Name: Govind Tripathi
Section: E
Class Roll No: 26
University Roll No: 2014396
Subject: Design and Analysis of Algorithms
Subject Code: TCS-505

# ASSIGNMENT-1 (DAA)

Ans 1: These notations are used to tell the complexity of an algorithm when the input is very large.

Asymptomatic → towards infinity

Asymptomatic notations: These are mathematical tools to represent the time complexity of Algorithms for asymptomatic analysis.

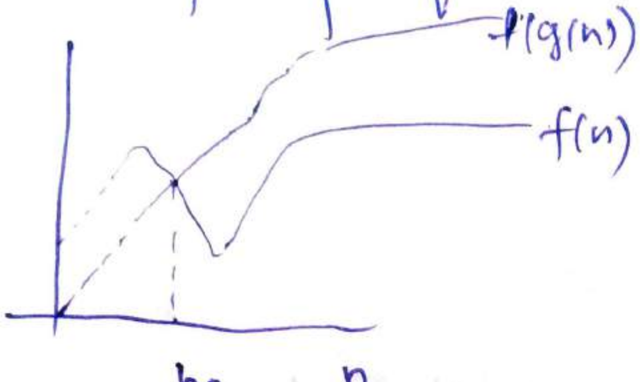
3-types of Asymptomatic notations:-

1) Big-O-Notation: This defines an upper bound of an algorithm it bounds functions only from above.

for ex: In case of insertion sort, it takes linear time in best case and quadratic time in worst case. So we can say that time complexity of insertion sort is $(O(n^2))$

Govind

It is useful only when we have upper bound on time complexity of an algorithm. (2)



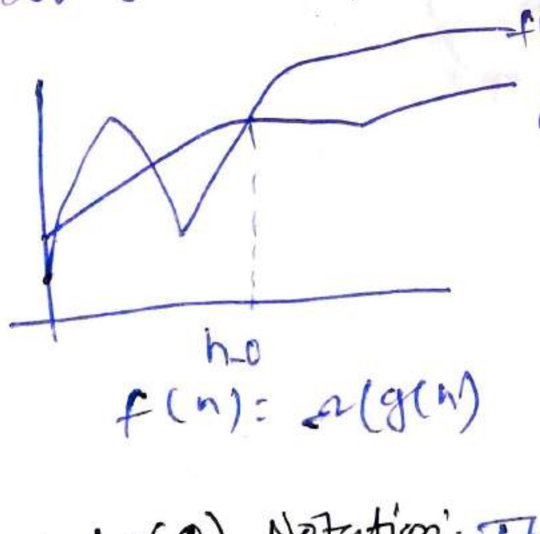

$f(n) = O(g(n))$

$O(g(n)) = \{f(n)$: there exist positive constant $c$ and $n_0$ such that $0 <= f(n) <= c \cdot g(n)$ for all $n >= n_0\}$.

(2) Big Omega ($\Omega$) Notation: This defines lower bound of an algorithm. This notation is the least used notation among all.

Ex. Time complexity of insertion sort can be written as $\Omega(n)$.

This notation can be useful when we have lower bound on time complexity of algorithm.




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

$\Omega(g(n)) = \{f(n)$: There exists +ve constant $c$ and $n_0$ such that $0 <= c \cdot g(n) <= f(n)\}$.

(3) Theta ($\Theta$) Notation: This notation bounds a function from above and below, so, it defines exact asymptomatic behaviour.

