# DAA

Name - Kanchan Rautela
Section - CE
Uni. Roll No. - 2015631

## ASSIGNMENT-1

Ans 1 - Asymptotic notations are methods/languages using which we can define the running time of the algorithm based on input size.
There are mainly 3 asymptotic notations

(1) Big-O Notation: Big-O Notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.

eg:

$O(\log n)$ = Binary search
$O(n)$ = simple search

(2) Omega Notation: It represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

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

EXAMPLE: for a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions.

$\Omega(g(n)) = \{f(n)$: there exist positive constants $c$ and $n_0$ such that $0 <= c * g(n) <= f(n)$ for all $n >= n_0\}$

(a) Theta Notation ($\Theta$): Theta notation encloses the function from above & below since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

for a function $g(n)$, $\Theta(g(n))$ :-

$\Theta(g(n)) = \{ f(n)$: there exist positive constants $c_1, c_2$ and $n_0$ such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$

**Ans 2:**

```
for(i=1 to n)
 { i = i*2
 }
```

Time complexity : $O(\log n)$

As per qstn. it will run $2^k = n$ which leads to $k = \log n$

**Ans 3:**

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

$T(n) = 3T(n-1)$ (solve using backward substitution)

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

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

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

=

=

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

$= 3^n T(0)$

$= 3^n$

This clearly shows that the complexity of this function is $O(3^n)$

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

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

$$T(n) = 1$$

Time complexity is $O(1)$.

Ans 5: Time Complexity: $O(\sqrt{n})$.

we know that the value of 'i' increases by one for each iteration. The value contained 's' at the ith iteration is the sum of the first 'i' positive integers. If k is total number of iterations taken by the program, then while loop terminates if: $1+2+3 \cdots + k = [k(k+1)/2] > n$ so $k = O(\sqrt{n})$

Ans 6:

```
void function(int n)
{ int i, count = 0;
  for (i = 1; i*i <= n; i++)
    count++;
}
```

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

Ans 7: Outer loop = $n/2 = n$
Inner 1'st loop = $\log n$
Inner 2nd loop = $\log n$

Time Complexity : $O(n^2 \log n)$

Ans 8: function (int n)
{ if (n==1)
return;
for (i=1 to n)
for (j=1 to n)
printf(" * ");
function (n-3);
}

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

$T(1) = O(1)$

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

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

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

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

$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \dots + (n-k)^2 + T(n-k-3)$

$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \dots + 4^2 + T(1)$

$= n^2 + (n-3)^2 + (n-6)^2 + \dots + 7^2 + 4^2 + 1$

$= \sum_{k=1}^{(n+2)/3} (3k-2)^2 = \sum (9k^2 + 4 - 12k)$

T.C. $\simeq \theta(n^3)$

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

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

$j = \frac{n}{1}, \frac{n}{2}, \frac{n}{3} \ldots \frac{n}{n}$

$T(n) = \frac{n}{1} + \frac{n}{2} + \frac{n}{3} \ldots \frac{n}{n}$

$= n\left[\frac{1}{1} + \frac{1}{2} + \frac{1}{3} \ldots \frac{1}{n}\right]$

T.C. $= n \log n$

Ans 11: The loop variable 'i' is incremented by 1, 2, 3, 4 --- until i becomes greater than or equal to n.
The value of i is $x(x+1)/2$ after x iterations.
So if loop runs x times, then $x(x+1)/2 < n$. Therefore TIME COMPLEXITY can be written as $(\sqrt{n})$.

Ans 12: fibonacci series:-

0 1 1 2 3 5 8 13 21 34 55 ----

$$fib(n) = \begin{cases} 1, 0 & (n=1 \text{ or } n=0 \text{ resp.}) \\ fib(n-2) + fib(n-1), & n > 1 \end{cases}$$

The recursive equation for fibonacci series is:-

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

First of all, assume $T(n-2) = T(n-1)$
Solving it using Backward substitution.

$T(n) = 2T(n-1) + 1$ //now solve

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

$$T(n) = 2[2T(n-2) + 1] + 1$$
$$= 4T(n-2) + 3$$

Next, we can substitute in $T(n-2) = 2T(n-3) + 1$

$$T(n) = 2[2[2T(n-3) + 1] + 1] = 8T(n-3) + 7$$

$$\vdots$$

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

Now, we can find k & thereby solve T(n), by Substituting in $T(0)=1$

For $T(0)$, we can see that $n-k=0$.

Rearranging, we get $k=n$. Now substituting in our values for $T(0)$ & $k$, we get

$$T(n)=2^n T(0)+(2^n-1)=2^n+2^n-1$$

$\boxed{\text{Time Complexity} = O(2^n)}$

Space Complexity would be : $O(1)$. Since, the program does not use extra spaces.

