DAA Assignment

Name : Mansi Agarwal
Section : DS Roll no : 42.
University Roll no - 2015176.

1) Asymptotic notations are methods / languages using which we can define the running time of the algorithm based on input size. Some of the algorithmic run-time in ascending order, the first one being the fastest

i) Logarithmic function = $\log_2 n$.
ii) Linear function = $an + b$
iii) Quadratic function = $an^2 + bn + c$
iv) Polynomial function = $a_2 n^2 + \cdots - a_2 n^2 + \cdots a$ where a & a2 are constants

We need to discard constants and lower order term as they don't weigh much in compared to higher order terms.

i) Logarithmic = $\log_2 n$
ii) Linear = n
iii) Polynomial = n^2.

* Following notations are used :

i) Big-Oh (O) -
$f(n) = O(g(n))$ means,
$g(n)$ is "tight" upper bound of $f(n)$

⇒ $f(n)$ can never go beyond $g(n)$.
⇒ $f(n)$ can never perform worse than $g(n)$.
eg : we have to take complexity of 2 algo.

Algo 1 - $O(n^2+3n+4)$
Algo 2 - $O(2n^2+4)$.

Now to compare them while calculating complexities -
⇒ constants are ignored if it comes as add^n, sub^n, multiplication or division.

Algo 1 - $O(n^2+n)$.
Algo - $O(n^2)$.

⇒ lower order terms are ignored in add^n & sub^n.
X in multiplication or division.

• Take the highest order term.

ii) Big Omega (Ω) -
$f(n) = \Omega\, g(n)$ means
$g(n)$ is "tight" lower bound of $f(n)$

⇒ f(n) can never perform better than g(n)

iii) Theta (Θ) - Θ gives the tight upper & lower bound both.
$f(n) = \Theta(g(n))$.

⇒

* $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$
$f(n) = \Omega(g(n))$.

iv) Small - oh (o) - o gives upper bound
$f(n) = o(g(n))$.
• cannot be equal to g(n) like in big-oh.

v) small - omega (ω) - ω gives lower bound
$f(n) = \omega\, g(n)$.

* If $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$ and vice versa.
* If $f(n) = \Theta(g(n)) \Rightarrow g(n) = \omega(f(n))$
* If $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

2) 1, 2, 4, 8 ... n is a GP.

$a = 1$, $r = 2$.

$n = ar^{k-1} / r-1$

$n = \frac{1 \cdot 2^{k-1}}{2-1} \Rightarrow n = \frac{2^k}{2}$, $2n = 2^k$.

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

Ans $\Rightarrow O(\log_2 n)$

3) $T(n) = 3T(n-1)$ if $n > 0$ — (1)

$T(0) = 1$.

Backward Substitution.

Putting $n = n-1$ in eqn (1).

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

$T(n-1) = 3T(n-2)$ — (2).

Putting (2) in eqn (1).

$T(n) = 3T(3(n-2))$ — (3).

$= 9T(n-2)$

Putting $n = n-2$ in eqn (1).

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

$= 3T(n-3)$

Putting this in (3).

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

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

$n - k = 0 \Rightarrow n = k$.

$T(n) = 3^n T(n-n) = 3^n T(0) = 3^n$.

Ans Time Complexity = $O(3^n)$.

Q $T(n) = 2T(n-1) - 1$ — (1)

$T(0) = 1$.

Backward Substitution.

Putting $n = n-1$ in (1).

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

$= 2T(n-2)$ — (2).

Putting this in (1).

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

$\Rightarrow 4T(n-2)$

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

$= 2T(n-2) - 1$. — (2).