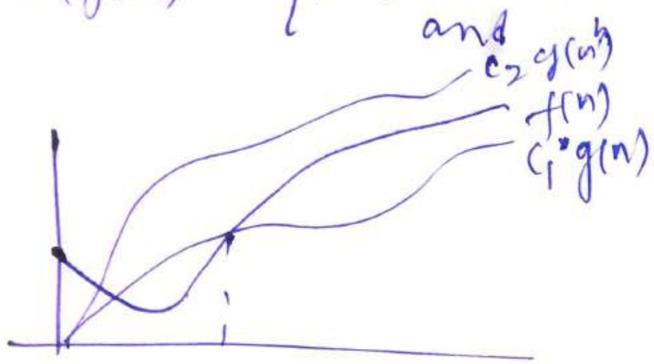
Govind

A simple way to get $\Theta$ notation of an expression is to drop low order terms and ignore leading constants.

eg. $3n^3 + 6n^2 + 6000 = \Theta(n^3)$

Dropping lower order terms is always good because there will always a no. (n) after which $\Theta(n^3)$ has higher values than $\Theta(n^2)$. Irrespective of constants involved.

$\Theta(g(n)) = \{f(n)$: There exist +ve constant $c_1, c_2$ and $n_0$ such that $0 <= c_1 * g(n) \leq f(n) = c_2 * g(n)$ for all $n \geq n_0$.




Ans 2. for ( i=1 to n ) { i = i*2

1, 2, 4, 8, 16 . . . n

its a G.P so,

$a = 1, r = 2$

So, $f(n) = \dfrac{a(r^{terms} - 1)}{r-1}$

$T_k = ar^{k-1}$

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

$n = \dfrac{2^k}{2}$

$2^k = 2n$

Taking log on both sides

$k = \log_2(2n)$

$k = \log_2(2) + \log_2(n)$

$k = \log_2(n) + 1$

Govind.

so, time complexity $= O(\log_2(n)+1) = O(\log_2(n))$
$= O(\log(n))$

Ans 3

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

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

put $n = n-1$ in eq (1)

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

put eq (1) in (2)

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

$T(n) = 3(3T(n-2)) \rightarrow$ (3)

Now put $n = n-2$ in eq (1)

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

put value of $T(n-2)$ in - eq (3)

$T(n) = 9(3T(n-3))$ - (5)

$T(n) = 3^k T(n-k)$ - (6)

$T(1) = 1$

$n-k = 1$

$k = n-1$

$T(n) = 3^{n-1} T(n-(n-1))$ - (7)

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

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

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

Ans 4:

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

$T(1) = 1$

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

$T(n-1) = 2T(n-2) - 1$ — ②

(5)

put $T(n-1)$ in eq ①

$T(n) = 2(2T(n-2)-1) - 1$ — ③

put $n = n-2$ in eq ①

$T(n-2) = 2T(n-3) - 1$ ④

put the value of $T(n-2)$ in eq ③

$T(n) - 4(2T(n-3) - 1) \cdot 2 - 1$ — ⑧

$T(n) = 8T(n-3) - 4 - 2 - 1$ — ⑥

$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - \dots 2^2 - 2^1 - 2^0$ — ⑥

Now $n-k = 1$, $k = n-1$, put $k$ in eq. ⑥

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

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

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

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

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

Ans 5 int i = 1, s = 1, while (s ≤ n) { i++; s = s+i;
printf ("#"); }

$\frac{k(k+1)}{2} \Rightarrow k^{th}$ term formula

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

$\boxed{T.C = O\sqrt{n}}$

Ans 6. void function (int n)

{ int i, count = 0;

for( i = 1; i*i <= n; i++)

{ count++;

Contd.

1, 2, 4, 8, 16 . . . n

it a GP, $a=1, r=2$

$t_k = a(r^{term-1}) = 1(2^{k-1}) \Rightarrow n = 2^{k-1} \Rightarrow 2^k = 2n$

$k = \log_2(n) + \log_2(2) \Rightarrow k = \log_2 n + 1$

$T.C = O(\log_2(n)+1)$

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

Ans 7:
```
Void function(int n)
{int i, j, k, count = 0;
for (i = n; i<=n; i++)
  {for(j=1; j<=n; j++)
   {for(k=1 k<=n; k++)
     { count++; }}}}
```

For the first loop Time complexity will be $O(n)$

For second it will be $O(\log(n))$

for last it will $O(\log(n))$

$T.\ complexity = O(n) * O\log(n) * O(\log(n))$

$= O(n \log^2 n)$

Ans 8.
```
function(int n) { if (n==1){return;}
for( i=1; i<=n; i++) {for (j=1; j<=n; j++)
                       {printf("#");
                        }} } for(  ) }
```

For first loop Time complexity will be $O(n)$

" second " " " " be $O(n)$

" last " " " " be $O(n)$

so, total T.C: $O(n^2)$.

