O(logn) time

DAA Assignment 1 (Name- Ishaan Gupta)
Sec-S-E Roll no-18

What do you mean by Asymptotic notations. Define different Asymptotic notations with examples.

Asymptotic notations are methods/languages using which we can define the running time of the algorithm based on input size.

There are ~~four~~ different types of Asymptotic notations:-

Big-O:- Big-O commonly wrote as O, is an asymptotic notation for the worst case or the ceiling of growth for a given function.
If f(n) is your algorithm run time and g(n) is arbitrary constant, then f(n) is O(g(n)) where constant c (c > 0) and no f(n) <= c g(n)

Eg:-

$f(n) = 3\log n + 100$

$g(n) = \log n$

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

$3\log n + 100 = O(\log n)$

$3\log n + 100 \le c * \log n$

Let us take c = 200

$\therefore$ $3\log n + 200 <= 200$

$\therefore$ $f(n)$ is $O(g(n))$

② **Big-Omega (Ω)** :- It is a notation for best case, or a floor growth rate for a given function. It provides us with an asymptotic lower bound for the growth of run time of an algorithm, where $f(n) >= c\, g(n)$

③ **Small-o (o)** :- It is a notation to denote the upper bound (i.e is asymptotically high) on the growth rate of run time of an algorithm. where $f(n) < c\, g(n)$.

④ **Small-omega (ω)** :- It is a notation to denote the lower bound (i.e is not asymptotically tight) on the growth runtime of an algorithm. $f(n)$ is $\omega(g(n))$ for all real constants $c (c > 0)$ and n_0 $(n_0 > 0)$ $\therefore$ $f(n)$ is $> c\, g(n)$.

⑤ **Theta (Θ)** :- It is a notation to denote the asymptotically tight bound on the growth rate of runtime of an algorithm. where $c_1 * g(n) < f(n) < c_2 * g(n)$

Scanned with CamScanner

Date
Page

Q2. T.C = for (i=1; i<=n; ~~i~~)
{
i = i * 2;
}

∴ 1, 2, 4, 8 . . . . n

$a = 1, \quad r = \frac{2}{1} = 2$

$t_n = ar^{n-1}$

$t_k = ar^{k-1}$

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

$n = 2^{k-1}$

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

$2n = 2^k$

$k = \log_2 (2n)$

$k = \log_2 n + 1$

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

Q3. $T(n) = 3T(n-1)$ if $(n>0)$
$T(1) = 1$

~~$T(n) = 3T(n-1) + 0$~~
~~$T(n) =$~~ $T(n) = 3T(n-1)$ - ①

Put $n = \frac{n}{2}$

Put $n = n-1$

$T(n-1) = 3T(n-2)$ - ② $\quad$ ~~$T\left(\frac{n}{2}\right) = 3T\left(\frac{n}{2} - 1\right)$~~

Put ② in ①

$T(n) = 3(3T(n-2))$ $\quad$ $T\left(\frac{n}{2}\right) = 3T\left(\frac{n-2}{2}\right)$ - ②

$T(n) = 6T(n-2)$ - ③

Scanned with CamScanner

Date
Page

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

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

$T(n-2) = 3T(n-3)$ – (4)

Put (4) in (3)

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

$T(n) = 24\,T(n-3)$ – (5)

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

$n-k = 1$

$n = k$

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

Ans. $\boxed{T(n) = 3^n}$

Q4. $T(n) = \{2T(n-1) - 1)$

$T(1) = 1$

$T(n) = 2T(n-1) - 1$ – (1)

Put $n = n-1$ in (1)

$T(n-1) = 2T(n-2) - 1$ – (2)

Put (2) in (1)

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

$T(n) = 4T(n-2) - 2$ – (3)

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

$T(n-2) = 2T(n-3) - 1$ – (4)

Put (4) in (3)

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

$T(n) = 8T(n-3) - 6$

$T(n) = \cancel{2^k \, 8T}\; 2^k T(n-k) - (2^k - 1)$

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

$n = k$

$- O(n^2)$

Date

Page

$T(n) = 2^{n}\cancel{\ } (2^n - 1)$ $\quad 2^n \cdot (2^n - 1)$

$T(n) = 2^{2n} - 2^n$ $\quad T.C = O(1)$

Ans $\boxed{T(n) = 2^{2n}}$

Q5.

$T.C = O(n)$

Q6. $T.C = O(\sqrt{n})$

Q7.

```
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++;
        } } }
```

Ans $T.C = O(n \log^2 n)$

Q8.

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

$T.C = O(n^2)$

Q9.

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

Q10.

n^k is $O(c^n)$

Q

Q11.

