DAA assignment

Mayank Joshi ①
1961101, BTech, CSE V-C
DAA TCS-505

Ans 1→ Asymptotic notations:-

→ The word 'asymptotic' means approaching a value or curve arbitrarily closely (i.e. as some sort of limit is taken)

→ Asymptotic analysis is a technique of representing limiting behavior. It can be used to analyze the performance of an algorithm for some large input data set.

→ Asymptotic notations are used to write fastest and slowest possible running time for an algorithm. These are also referred to as 'best case' and 'worst case' scenarious respectively.

→ In asymptotic notations, we derive the complexity concerning the size of the input.

→ These notations are importatnt because without expanding the cost of running the algorithm, we can estimate the complexity of algorithm.

→ They give simple characteristics of an algorithm's efficiency.

→ They allow the comparisons of the performances of various algorithms.

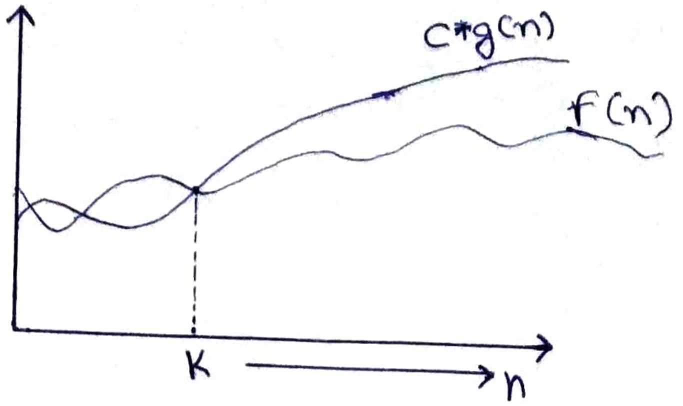
→ Asymptotic notation is a way of comparing function that ignores constant factors and small input sizes. Three notations are used to calculate the running time complexity of an algorithm:

1- Big 'O' notation: This is the formal method of expressing the upperbound of an algorithm's running time. It is the measure of the longest amount of time. The function $f(n) = O(g(n))$ if and only if exist positive constant c and such that.

$f(n) \le K.g(n)$ $f(n) \le K.g(n)$ for ~~n~~ $n_0 < n_0 < n$

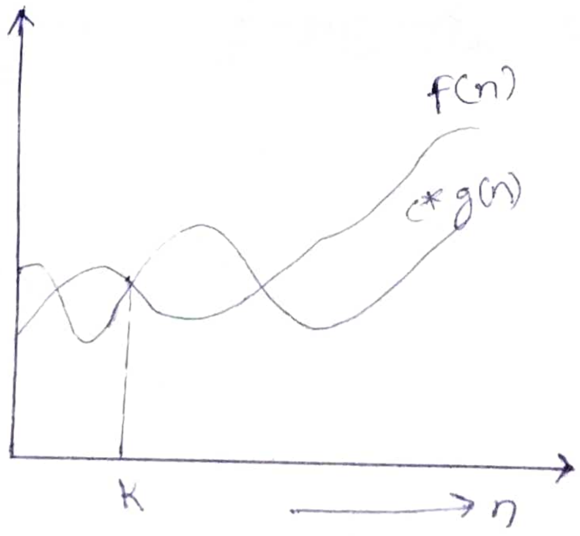
Hence function $g(n)$ is an upper bound for function $f(n)$, as $g(n)$ grows faster than $f(n)$, Or

The function $f(n) = O(g(n))$, if and only if there exist a positive constant C and K such that $f(n) \le C * g(n)$ for all $n, n \ge K$



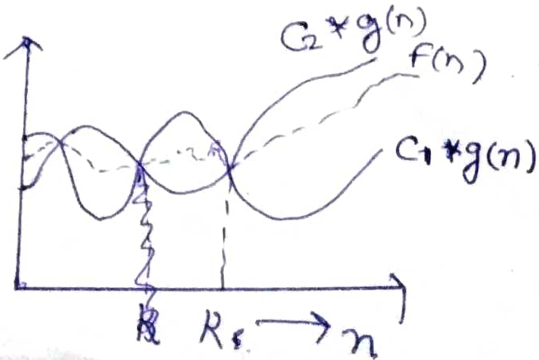
2- Big-omega (Ω) notation: (Asymptotic Lower bound):

The function $f(n) = \Omega(g(n))$, if and only if there exists a positive constant C and K such that $f(n) \ge C * g(n)$ for all $n, n \ge K$.