Ans.9 void function (int n)
```
{ for (i:=1 to n)
  { for( j=1 ; j<=n; j+i)
    { printf ("*"); }}
```

first loop = $O(n)$
second loop = $O(n)$ = $O(n^2)$

Ans 10: for the functions $n^k$ and $a^n$, what is the asymptomatic relationship b/w these functions? Assume $k>=1$ and $a>1$ are constants find out the value of c and no for which relation holds $f(n) = n^k$ ; $g(n) = a^n$

g(n) is tight upper bound of f(n)

$f(n) = \Theta(g(n))$ , $n^k = O(a^n)$

iff $f(n) \leq c \cdot g(n)$

$n^k \leq c \cdot a^n \quad \forall \; n > n_0$ and $c > 0$

$f(n) = O(g(n))$ , $n^k = O(a^n)$

Ans 11 find Time complexity

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

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

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

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

Govind.

Ans 12 Space complexity :-

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

$fib(n-1) \to T(n-1)$, $fib(n-2) \to T(n-2)$

solving this using tree method

n

n-1    n-2

n-2   n-3   n-3    n-4 ..... $2^n$

$T(n) = 1+2+4 \cdots + 2^n$ is a GP.

$a = 1,\ r = 2$

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

$T(n) = O(2^{n+1}) = O(2^n \cdot 2) = O(2^n)$

Since maximum depth is proportional to n

so, space complexity will be $O(n)$.

Ans 13 (a)

```
for (int i=1; i<=n; i*=2)   → O log(n)
{ for (int j=1 .. to n)     → O(n)
  { sum += j;
  } }
```

T.C. - $O(n \log(n))$

Ans 13 (b)

```
for ( int i=1; i<=n ; i++)          → O(n)
{ for (int j=1; j<=n; j++)          → O(n)
  { for (int k=1; k<=n; k++)        → O(n)
    { sum+=k; }}}
```

$= O(n^3)$

Contd

(c) for(int i = 2 ; i<=n; i=pow(i,i)){
{ for(int j = n; j>= i ; j = fun(i))
{ .
} }

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

(14) Solve: $T(n) = T(n/4) + T(n/2) + cn^2$
using Master Method:-
let $T(n/2) > T(n/4)$
then $T(n) = 2T(n/2) + cn^2$ $\quad T(n) = aT(n/b + f(n))$.

$a=2,\ b=2$
$c = \log_b a = \log_2 2 = 1$
$n^c = n^1 = n$
$f(n) = n^2 \rightarrow f(n) > n^c \rightarrow n^2 > n$
$T(n) = \Theta(f(n)) \rightarrow T(n) = \Theta(n^2)$

Ans15 int fun( int n) {
for ( int i = 1 ; i<=n; i++)
{ for( j = i ; j<=n ; j+=i)
{
...
} }

complexity of 1st loop = $O(n)$
" " IInd loop = $O(\log(n))$
$T(n) = O(n \log(n))$

**Ans 16:** What should be the time complexity?

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

where k is constant

Here i takes values $2, 2^k, 2^{k^2} = 2k^2, 2k^3 \ldots$

$2^{k(\log_k(\log(n)))} = 2^{\log k} = n$

Therefore we have $\log_k(\log(n))$ iterations and each iteration takes a constant time complexity is $O(\log(\log(n)))$

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

**Ans 17:** The partitioning scheme of 99% and 1% is one of the most unbalanced partition possible.

Time Complexity

Time (n)

