DAA

Assignment - 1

1) Asymptotic notations are the mathematical notation used to describe the running time of an algo when the I/P tends towards a particular value or limiting value.
There are mainly 3 asymptotic notations:-

★ **Big-O notation:** It represent the upper bond of running time of an algo.

- This notation is called as upper bound of an algo or a worst case of algo.
- $O(g(n)) = \{ f(n) :$ There exist positive constraint c and no such that $0 \leq f(n) \leq g(n)$ for all $n > no$, where $c > 0$ and $n \geq no \}$
- eg: $f(n) = 3 \log n + 100$
  $g(n) = \log(n)$
  $3 \log n + 100 <= c \times \log(on)$
  $c = 150$ and $n > 2$ (undefined at $n = 1$)

★ **Big Omega (Ω) notation:** It represent the lower bound of the running time of an algo.

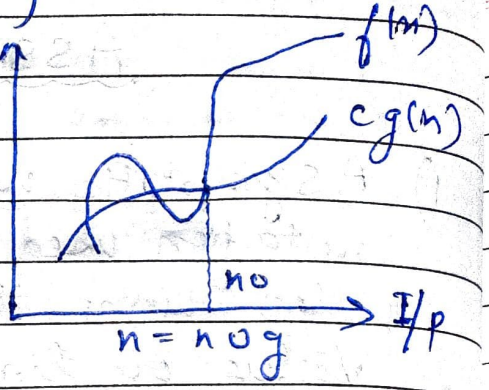
- This notation is known as lower bound of an algo, or best case of an algo.
- $\Omega(g(n)) = \{ f(n) :$ There exist positive constrains c and no such that $0 \leq cg(n) \leq f(n)$ $\forall n$, $n > no \}$
- eg: $f(n) = 3n + 2$
  $cg(n) < f(n)$

(c = constant, $g(n) = n$)

$cn \leq 3n+2$

$n(c-3) \leq 2$

$\Rightarrow n \leq \frac{2}{c-3}$



if we assume $c = 4$, then $no = 2$

$[c = 4, no = 2]$.

★ Theta(θ) notation:- It enclose the function from above and below. Since it represent upper and lower band.

- This is known as tight bounds of an algo, or an average case of algo.
- $\theta(g(n)) = f(n)$: There exist positive constant c_1, c_2, and n_0 such that:

$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n) \quad \forall \; n > n_0$.

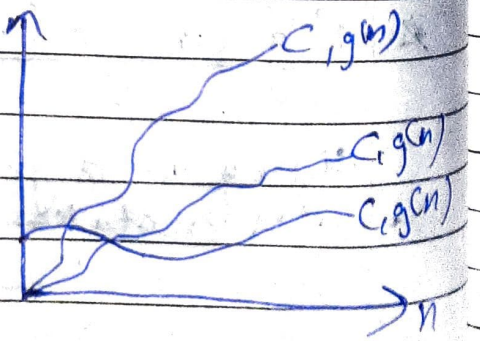
- eg: $f(n) = 5n^3 + 16n^2 + 3n + 3$

$5n^3 \leq 5n^3 + 16n^2 + 3n + 8 \leq (5+16+3+8)n^3$

$5n^3 \leq f(n) \leq 32n^3$

$[c_1 = 5; \; c_2 = 32; \; n_0 = 1]$

$f(n) \leftrightarrow \theta(n^3)$.



vibhu
PAGE No
DATE

2) $i = 2, 4, 8, 16 \dots$ k^{th} term $\dots n$

$G_n = ar^{n-1}$

$G_n = 1(2)^{k-1}$

$n = 2^{k-1}$

$\log_2 n = (k-1)\log_2 2$

$[k = \log_2 n + 1]$

$O(n) = \log n$.

3) $T(n) = 2\,3T(n-1)$ if $n > 0$

$T(n) = 3T(n-1)$ ← $T(n-1) = 3T(n-2)$

$T(n) = 3 \times 3T(n-2)$ ← $T(n-2) = 3T(n-3)$

$T(n) = 3 \times 3 \times 3T(n-3)$

$T(n) = 3^3 T(n-3)$ ← $T(n-3) = 3T(n-4)$

$T(n) = 3^3 \times 3T(n-4)$

$T(n) = 3^4 \times T(n-4)$

General form :

$T(n) = 3T(n-1)$ ---① $[T(0) = 1]$

$T(n-i) = T(0)$

$n - i = 0 \Rightarrow [n = i]$

Putting $n = i$ in ① ;

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

$T(n) = 3^n$

$[T(n) = O(3^n)]$.