Putting this in (1)

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

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

$= 4T(n-2) - 3$. — (3)

Putting $n = n-2$ in (1)

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

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

Putting this in (3)

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

$= 8T(n-3) - 7.$

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

$\Rightarrow 2^n - 2^{n-1} - 2^{n-2} - \cdots - 2^0.$

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

$= 2^n - 2^n + 1$

$T(n) = 1$

Ans $\Rightarrow O(1)$

5. $1, 3, 6, 9 \cdots - k <= n.$

$k(k+1) <= n$

$k^2 + k <= n.$

$O(k^2) <= n.$

$k <= \sqrt{n}$

Ans $O(\sqrt{n})$

6. Loop ends if $i^2 > n$ so,

$1 <= n$

$4 <= n$

$9 <= n$

$16 <= n.$

$k^2 <= n$

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

7. $i \rightarrow n/2.$ $j \rightarrow \left(\frac{n}{2}\right) \rightarrow n.$

$k \rightarrow \log n\left(\frac{n}{2}\right).$

$O(\log n) * \log(n)$

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

8. $T(n) = T(n-3) + Cn^2$
$T(n) \Rightarrow O(n^3)$.

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

Inner loop runs n/i times for each value of i.

$\Rightarrow n * (\sum_{i=1}^{n} n/i)$.

$= O(n \log n)$

10. n^k is $O(a^n)$
$n^k \leq c \cdot a^n$.
$a^n + n^k \leq c \cdot a^n - an$.
$a^n + n^k \leq an(c-1)$
$\frac{a^n + n^k}{a^n} \leq c-1$

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

$$c \geq \left(\frac{n}{a}\right)^k + 2.$$

$k = 1$, let $a = 1.5$.

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

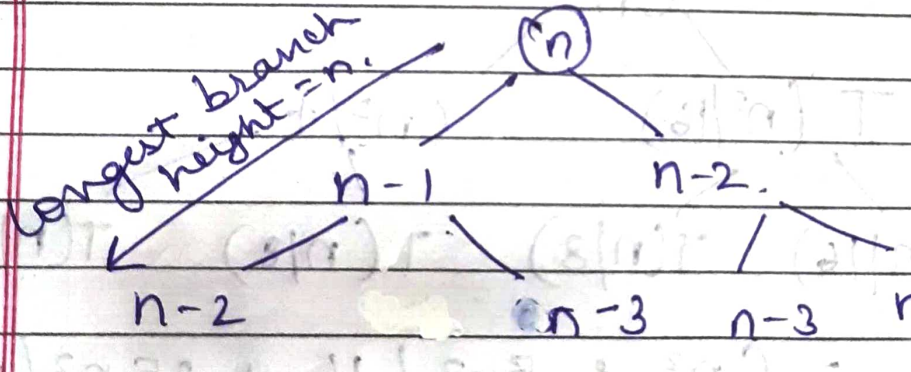
$n_0 = 1$.

$c \geq 30 + 1. \Rightarrow c \geq 4.$

12. It is same as $\sqrt{n}$ $\therefore$ Ans $= O(\sqrt{n})$.

13. $T(n) = T(n-1) + T(n-2) + 1.$

Solving using ~~tree~~ method.



T.C $\Rightarrow$ $1 + 2 + 4 + \cdots 2^n.$

This is a GP

$a = 1, \quad r = 2.$

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

Ans

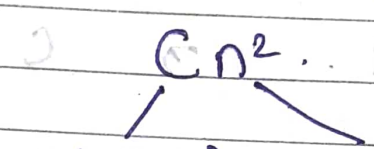
T.C = $O(2^{n+1}) = O(2^n \cdot 2)$
$= O(2^n)$

Space Complexity = $O(n)$.

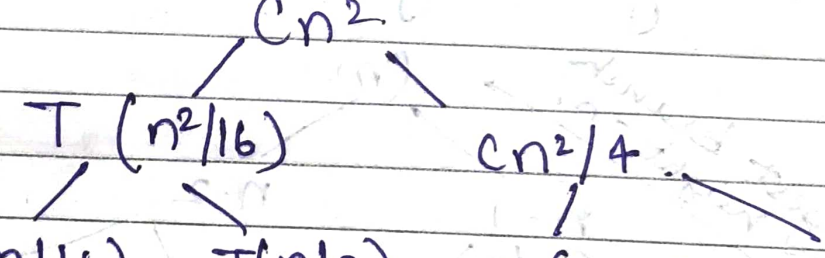
13

$O(\log(\log n))$ ⇒ we get this from Q7.
$O(n^3)$ ⇒ we get this from Q8.
$O(n \log n)$ ⇒ we get this from Q9.