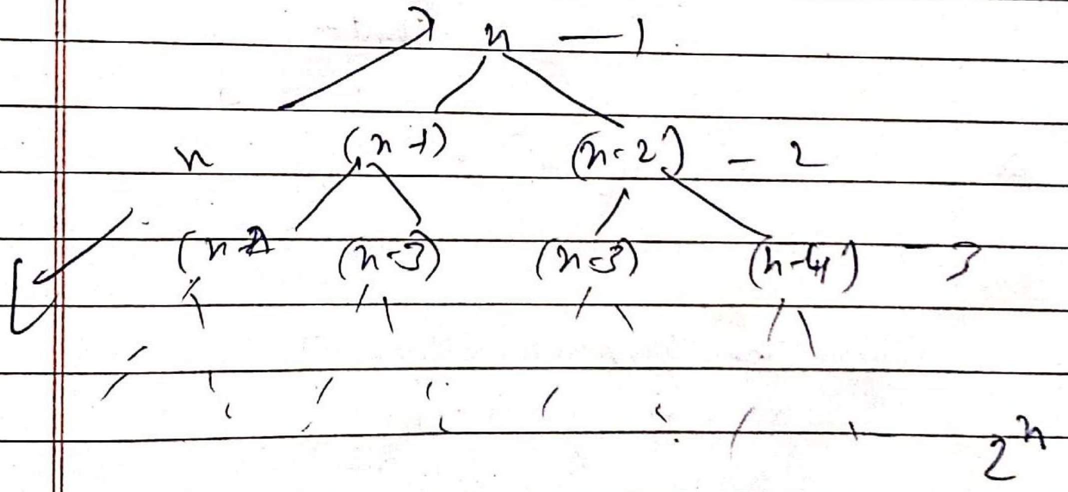
$T.C = O(n)$ because the loop is running from 1 to n i.e n times.

Q12.

Recurrence relation for fibonacci series is

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

$T(1) = 1$



$T.C = 1 + 2 + 4 + 8 - - - 2^n$

$a = 1, \; r = \frac{2}{1} = 2$

Sum of gp $= \frac{a(r^{n+1} - 1)}{(r-1)}$

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

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

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

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

Ans. $T.C = O(2^n)$

Q13.

- For time complexity = $n(\log n)$

Eg:-

```
for (i=0; i<n; i=i*2)
{   Some O(1) work
}
```

- For $T.C = O(n^3)$

Eg:-

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        for (k=0; k<n; k++)
            { Some O(1) work }
```

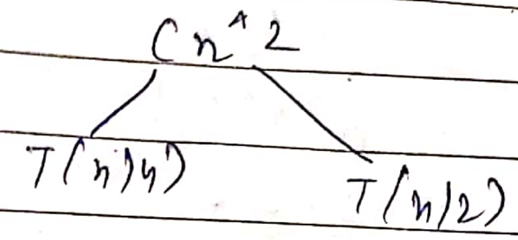
- For: $T.C = O(\log(\log n))$

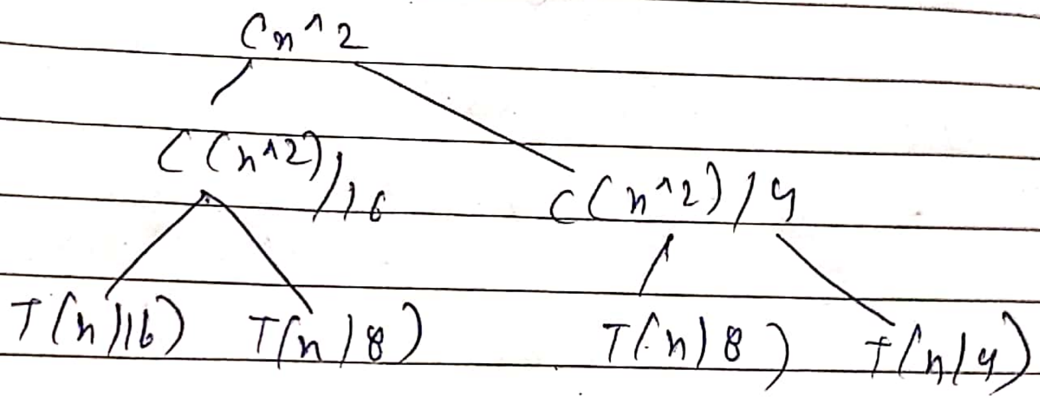
Eg:-

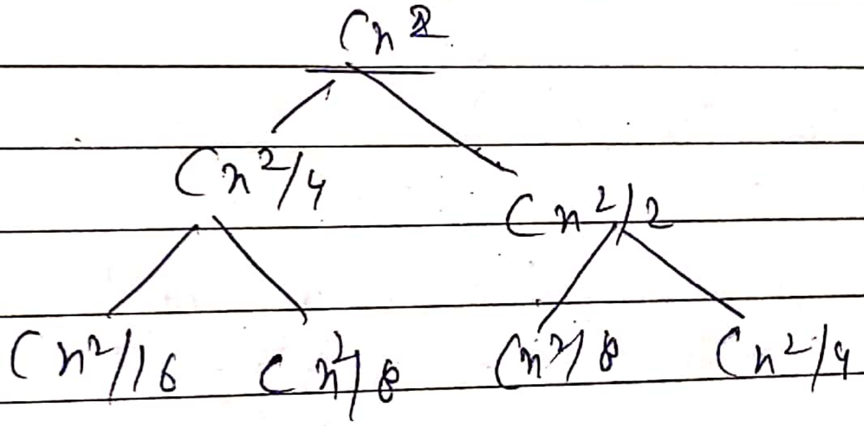
```
for (int i=2; i<n; i=pow(i,c))
{
    Some O(1) work
}
```

Scanned with CamScanner

$T(n) = T(n/4) + T(n/2) + Cn^2$







$\therefore\ Cn^2 + 3Cn^2/4 + 9Cn^2/16$

$O\left(\frac{Cn^2}{1-3/4}\right) = O(n^2)$

Q15.

T.C = $O(n^2)$

Q16.

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

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

Q18.

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

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

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

Q19. Linear Search