Ques 13 $T(n)=O(n\log n)$

```
for (int i=0; i<n; i++)
    for (int j=0; j<n; j=j*2)
        cout<<"*";
```

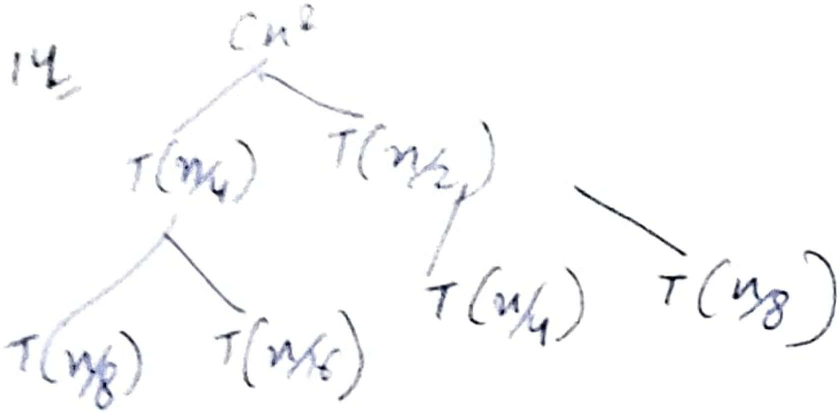
$T(n)=n^3$

```
for (int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        for(int k=0; k<n; k++)
    cout<<i<<j<<k<<endl;
```

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

```
for(int i=0; i<log(n); i=i*2)
    cout<<i<<" ";
```

Ques, 14




$$T(n) = cn^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \cdots$$

This GP with ratio $5/16$

$$\therefore T(n) = \frac{n^2}{1 - 5/16} \Rightarrow T.C. = O(n^2)$$

Q 15 $\Rightarrow$ int fun (int n)

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

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

$j = n, n/2, n/3, n/4 \cdots n/n$

$\therefore$ T.C. $= n, \frac{n}{2}, \frac{n}{3} + \cdots \frac{n}{n}$

$n\left[1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right]$

$= n \log n$

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

Q16 for (int i=2; i<=n; i=pow(i,k))

{ O(1)

$i = 2, 2^k, 2^{k^2}, 2^{k^3}, \ldots, 2^{k^x}$ i.e. $(x+1)$ terms

$2^{k^x} = n$

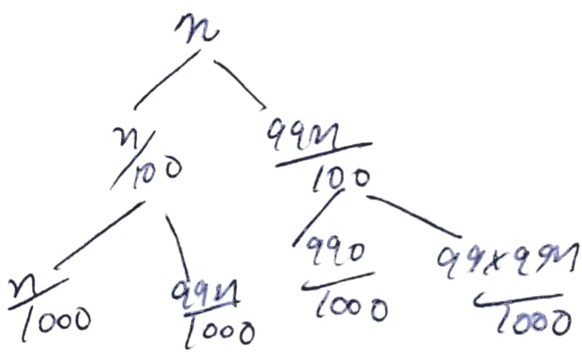
$k^x = \log_2 n$

$x = \log_k(\log_2 n)$

$\therefore$ T.C. $= O(\log_k \log_2 n)$

Q17 ⇒ $T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right) + cn$

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




Q18 ⇒ int linearSearch (int arr, int n, int key)

```
{ int i;
for(i=0; i<n; i++)
  if (arr[i] == key)
    return i;
  else if (arr[i] > key)
    return -1;
}
```

$\boxed{\begin{array}{l} T.C. = O(n) \\ S.C. = O(1) \end{array}}$

Q20 ⇒ Iterative Insertion Sort

```
void insertion sort (int *a, int n)
{ int i,j, temp;
  for (i ← 1 to n)
    temp ← a[i]
    j ← i-1
    while (j >=0 && a[j] > temp)
      a[j+1] = a[j]
      j ← j-1
    a[j+1] ← temp
}.
```

Recursive Insertion sort.

```
void insertionSort (int *a, int n)
{ if (n<2)
      return;
  insertionSort (a, n-1)
  int last ← a [n-1]
  int j ← n-2
  while (j >=0 && a[j] > last)
      a[j+1] ← a[j]
      j = j -1
  a[j+1] ← last
}
```

Online sorting is one that will work if elements to be sorted are provided 1 at a time with understanding that algo must keep sequence sorted as more & more elements are added. Insertion Sort is online.

Ques, 21, 22

| Algo | Best | avg | worst | worst space | Inplace | Stable | Online |
|---|---|---|---|---|---|---|---|
| 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 | $O(n\log n)$ | $O(n\log n)$ | $O(n\log n)$ | $O(1)$ | ✓ | ✗ | ✗ |

Q 23 ⇒ Iterative Binary Search

```
int binarySearch(int *a, int l, int r, int key)
{ while (l<=r)
      m = (l+r)/2
      if (a[m] == key)
          return m
      if (a[m] < key)
                    l ← m+1
      else
                    r ← m-1
```

return -1

}

Q24 => Binary Recursive Search

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