14



On further breaking this down



$T(n) = Cn^2 + 5n^2/16 + 25n^2/256 + \cdots$

This is a GP with ratio 5/16.

So, $\frac{n^2}{1-\frac{5}{16}}$ ⇒ T.C ⇒ $O(n^2)$.

15 This is same as question 9 : O(nlogn)

16 i ⇒ $2, 2^k, (2^k)^k \ldots 2^{k \log_k(\log n)}$
The last term must be less than or equal to n, we have.
$2^{k \log_k(\log_2 n)} = 2^{\log_2 n} \Rightarrow n$.

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

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

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

19
```
int ls ( int a[], int n, int d)
{
    for ( i=0; i<n-1; i++)
    {
        if (a[i] == d)
            return i;
        else if ( a[i] > d)
            return -1;
    }
}
```

15. This is same as question 9 . $O(n \log n)$

16. $i \Rightarrow 2, 2^k, (2^k)^k \ldots 2^{k \log_k(\log n)}$
The last term must be less than or equal to n, we have.
$2^{k \log_k(\log_2 n)} = 2^{\log n} \Rightarrow n$.

18. a) $100 < \log(\log n) < \log n < \sqrt{n} < n < \log(n!) < n \log n < n^2 < 2^n < 4^n < n! < 2^{2^n}$

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

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

19.
```
int ls ( int a[], int n, int d)
{
    for ( i=0; i< n-1 ;i++)
    {
        if (a[i] == d)
            return i;
        else if ( a[i] > d)
            return -1;
    }
}
```

Ans: Time Complexity: Best ⇒ O(1)
~~Worst = O(1)~~, Average ⇒ O(n)

Space Complexity ⇒ O(1)

20

```
void insertion_s(int a[], int n)
{
    int i, j, temp;
    for(i=1; i<n; i++)
    {
        temp = a[i];
        j = i-1;
        while(j>=0 && a[j]>temp)
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = temp;
    }
}
```

This is the pseudo code for iterative insertion sort.

```
void i_s(int a[], int n)
{
    if(n<=1)
        return;
    i_s(a, n-1);
    int l = a[n-1];
    int j = n-2;
```

```
while (j >= 0 && a[j] > t).
{
    a[j+1] = a[j];
    j--;
}.
a[j+1] = t;
}.
```

An online algo is the one that can process its input piece by piece in a serial fashion i.e. in the order that the input is fed to the algorithms without having the entire input available from the beginning.

Insertion sort considers one input element per iteration and produces a partial solution without considering future elements. Thus, it is online algo

21

Sorting Algo	Time Complexity			Space Complexity
	Best	Avg	Worst	
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.	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$.	$O(1)$.

22) Stable sort → insertion, merge, bubble sort
- online sort → insertion sort
- inplace sort → bubble, selection, insertion, heap sort.

23) Iterative pseudo code for binary search.

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

Time Complexity : Best case O(1)
Avg, Worst : O ($\log_2 n$)

Space Complexity : O(1)

Date
Page

» Recursive binary search.

```
int b_s (int a[], int l, int r, int x)
{
  if (r >= l)
  {
      mid = (l+r)/2;
  }
  if (a[mid] == x)
      return mid;
  else if (a[mid] > x)
      return b_s(a, l, mid-1, x);
  else
      return b_s(a, mid+1, r, x);
  }
  return -1;
}.
```

Time Complexity : Best O(1), Avg, Worst $O(\log_2 n)$

Space Complexity : Best O(1), Avg, Worst $O(\log_2 n)$.

» $T(n) = T(n/2) + 1$. is the recurrence relation for binary recursive search