3- Big-Theta notation (Θ): (Asymptotic tight point):

The function $f(n) = \Theta(g(n))$, if and only if there exist a positive constant C1, C2 and K such that $C1 * g(n) \le f(n) \le C2 * g(n)$ for all $n, n \ge K$.



Ans 2.

$$\text{for}(i=1 \text{ to } n)\ \{i=i*2\}\ O(1) \Rightarrow \text{for}(i=1;\ i<=n;)\ \{\ i=i*2;\ O(1)\ \}$$

$O(1)$ $\quad$ $O(1)$ $\quad$ $O(1)$

$i = 1, 2, 4, 8, 16 \ldots n$

• let the loop runs K times

~~Kth term = a+(K-1)d~~

~~Sum of Kth term of G.P.~~

~~$S = \frac{1(2^K-1)}{2-1} = 2^K-1$~~

Kth term of g.p.

$n = 1.(2)^{K-1}$

$\log_2 n = K-1$

$K = 1+\log_2 n$

~~So $S = 2^K - 1 = 2^{1+\log_2 n} - 1 = 2.2^{\log_2 n} - 1 = 2n-1$,~~

~~Total time taken $= O(1) + O(2n*1) + O((2n-1)1)$~~

~~$= O(1)+O(2n)+O(2n-1)$~~

~~$= O(n)$~~

loop will run K $(=1+\log_2 n)$ times,

So, Total time taken $= c_1 + c_2*(K+1) + c_3*(K)$

$= c_1 + c_2*(1+\log_2 n+1) + c_3*(1+\log_2 n)$

$= c_1 + 2c_2 + c_2\log_2 n + c_3 + c_3\log_2 n$

$= c_1+2c_2+c_3 + (c_2+c_3)\log_2 n$

$= O(1) + O((c_2+c_3)\log_2 n)$

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

$= O(\log_2(n))$

Ans3. $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

$\cancel{O(0)=1}$ $T(0) = 1$

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

put $n=1$

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

put $n=2$

$T(2) = 3T(1) = 9$

put $n=3$

$T(3) = 3T(2) = 27$

~~put n=m~~

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

$\Rightarrow T(n) = 3^n$

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

Ans 4) $T(n) = \begin{cases} 2T(n-1)-1 & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

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

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

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

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

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

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

$T(n) = 16T(n-4)-8-4-2-1$

$T(n) = 2^k T(n-k) - 1 - 2 - 4 - 8 \dots - 2^{k-1}$

for base condition,

$T(n-k) = T(0) = 1$

$n-k=0$

$n=k$

$T(n) = 2^n(1) - [1+2+4+8+\dots 2^{n-1}]$

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

$= 1$

~~$T(n) = O(2^n)$~~

$T(n) = O(1)$

Ans 5→ int i=1, s=1; O(1) O(1)

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

i = 1 2 3 4 5 6 7 8 .... K

S = 1 3 6 10 15 21 28 36 .... n

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

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

$2n+\frac{1}{4} = \left(K+\frac{1}{2}\right)^2$

$\frac{1}{2}\sqrt{8n+1} - \frac{1}{2} = K$

$K = \frac{1}{2}(\sqrt{8n+1}-1)$

loop will run $\left(\frac{1}{2}\sqrt{8n+1}-1\right)$ times

total time complexity $= O(1)+O(1)+O\left(1.\frac{1}{2}\sqrt{8n+1}-1\right)+O\left(1.\frac{1}{2}\sqrt{8n+1}-1\right)+O\left(1.\left(\frac{1}{2}\sqrt{8n+1}-1\right)\right)+O\left(1.\left(\frac{1}{2}\sqrt{8n+1}-1\right)\right)$

