) is high upper bound
f(n)

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

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

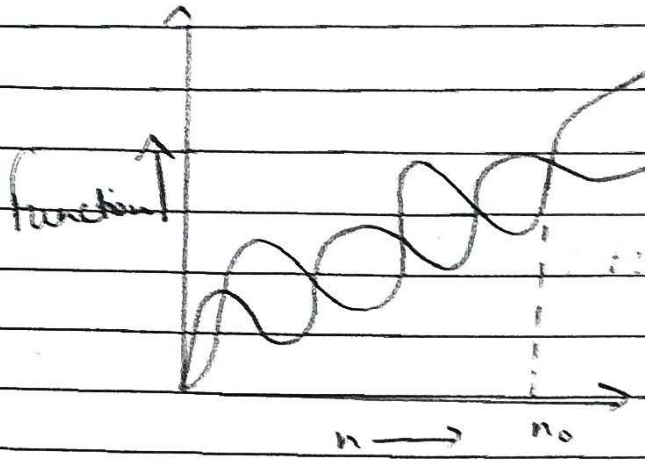
$2n + 3 \leq 5n^2$

$f(n) = O(n)$

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

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

ii) Omega (Ω) Notation :- It represents the lower bound of a function. It is used to describe the best case scenario for an algo. It provides a lower bound on the growth rate of function



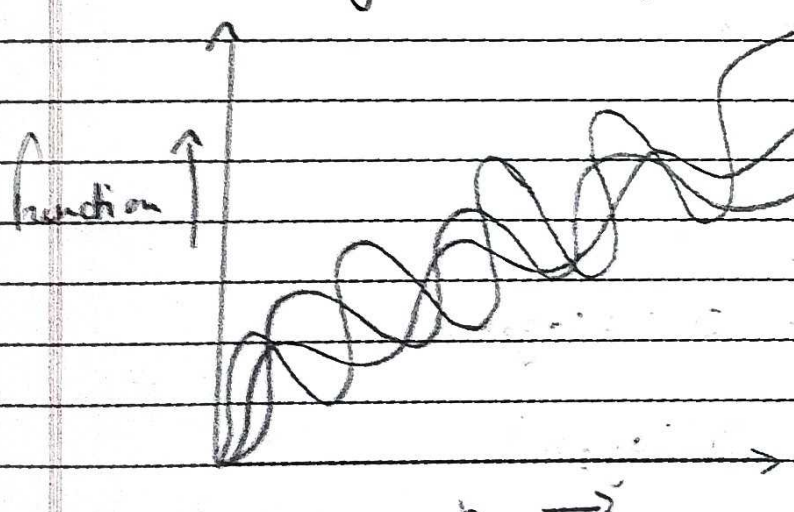
$f(n) = \Omega(g(n))$
iff $f(n) \geq c\,g(n)$
$\forall\ n \geq n_0$ and some constant $c > 0$
* $g(n)$ is tight lower bound of $f(n)$

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

$2n + 3 \geq 1 \times \log n \ ; \ \forall n \geq 1$

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

iii) Theta(θ) notation :- It represents both the upper bound and lower bounds of a function. It is used to describe the tight bounds of an algo. It provides an exact bound on the growth rate of the function. It tells exact time



$f(n) = \theta(g(n))$
iff $c_1(g(n)) \leq f_n \leq c_2 g(n)$
$\forall\ n > \max(n_2, n_1)$
& some constant $c_1 > 0$
$c_2 > 0$

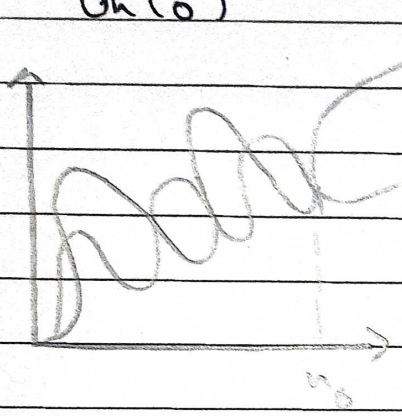
Ex: $f(n) = 2n+3$

$1 \times n \leq 2n+3 \leq 5 \times n$

$c_1 \; g(n) \qquad f(n) \qquad c_2 \; g(n)$

$f(n) = O(n)$

(iv) Small Oh (o)