$c(n-1)$

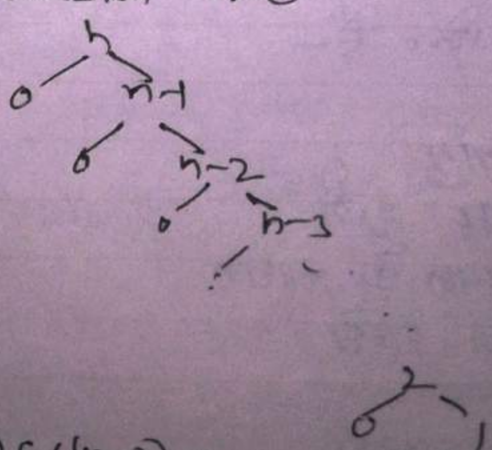
$c(n-2)$

$\vdots$

$2c$

$0$

Recursion tree



Total time $= cn + c(n-1) + c(n-2) + \ldots + 2c$

$= c(\,c(n+1)((n/2)-1)$

using big theta ignore trivial terms

$\therefore$ worst case $=$ $TC = O(n^2)$

Gourav

It is expressed that original call takes (n) time where c is some constant

Ans 18:

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

100 < root (n) < log log (n) < log (n) < n log (n) < n < $n^2$ < log (n!) < $2^n$ < $2^{2n}$ < $4^n$

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

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

Ans 19:

```
int linear (int * arr, int n, int key)
{ for( i<0 to n-1)
    if ( arr[i] = key)
      return i;
    return -1
}
```

Karuna

Ans 20 Iterative insertion sort

```
void insertion (int arr[], int n)
{ for i=1 to n
    int value ← arr[i];
    int j ← i ;
    while (j>0 & arr[j-1] < value)
      arr [i+1] ← arr [j]
        i--;
    } arr[i] ← value ;
  }
```

Recursive insertion sort :-

```
void insertion (int arr[] , int i, int n)
  { int value ← arr[i];
    int j ← i;
   while (j>0  and  arr(j-1)  > value)
    { arr[j] ← arr [j-1];
        j--;
     }
    arr[j] ← value
    if( i<=n)
    { insertion (arr, i+1, n);
    }}
```

Therefore sort is an online sorting algorithm since it can sort a list as it receives it. In all other algo we need all element.

Govind

Ans 21

| ALGORITHM | Time Complexity Best | Avg. | Worst | Space Complexity Worst |
|---|---|---|---|---|
| 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 | $O(n\log n)$ | $O(n\log(n))$ | $O(n\log n)$ | $O(n)$ |
| Heap Sort | $O(n\log n)$ | $O(n\log n)$ | $O(n\log n)$ | $O(1)$ |
| quick sort | $O(n\log n)$ | $O(n\log n)$ | $O(n^2)$ | $O(\log(n))$ |
| Radix sort | $O(nk)$ | $O(nk)$ | $O(nk)$ | $O(\log(n))$ |

Ans 22:

| Inplace | Stable | online |
|---|---|---|
| Bubble Sort | Merge sort | Insertion sort |
| Selection sort | Insertion sort | |
| Heap sort | Bubble Sort | |
| quick sort | | |
| insertion sort | | |

Ans 23. Recursive:

```
Binary_search (arr, l, r, key)
{ if (l>= r) return -1;
  x = key;
  mid = l + (r-l)/2
```

Teacher's signature

```
if (arr [mid] == x) return mid;
if ( arr [mid] > x)
    return Binary-search (arr, l, mid-1, key)
    return Binary search (arr, mid+1, r, key)
```

T.C = O(log(n))   S.C = O log(n)

<u>Iterative:</u>

```
Binary-search( arr, l, r, x)
{ while (l <= r)
  { m = l + (r-l)/2
    if (arr [m] == x)
        return m
    if ( arr[m] < n)
        l = m+1
    else
        r = m-1
  }
  return -1 }
```

T.C = O(log(n))

S.C = O(1)

<u>Ans 2 q:</u>

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

$= T(n/2) + c$

so, $T(n) = T(n/2) + c$

T.C $O(\log n)$

Govind