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

Ans 6→ void function (int n) {

```
  int i, count=0;  O(1) O(1)
  ~~for (i=n; ...) {~~  O(1) O(1)
  for (i=1; i*i<=n; i++)  O(1)
      count++;   O(1)
```

here, for

i	n
1	1
1	2
1	3
2	4
K	$\sqrt{n}$

loop will run $\sqrt{n}$ times

Time complexity $= O(1)+O(1)+O(1).O(\sqrt{n})+O(1)+O(1).O(\sqrt{n})$

$= O(\sqrt{n})$

6

Ans 7 → void Function(int n){

```
    int i, j, K, count=0;        O(1)
    for(i=n/2 ; i<=n ; i++)
        for(j=1 ; j<=n ; j=j*2)
            for(K=1; K<=n; K=K*2)
                count++ ;         O(1)
```

~~Outer~~ Outer loop will have O(n) time comlexity,
inner loop will have O($\log_2 n$) time complexity
inner most loop will have O($\log_2 n$) time complexity.
So final time complexity $= O(n)\,O(\log_2 n)\,O(\log_2 n).O(1) + O(1)$
$= O(n(\lg n)^2)$

Ans 8 → function(int n) {

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

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

~~T(1) = 1 T(1)=T(2)=T(3)=T(4)=1~~
~~T(2) =~~
~~T(1) = 1+1 = 2 | T(2) = T(1)+2^2 = 6 | T(3) =~~

~~T(1) = T(-2) + 1^2 = 2~~
~~T(2) = T(-1) + 2^2 = 5~~
~~T(3) = T(0) + 3^2 = 10~~
~~T(4) = T(1) + 4^2 = 18~~
~~T(5) = T(2) + 5^2 = 30~~
~~T(6) = T(3) + 6^2 = 46~~

~~T(1) = 1, T(2) = 5, T(3) = 10, T(4) = 17~~
~~T(5) = 27, T(6) = 46 ...~~

~~$T(n) = T(n-3) + n^2$~~
~~here $T(n-3) = T(n-6) + (n-3)^2$~~
~~$T(n) = T(n-6) + (n-3)^2 + n^2$~~
~~$T(n) = T(n-9) + (n-6)^2 + (n-3)^2 + n^2$~~

Ans $\cancel{T(n-3)=}$

$T(1)=1$, $\cancel{T(n>3)=1}$

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

$T(7)=T(7-3)+7^2 = 1+4^2+7^2$

$T(10)=T(10-3)+10^2 = 1+4^2+7^2$

$\vdots$

$T(n) = 1+4^2+7^2+\dots+(3n-2)^2$

$T(n) = \sum_{i=1}^{n}(3i-2)^2 = \sum 9i^2 - \sum 6i + \sum 4$

$= 9\sum i^2 - 6\sum i + 4\sum 1$

$= 9\,\frac{n(n+1)(2n+1)}{6} - 6\,\frac{n(n+1)}{2} + 4n$

$\approx n^3$

So, time complexity $= O(n^3)$

Ans 9 → void function (int n)

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

i	j	
1	1, 2, 3, ... n	(n times)
2	1, 3, 5, 7 ... n	($\approx \frac{n}{2}$ times)
3	1, 4, 7, 10, ... n	($\approx \frac{n}{3}$ times)
⋮		
n	... 1, n	($\approx \frac{n}{n} = 1$ time)

total no. of times the loop execute is,

$= n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \dots \; 1$

$= n\left(1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\dots\frac{1}{n}\right) = n\log_2 n$

Time complexity $= O(n\log n)$

Ans 10→ $n^k = O(a^n)$ ~~(How?)~~

For all values of $n \geq 1$, $k > 1$, $a > 1$

Ans 11→
```
void fun(int n) {
    int j=1, i=0;
    while(i<n){
        i= i+j;
        j++ ;}}
```

$i = 0,\ 0+1,\ 0+1+2,\ 0+1+2+3,\ \ldots\ldots,\ \frac{k(k+1)}{2}$

let the loop run k times so,

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

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

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

$$\left(k - \frac{1}{2}\right)^2 = 2n + \frac{1}{4}$$

$$k - \frac{1}{2} = \sqrt{2n + \frac{1}{4}}$$

$$k = \sqrt{2n + \frac{1}{4}} + \frac{1}{2}$$

$$k \approx \sqrt{n}$$

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

Ans 12→ $T(n) = \begin{cases} T(n-1) + T(n-2) + C & \text{if } n \geq 2 \\ 1 & \text{, otherwise} \end{cases}$