$f(n) = o(g(n))$ iff $f(n) < cg(n)$ $\forall n > n_0 \; \forall c > 0$

$g(n)$ is upper bound of f_n

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

$2n+3 < 2n^2 + 3n^2$

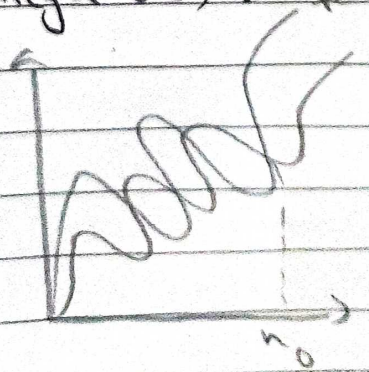
$2n+3 < 5n^2$

$f(n) = o(n)$

$g(n) = o(n^2)$

$f(n) < cg(n)$

(v) Small Omega (ω):



$f(n) = \omega(g(n))$ iff $f(n) > cg(n) \; \forall n > n_0 \; \forall c > 0$

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

$2n+3 > 1 \times \log n$

$f(n) > g(n)$

2> Time Complexity of :

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

1, 2, 4, 8 ..... K terms

K^{th} term = $a \cdot r^{K-1}$; $r = 2$

K^{th} term => $1 \cdot 2^{K-1}$

$n = 2^{K-1}$

$n = \frac{2^K}{2^1}$ => $2n = 2^K$

Taking $\log_2$ both side :

$\log_2 2n = \log_2 2^K$

$\log_2 2n = K$

$K = \log_2 2 + \log_2 n$

$K = 1 + \log_2 n$

Complexity = $O(1 + \log_2 n)$

=> $O(\log_2 n)$

3) $T(n) = 3T(n-1)$ — ①

$T(0) = 1$ — ②

Putting, $n = n-1$ in ①

$T(n-1) = 3T(n-2)$ — ③

Putting, $n = n-2$ in ①

$T(n-2) = 3T(n-3)$ — ④

Putting, $n = n-3$ in ①

$T(n-3) = 3T(n-4)$ — ⑤

Putting ⑤ in ④

$T(n-2) = 9T(n-4)$ — ⑥

Putting ⑥ in ③

$T(n-1) = 3^3 T(n-4)$ — ⑦

Putting ⑦ in ①

$T(n) = 3^4 T(n-4)$ — ⑧

for k terms,

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

$n-k = 0$

$\Rightarrow n = k$

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

$T(n) = 3^n$

Complexity $= O(3^n)$

Date
Page No.

4) $T(n) = 2T(n-1) - 1$ -①

$T(0) = 1$ -②

Putting $n = n-1$ in ①

$T(n-1) = 2T(n-2) - 1$ -③

Putting $n = n-2$ in ①

$T(n-2) = 2T(n-3) - 1$ -④

Putting $n = n-3$ in ①

$T(n-3) = 2T(n-4) - 1$ -⑤

Putting ⑤ in ④

$T(n-2) = 2 \cdot 2T(n-4) - 2 - 1$ -⑥

Putting ⑥ in ③

$T(n-1) = 2 \cdot 2 \cdot 2T(n-4) - 2 \cdot 2 - 2 - 1$ -⑦

Putting ⑦ in ①

$T(n) = 2^4 T(n-4) - 2^3 - 2^2 - 2^1 - 2^0$ -⑧

for K terms

$T(n) = 2^K T(n-K) - 2^{K-1} - 2^{K-2} - 2^{K-3} \ldots - 2^1 - 2^0$ -⑨

taking $n-K=0$

$\Rightarrow 2^K T(0) - 2^{K-1} - 2^{K-2} - 2^{K-3} \ldots - 2^1 - 2^0$

$= 2^K - (2^0 + 2^1 + 2^2 \ldots \ldots 2^{K-3} + 2^{K-2} + 2^{K-1})$

$= 2^K - \left(\frac{1-2^K}{1-2}\right) \Rightarrow 2^K + 1 - 2^K$

$= 1$

Complexity = $O(1)$

5)
```
int i=1, s=1;
while (s<=n)
{
    i++;
    s=s+i;
    print ("#"),
}
```

i = 1, 2, 3 - - - - -

s = 1, 3, 6, 10, 15, 21, 28 - - - k^{th} term

loop will run until $s <= n$

$$k^{th} \text{ term} = \frac{k*(k+1)}{2}$$

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

$$2n = k^2 + k$$

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

$$k^2 = 2n$$

$$k = \sqrt{n}$$

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

6>
```
void function (int n)
{
  int i, count = 0;
  for (i=1; i*i <= n; i++){
    count++;
  }
}
```