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

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

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

← $T(n-2) = 2T(n-3) - 1$

vibhu
PAGE No
DATE

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

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

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

$T(n) = 2(2T(n-4)-1) - 2 - 2 - 1$

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

General form:

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

$T(n-i) = T(0)$

$n - i = 0$

$(n = i)$

$T(n) = 2^n T(0) - (1 + 2 + 2^2 + 2^3 + \cdots 2^{n-1})$

$[T(0) = 1]$

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

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

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

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

$[T(n) = O(2^n)]$

---

5) No of steps (K) | S | i

0	0	1
1	1	2
2	3	3
3	6	4
4	10	5
5	15	6
6	21	7
⋮	⋮	⋮
K	n	

vibhu
PAGE No
DATE

$T(n) = O(K)$

$i = 0, 1, 3, 6, 10 \dots n$

$S_n = 1 + 3 + 6 + 0 + 15 + \dots + n$

$S_n = 1 + 3 + 6 + 10 + \dots + (n-1) + n$

$0 = 1 + 2 + 3 + 4 + 5 \dots n$

$n = 1 + 2 + 3 + 4 + \dots$ K step

$n = \frac{K}{2}[2(1) + (K-1)1]$

$2n = K[2 + K - 1]$

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

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

$K + \frac{1}{2} = \sqrt{2n + \left(\frac{1}{2}\right)^2}$

$K = \sqrt{2n + (1/2)^2} = 1/2$

$T(n) = T(K)$

$T(n) = T\left(\sqrt{2n + (1/2)^2} - 1/2\right)$

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

---

6) Since, i is moving from 1 to $\sqrt{n}$ with linear growth so.

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

---

7) $O(n \log n \log n)$

$[O(n(\log n)^2]$.

---

vibhu
PAGE No
DATE

8) $T(n) = T(n-1) + n^2 \quad [T(n-1) = T(n-2) + (n-1)]$

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

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

General formula:

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

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

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

$n = i+1 \Rightarrow [n = 1 = i]$

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

$\cdots + (n-(n-1))^2$

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

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

$T(n) = \frac{n(n+1)(2n+1)}{6} \Rightarrow [T(n) = O(n^3)]$

---

9) $O(n\sqrt{n})$

---

10) If $c > 1$, then exponential c^n for outgrows any term, so the answer is: n^k is $O(c^n)$

---

11) $i = 0, 1, 3, 6, 10, 15 \cdots$

$j = 1, 2, 3, 4, 5, 6, \cdots$

So, i will go n till n and general formula for K^{th} term is $n = \frac{K(K+1)}{2}$

$\therefore TC = O(\sqrt{n})$

---

vibhu
PAGE No
DATE

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

$T(n-2) \approx T(n-1)$

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

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

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

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

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

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

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

General form:

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

$n - i = 0$

$n = i$

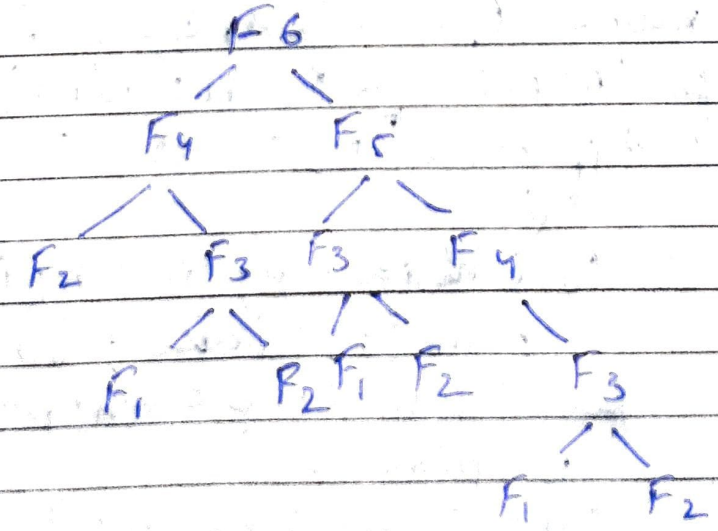
$T(n) = 2^n T(0) + (2 + 2^1 + 2^2 \cdots 2^{n-1}) C$

$T(n) = 2^n (1) + 2^0 \frac{(2^{n-1} - 1)}{2-1} C$

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

$[T(n) = O(2^n)]$

Fig →



The max depth is proportional to N, hence the space com of Fibonacci recursive is O<N.

13) i) $n \log n$

```
void func()
{int i, j;
  for (i=1; i<=n; i++)
    { for(j=0; j<=n; j=j*2)
      printf("#");
      printf("\n"); }}
```

ii) n^3

```
Void fun(int n)
 { int i, j, k;
   for(i=0; i<=n; i++)
    { for(j=0; j<=n; j++)
      { for(k=0; k<=n; k++)
        printf("#"); }}}
```

iii) $\log(\log n)$

```
Void sieve of Eratosthenes (int n)
 { bool prime[n+1];
   memset(prime, true, sizeof(prime));
   for(int p=2; p*p<=n; p++)
    { if(prime[p]==true)
      {
       for(int i=p*p; i<=n; i+=p)
         prime[i]=false; }}
   for(int p=2; p<=n; p++)
     if(prime[p])
     cout<<p<<endl; }
```