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

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

$T(3) = (1+1)+1 = 3$

$T(4) = (1+1+1)+(1+1) = 5$

$T(5) = (1+1+1+1+1)+(1+1+1) = 8$

$T(6) = (1+1+1+1+1+1+1+1)+(1+1+1+1+1) = 13$

$T(n) \approx 2T(n-2) + C$ {since for larger $T(n-1) \approx T(n-2)$

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

here $T(n-2) = 2T(n-4) + C$

$T(n) = 2^2 T(n-4) + 2C + C$

$T(n) = 2^3 T(n-6) + 2^2 C + 2C + C$

$T(n) = 2^k T(n-2k) + C(1 + 2 + 2^2 + \ldots + 2^{k-1})$

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

base condition

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

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

$$T(n) \approx 2^{\frac{n}{2}}$$

Time complexity $\approx O(2^n)$.

Ans 13 → for $n(\log n)$

recurrence relation, $T(n) = aT(\frac{n}{b}) + f(n)$

for $n\log n$, $n^{\log_b a} = n = f(n)$

so $a = b = 2$, since $b > 1$

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

```
int called_function (int n, int arr[])
{ if (n==1)
      return 0;
  int i, max = MIN_INT, mxind = -1;
  for (i=0; i<n; i++)
  {
      if (arr[i] > max)
      { max = arr[i];
        mxind = i;
      }
  }
  called_function(n/2, arr);
  return called_function(n/2, arr);
}
```

for $O(n^3)$,

```
for (i=0; i<n; i++)
{
    for (j=0; j<n; j++)
    {
        for (k=0; k<n; k++)
        {
            printf("sum of %d,%d,%d=%d", i,j,k, i+j+k);
} } }
```

For $\log(\log n)$

```
for (i=0; i<=fun(log(n)); i++)
    printf("*");

float fun(float a)
    return log(a);
```

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

~~$T(n)$~~

~~$T(\frac{n}{4})$ $T(\frac{n}{2})$ $= cn^2$~~

~~$T(\frac{n}{16})$ $T(\frac{n}{8})$ $T(\frac{n}{8})$ $T(\frac{n}{4})$ $= 2(\frac{cn^2}{16}) + 2(\frac{cn^2}{4}) = \frac{cn^2}{8} + \frac{cn^2}{2}$~~

~~$= \frac{5cn^2}{8}$~~

19→ $T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + cn^2$

cn^2 — cn^2

$\frac{c.n^2}{16}$ $\frac{cn^2}{4}$ — $\frac{5cn^2}{16}$

$\frac{cn^2}{256}$ $\frac{cn^2}{64}$ $\frac{cn^2}{64}$ $\frac{cn^2}{16}$ — $\frac{25cn^2}{256}$

$\vdots$

$\log_4 n$ times

11

$\Rightarrow T(n) = cn^2\left(\frac{5}{16}\right)^0 + cn^2\left(\frac{5}{16}\right)^1 + \dots + cn^2\left(\frac{5}{16}\right)^{\log_4 n - 1}$

$= cn^2\left[\frac{1-\left(\frac{5}{16}\right)^{\log_4 n - 1}}{1-\frac{5}{16}}\right]$

$= \frac{15}{11}cn^2\left[1-\left(\frac{5}{16}\right)^{\log_4 n} \times \frac{16}{5}\right]$

$= \frac{15}{11}cn^2\left[1-\frac{5^{\log_4 n}}{2^{4\log_4 n}} \times \frac{16}{5}\right]$ | $n^{\log_4 16}$ n^4

$= \frac{15}{11}cn^2\left[\frac{2^n - 3n^{\log_4 5}}{2^n}\right]$

$= \frac{15}{11}\left(cn^2 - \frac{cn^2 n^{\log_4 5}}{2^n}\right)$

here $cn^2 > \frac{cn^2 n^{\log_4 5}}{2^n}$

$\approx O(n^2)$

15) fun(n)

```
for ( i=1; i<=n; i++ )
{    for (j=1 ; j<n ; j+=i ){
        //O(1)
                          }
}
```

$\sum_{i=1}^{n} i = 1+2+3+4+\dots+n = \frac{n(n+1)}{2} \approx O(n^2)$

16) for (i=2 ; i<=n ; i = pow(i,k))

i	j
1	1 ... n
2	1 3 5 7 9 ... $\frac{n}{2}$ term
3	... $\frac{n}{3}$
4	... $\frac{n}{4}$
⋮	⋮
k	... $\frac{n}{k}$ times