$1^2, 2^2, 3^2, 4^2, 5^2 \dots\dots K^2$

K^{th} term $= K * K$

K^{th} term $\leq n$

$K^2 = n$

$K = \sqrt{n}$

Complexity $= O(\sqrt{n})$

7>
```
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++;
      }
    }
  }
}
```

For Innermost loop

$k = 1$ to n, $k = k * 2$

1, 2, 4, 8, 16 ... k^{th} term

k^{th} term $= 2^{k-1}$

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

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

$\log_2 2 + \log_2 n = k$

$k = 1 + \log_2 n$

Complexity $= O(\log_2 n)$

For middle loop

$j = 1$ to n ; $j = j * 2$

Complexity $= O(\log_2 n)$

For first loop

$i = n/2$ to n ; $i = i++$

$n/2, \frac{n}{2}+1, \frac{n}{2}+2$ ... k^{th} term

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

$n = \frac{n}{2} + k \Rightarrow k = n - \frac{n}{2} \Rightarrow k = \frac{n}{2}$

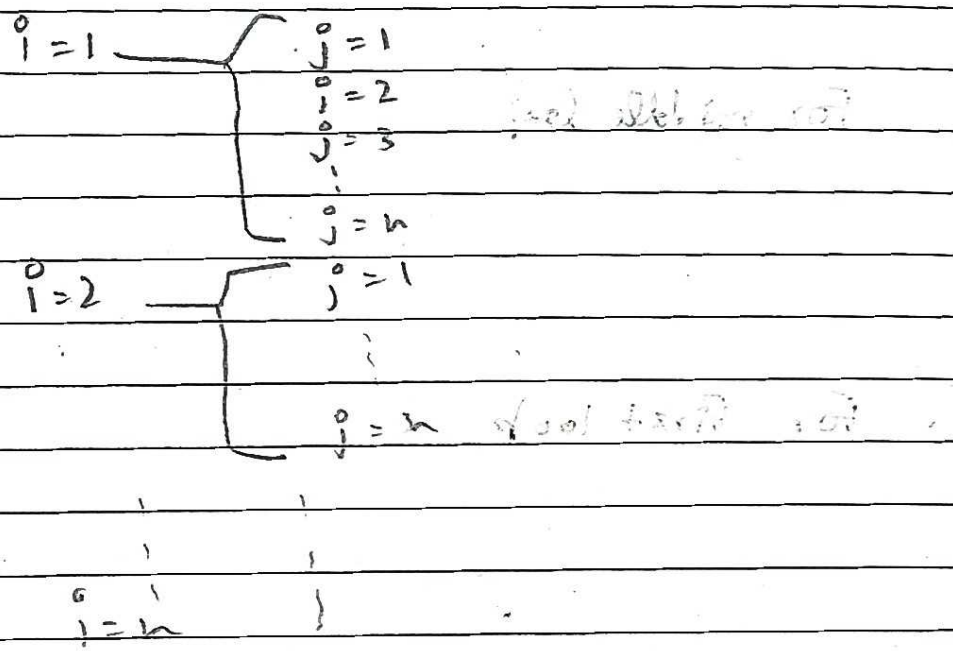
Complexity $= O(n)$

Total complexity = $\frac{n}{2} * \log_2 n * \log_2 n$

$O(n(\log_2 n)^2)$

8 >
```
function (int n)
{
  if (n==1)
    return;
  for (i to n){
    for (j to n){
        printf("*");
    }
  }
  function(n-3);
}
```

Complexity of nested for loop here = $O(n^2)$



for function(n-3)

n, n-3, n-6, n-9 ---- k^{th} term

n, n-1.3, n-2.3 ----- "

k^{th} term = $n-3k-3$

$1 = n-3k-3 \Rightarrow k = \frac{n-4}{3}$

Date
Page No.

Total Complexity = $\frac{n-4}{3} * n^2 = \frac{n^3 - 4n^2}{3}$

Complexity = $O(n^3)$

9) void function (int n)
{
for (i=1 to n) {
for (j=1, j<=n; j=j+i) {
printf("x");
}
}
}

Outer loop will run n times (i)

for (i=1), j will run n times
i = 2, j will " n/2 "
i = n, j will " n/n "

Inner loop will run = $\left(n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}\right)$

$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right)$

$= n \log n$

Complexity = $O(n \log n)$