vibhu
PAGE NO
DATE

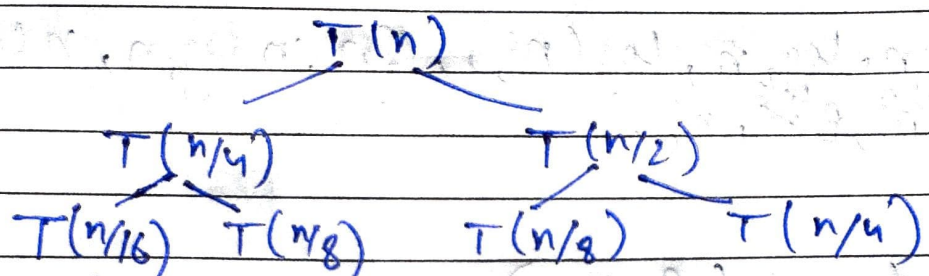
14) $T(n) = T(n/4) + T(n/2) + cn^2$

$T(1) = c$

$n = n/2$

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

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



$T(n) = c\left[n^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \cdots\right]$

$T(n) = n^2 c\left[1 + \frac{5}{16} + \frac{5^2}{16^2} + \cdots\right]$

$[T(n) = O(n^2)]$

15) For $i = 1$; inner loop is executed n times.
For $i = 2$; inner loop is executed $n/2$ times.
For $i = 3$; inner loop is executed $n/3$ times.
For $i = n$; inner loop is executed n/n times.

Total time $= n + n/2 + n/3 + \cdots n/n$

$= n(1 + 1/2 + 1/3 + \cdots 1/n)$

$= n \log n.$

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

16) $O(\log(\log n))$

18) a) $100, \log\log n, \log n, \sqrt{n}, n, n\log n, n^2, 2^n, 2^{2n}, 4^n, n!$

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

c) $96, \log n, \log_2 n, \log(n!), \sqrt{n}, n\log_8 n, n\log_2 n, 8n^2, \pi n^3, 8^{2n}, n!$

---

19) Linear Search (A, Key)

```
comp ← 0, t ← 0
for i = 1 to A length
    comp ← comp + 1
    if A[i] == Key
        print "Element found"
        f = 1
if f == 0
    print "Element not found"
    print comp.
```

---

20) Iterative method of Iteration Sort →

```
for j - 2 to A.length
        Key = A[j]
        i = j - 1
while i > 0 and A[i] > Key
    A[i+1] = A[i]
    i = i - 1
    A[i+1] = Key
```

Recursive Method →

```
Insertion Sort (A, n)
    if n ≤ 1
    return
Insertion Sort (A, n-1)
    Key = [n-1];
    j = n-2;
while j <= 0 and A[j] > Key
    A[j+1] = A[j]
    j = j - 1
A[j+1] = Key.
```

Insertion sort considers one I/p element per iteration and produces a partial sol^n without considering future elements. That's why it is called online sorting.

Other sorting algo that have been taught:

- Bubble sort
- Merge sort
- Selection sort
- Heap sort
- Quick sort
- Counting sort.

21)

	Best case	Avg case	Worst case.
Bubble sort	$\Omega(N)$	$\Theta(N^2)$	(N^2)
Selection	$\Omega(N^2)$	$\Theta(N^2)$	(N^2).
Insertion	$\Omega(N)$	$\Theta(N^2)$	(N^2)
Merge	$\Omega(N \log N)$	$\Theta(N \log N)$	$(N \log N)$
Heap	$\Omega(N \log N)$	$\Theta(N \log N)$	$(N \log N)$
Quick	$\Omega(N \log N)$	$\Theta(N \log N)$	(N^2)
Counting	$\Omega(N+K)$	$\Theta(N+K)$	$O(N+K)$

22)

	Inplace	Stable	Online
Bubble	Yes	Yes	Yes
Insertion	Yes	Yes	Yes
Selection	Yes	NO	Yes
Merge	No	Yes	Yes
Quick	Yes	NO	Yes
Heap	Yes	No	Yes
Count	No	Yes	Yes

23) • Linear Seach (A ; Key)

```
found ← 0
fa (i=1' toN)
    if A[i] == Key
        found ← 1
    print "Element found"
    break
If found == 0
print "Elenet not found"
```

→ Time complexity - O(n)
→ Space complexity - O(1)

• Binary Search(A, beg , end, Key)

```
while beg ≤ end
    mid = beg + (end - beg)/2
    if mid == key
    return mid                  { Iterative
    if A[mid] ≤ Key
        beg = mid + 1
    if A[mid] > Key
        end = mid - 1
```

return -1

→ Time complexity - O(log n)

→ Space complexity - O(1)

- Binary search (A, beg, end, key)

```
if end > beg
    mid = (beg + end)/2
    if A[mid] == item
    return mid + 1
    else if A[mid] < item
return binary-search (A, mid+1, end, key)
 else
return binary-search (A, beg, mid-1, end)
return -1
```

→ Time complexity - O(log n)

→ Space complexity - O(1)

---

24) $T(n) = T(\frac{n}{2}) + c$.