DAA ASSIGNMENT-1

- SUHANI THAPLIYAL
CST
2015397

Q1. Asymptotic notations describes the ~~algorithm's~~ algorithm's efficiency & performance. It describes the time & space complexities of algorithms.
Different types of Asymptotic notations are:

① Big-O :- This notation defines an upper bound of an algorithm i.e. bounds a function only from above.
Example: Big O notation of linear search → $O(n)$.

② Big Omega (Ω) :- It is notation used for best case. It provides us lower bound of algorithm.
e.g.- linear search → $\Omega(1)$.

③ Theta (Θ) :- It gives us the tight upper & lower bound of algorithms
eg. → linear search → $\Theta(n)$.

④ Small-o :- Denotes the upper bound (not asymptotically tight) of an algorithm.
$f(n) = o(g(n))$; $f(n) < c.(g(n))$

⑤ Small-omega (ω): Denotes the lower bound (not asymptotically tight) of an algo.
$f(n) = \omega(g(n))$; $f(n) > c.(g(n))$

Date

Page

Q2. for (i=1 to n)

$i = i * 2$

$2^0, 2^1, 2^2, 2^3, \ldots 2^k$

$2^k = 2n$

$k = \log_2(2n)$

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

$k = \log_2 n + 1$

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

$= O(\log n)$

Q3. $T(n) = 3T(n-1)$

$T(1) = 1$

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

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

$T(4) = 3T(4-1) = 3T(3) = 27$

$\vdots$

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

$\therefore$ T.C. $= O(3^n)$

$3 \cdot 3^1 \quad 3^1$

$3 \cdot 3^2 \quad 3^2$

27

$T(5) = 3T(5-1)$

$= 3 \cdot T(4)$

$= 3 \cdot 27$

81

Q4. $T(n) = 2T(n-1) - 1$ — (1)

$T(1) = 1$

By using backward substitution

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

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

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

Put $n = n-2$

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

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

$\vdots$

Page

$T(n) = 2^k T(n-k) - \dots - 2^3 - 2^2 - 2^1 - 2^0$

w.k.t. $T(1) = 1$

$n-k = 1$

$k = n-1$

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

$= 2^{n-1} \times 1 - [2^{n-1} - 1]$

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

$= 1$

$\therefore$ T.C. $= \underline{O(1)}$.

Q5.

```
i=1, s=1
while(s<=n)
    i++
    s=s+i
    print("#")
end
```

$\Rightarrow$

i	s
1	1
2	3
3	6
4	10
5	15
⋮	⋮
k	$\frac{k(k+1)}{2}$

$k(k-1)$

$\sum_{i=1}^{k} \sum_{s=s+i}^{k} 1$

$\sum_{i=1}^{k} (1+1+\dots k-(s+i)+1)$

$(F = U - L + 1$

$= k - (s+i) + 1$

$\sum_{i=1}^{k} (k - s + i + 1)$

$\sum_{i=1}^{k} k - \sum_{i=1}^{k} s+i+1$

$k^2 - (1+3+6+10 \dots k)$

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

Scanned with CamScanner

Q6.
```
void func(int n) {
    int i, c=0
    for(i=1; i*i<=n; i++)
        { c++
        }
```

$i*i = n$

$i^2 = n$

$i = \sqrt{n}$

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

Q7.
```
void func(int n) {
  int i, j, k, c=0
  for (i=n/2; i<=n; i++)          ← O(n)
    for (j=1; j<=n; j=j*2)        ← O(log n)
      for(k=1; k<=n; k=k*2)       ← O(log n)
        c++
  }
```

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

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

Q8.
```
func. (int n) {
  if (n==1) return;
  for (i=1 to n)              ← O(n)
    for (j=1 to n)            ← O(n)
      printf("*");
  }
  func(n-3);                  ←
}
```

n
n-3
n-6
n-9
⋮
n/3 levels → O(n/3)

$\therefore T.C. = n/3 * n * n$
$= \underline{O(n^3)}$

~~Q~~ Q9. void func (int n)

```
for (i=1 to n)                      ← O(n)
    for (j=1; j<=n; j=j+i)          ← O(logn)
        printf("*")
```

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

Q10. n^k, a^n ; $k >= 1$ & $a > 1$

$c = ?$ $n_0 = ?$

Asymptotic relation:

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

If we take $n=2$, $k=2$, $c=2$

then, $2^2 \leq 2^2$ so c^n is upper limit of n^k.