Total time $= n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} \dots = n\left(1+\frac{1}{2}+\frac{1}{3}+\dots\right)$
$= O(n \log n)$

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

i
2
2^k
2^{2k}
2^{3k}
2^{4k}
⋮
2^{jk}

$2^{jk} = n$

$jk = \log_2 n$

$j = \frac{\log n}{k}$

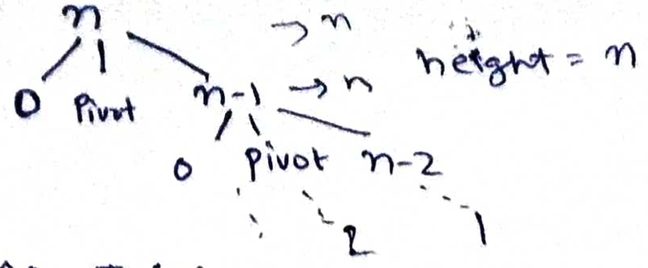
loop will run $\frac{\log n}{k}$ times

$T(n) = 1+1+1+1 \ldots \frac{\log n}{k}$ times

$\approx O\left(\frac{\log n}{k}\right) = O(\log n)$

17 → For sorted array, worst case of quick sort when
$T(n) = T(n-1) + n$

let us assume an array 1 2 3 4 5
let last element is pivot element 1 2 3 4 5
we will compare each element with pivot element
1, 2, 3, 4, 5, after comparison pivot element
will be placed at the last position, but comparision
with (n-1) element leads to time complexity of O(n)
The array will be divided in to subarray of (n-1)
size so recurrence tree is.



so, $T(n) = T(n-1) + n$

$\approx O(n^2)$

Difference in ~~extreme~~ height $= n - 0 = n$

So, the quicksort has time complexity of $O(n^2)$ when the array is sorted.

18 →

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

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

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

19 →
```
int LS ( int arr, int n, int key)
{
    int i;
    for (i=0; i<n; i++)
    {   if (arr[i]==key)
            return i;
    return -1.
```

20 →
```
void IIS (arr, n)
    for i → 1 to n
        t = arr[i]
        j = i-1
        while j > -1 and t < arr[j]
            arr [j+1] = arr [j]
            j = j-1
        arr [j+1] = t
```

```
void RIS (arr, n)
    if n == 1 :
        return
    RIS (arr, n-1)
    t = arr[n-1]
    j = n-2
    while j >= 0 and t < arr[j]
        arr [j+1] = arr [j]
        j = j-1
    arr [j+1] = t
```

Insertion sort is called to be online sorting ~~algom~~ because elements are sorted one by one and it will work if the elements to be sorted are provided one at a time with the understanding that the algorithm must keep the sequence sorted as more & more elements are added in. Other algorithms are ~~offline~~ not online sorting algorithm.

Ans21:-

	Time complexity Best	Average	Worst	Space complexity
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$ ~~$O(\log n)$~~
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$
Counting Sort	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(k)$

Ans 22)

	Inplace	Stable	Online
Bubble Sort	✓	✓	✗
Selection Sort	✓	✗	✗
~~Heap~~ Insertion Sort	✓	✓	✓
Merge Sort	✗	✓	✗
Quick Sort	✓	✗	✗
Heap Sort	✓	✗	✗
Counting Sort	✗	✓	✗

Ans 23) BSearch (arr , key, f , l)

```
m = (f + l)/2
if arr[m] == key:
        return m
else if arr[m] > key
        return Bsearch(arr, key, f, m-1)
else if arr[m] < key
        return Bsearch (arr, key, m+1, l)
else
        return -1
```

Recursive	Linear search	binary search (R)	binary search (I)
Space complexity	O(1)	O(n)	O(1)
Time complexity	O(n)	O(log n)	O(log n)

~~Iterative~~
~~Space complexity~~
~~Time complexity~~

Ans 24) Recurrence relation for binary search:

$$T(n) = \begin{cases} 1 & , \text{if } n = 1 \\ T(\frac{n}{2})+1 & , \text{otherwise} \end{cases}$$