```
for (i=0; i<n; i++)
{
    if (arr[i] == key)
        return true
}
return false
```

Q20

<u>Iterative Insertion sort</u>

```
for (i=1; i<n; i++)
  {
    int key = arr[i];
    int j = i-1
    while ( j>=0 && arr[j]>key )
    {   arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = key;
  }
}
```

<u>Recursive Insertion sort</u>

```
void insertionSort (int arr[], int n)
{
    if (n<=1)
        return;

    insertionsort ( arr, n-1);
    int last = arr[n-1]
    int j = n-2;
    while (j>=0 && arr[j] > last)
    {
        arr[j+1] = arr[j];
        j--;
    }  arr[j+1] = last; }
```

Insertion sort is an online sorting algorithm because it process input piece by piece in a serial fashion i.e in the order that the input is fed to algorithm without having the entire input available from the beginning

Q21.

① Bubble sort
$T.C = O(n^2)$

② Insertion Sort
$T.C = O(n^2)$

③ Selection Sort
$T.C = O(n^2)$

④ Merge Sort
$T.C = O(n \log n)$

⑤ Quick Sort
$T.C = O(n \log n)$

Q22.

① Bubble Sort
- It is stable.
- It is inplace
- It is offline

② Selection Sort
- It is not stable
- It is inplace
- ~~It is~~

③ Insertion Sort
- It is stable

Scanned with CamScanner

Date
Page

- It is inplace
- It is online

(4) Merge Sort
- It is stable.
- It is not-inplace
- It is offline

(5) Quick sort
- It is not stable.
- It is inplace
- It is offline

Q28

<u>Iterative binary search</u> T.C = O(logn)
S.C = O(1)

```
int binarysearch (int* arr, int l, int r, int key)
{
    while (l <= r)
    {
        int m = l + (r-l)/2;
        if (arr[m] == key)
            return m;
        else if (arr[m] > key)
            l = m+1;
        else
            r = m-1;
    }
```

Date
Page

Recursive binary Search

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

T.C = O(logn)
S.C = O(logn)

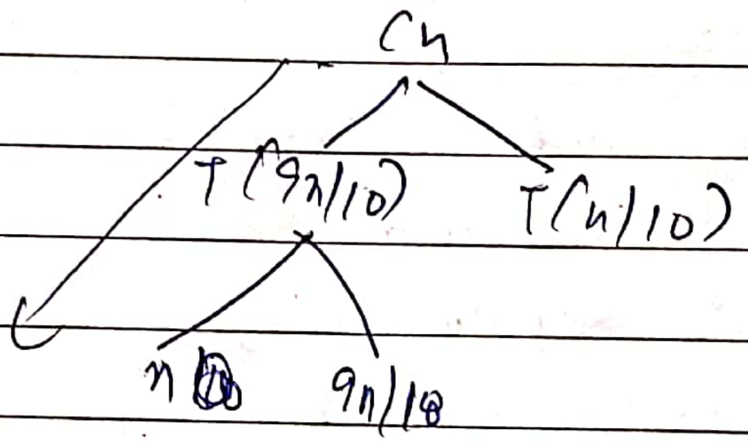
Q24.

Recursive Relation for binary Search

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

Recurrence relation will be

$$T(n) = T(9n/10) + T(n/10) + O(n)$$



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