Q11. void fun (int n) {

```
    int j=1, i=0;
    while (i<n)
        i = i+j;
        j++ ;
}
```

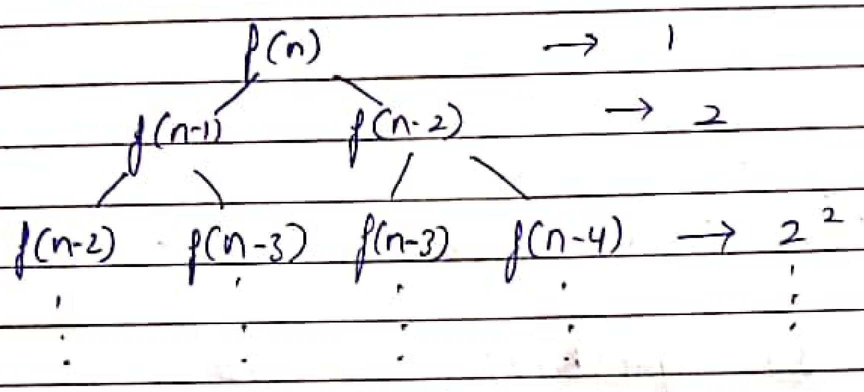
i	j
0	1
1	2
3	3
6	4
10	5
⋮	⋮
$\frac{k(k+1)}{2}$	k

The value of i is $\frac{k(k+1)}{2}$ after k iterations. So if loop runs k times then $\frac{k(k+1)}{2} < n$. Therefore time complexity is $\underline{O(\sqrt{n})}$.

Q12. Recurrence Relation for Fibonacci series (recursive):

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

Solving this using tree method



$T.C. = 1 + 2 + 2^2 \dots + 2^n$

Using G.P. formula ; $a = 1$, $r = 2/1 = 2$

$$= \frac{a\,(r^{\text{terms}} - 1)}{(r-1)}$$

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

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

Space Complexity will be $O(n)$ as there are 'n' stacks or 'n' is the height of the tree.

Q13. (i) $O(n \log n)$:

```
for (i=1; i<=n; i=i*2)          // O(logn)
{
  for ( j=1; j<=n; j++)          // O(n)
     sum++;
}
```

(ii) $O(n^3)$:-

```
for (i=0; i<n; i++) {                    // O(n)
  for (j=0; j<n; j++) {                  // O(n)
    for (k=0; k<n; k++)                  // O(n)
      sum++
}}
```

(iii) $O(\log(\log n))$:-

```
~~for (i=1; i<=n; i=i*2)~~               ~~// O(logn)~~
~~{~~
~~   s += sqrt(i);~~                       ~~//~~
~~}~~
```

```
for (int i=2; i<=n; i=pow(i,k))          // loop variables are
{   sum++;                                  increased or
}                                           decreased exponentially
                                            by constant amount(k)
```

Q14: $T(n) = T(n/4) + T(n/2) + cn^2$

$T(\frac{n}{2}) >= T(n/4)$

$\therefore T(n) <= 2T(\frac{n}{2}) + cn^2$

$a=2$; $b=2$, $\log_b a = \log_2 2 = 1$; $f(n) = n^2$

$f(n) > n^{\log_b a}$; $n^2 > 1$

$\therefore$ Time Complexity $= O(n^2)$

Q15.
```
int fun (int n) {
  for (int i = 1; i <= n; i++)
    for (j = 1; j <= n; j += i)
      // O(1)
}
```

$i=1 \Rightarrow j=1$ to n times
$i=2 \Rightarrow j=1$ to $n/2$ times
$i=3 \Rightarrow j=1$ to $n/3$ times
$\vdots$
$i=n \Rightarrow j=1$ to $\frac{n}{n}$ times

$$T(n) = n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} \cdots \frac{n}{n}$$
$$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \cdots \frac{1}{n}\right)$$
$$= n \sum_{k=1}^{n} \frac{1}{k}$$
$$= n \log n$$

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

Q16. $O(\log \log n)$ (loop variables are increased exponentially by constant amount).

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

$T(1) = 1$

using master theorem ~~$a=1$, $b=100$, $\log_b a = \log_{100} 1$~~
~~$\log_{100} 1 = \frac{\ln 1}{\ln 100} = \frac{0}{4.6} = 0$~~

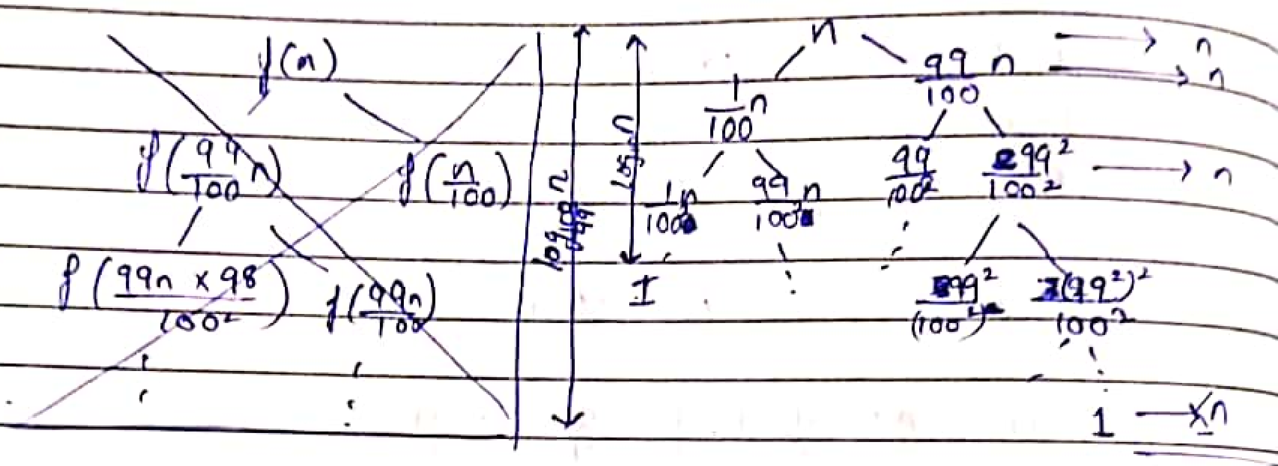
$\therefore n^0 = n$

$f(n) = n$

$f(n) = n$

~~$\therefore$ Time complexity $= O(n^{\log_b a} \log n)$~~

~~$\Rightarrow O(n \log n)$~~



$O(n \log n)$

$$T.C. = O(n \log_{\frac{100}{99}} n)$$

$$= O\left(\frac{n \log_2 n}{\log_2 \frac{100}{99}}\right) = O(n \log_2 n)$$

Q18 a.) $100 < \log\log n < \log n < \sqrt{n} < \log(n!) < n\log n < n^2 < 2^n < 2^{2n} < 4^n < n!$

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

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

Q19.
```
int linear (int a[], int key, int n) {
        for i <- 0 to n
            if (a == key)
                return i;
    return 1;
}
```

classmate
Date
Page

Recursive:-

p20.
```
void insertion(int a[], int n)
{
    if(n<=1)
        return;
    insertion(a, n-1);
    int last = a[n-1];
    int j = n-2;
    while (j>=0 && a[j]>last)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = last;
}
```

Iterative:-

```
void insertion (int a[], int n)
{
    int i, temp, j;
    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;
    }
}
```

Q21. Bubble Sort $\Rightarrow O(n^2)$
Selection sort $\rightarrow O(n^2)$
Insertion sort $\Rightarrow$ Best - $O(n)$, worst & Avg $= O(n^2)$
Merge sort $\Rightarrow \Theta(n \log n)$
Quick sort $\Rightarrow$ Best & Avg $= O(n \log n)$, worst $= O(n^2)$
Heap sort $\Rightarrow O(n \log n)$

Q22.

~~Stable~~	stable	Inplace	Online
Bubble	✓	✓	X
Selection	X	✓	X
Insertion	✓	✓	✓
Merge	✓	X	X
Quick	X	X	X
Heap	X	✓	X

Q23. Iterative :

```
l=0, r= n-1
while (l <=r)
 {
     mid = (l+r)/2 l+(r-l)/2;
     if (a[mid] = x)
         return mid;
     if (a[mid] < x)
         l = m+1 ;
     else
         r = mid-1;
   }
 return -1;
}
```

classmate
Date
Page

Recursive:

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

Time complexity (Recursive & Iterative) = $O(\log n)$
T.C. of linear search = $O(n)$

#24. Recurrence for binary search:

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

$T(1) = 1$

—x—