PAGE NO.: 1
DATE: / /20

Sarthak Sharma
BTech CSF 5th Sem
Section C
1961152

Assignment 1 (DAA)

1)

Sol Asymptotic means towards infinity. These are used to tell the complexity of an algorithm when the input is very large (10^8, 10^9, etc)

Types of Asymptotic Notations:

i) Big-O $\Rightarrow$ $f(n) = O(g(n))$

where $g(n)$ is tight upper bound of $f(n)$. $f(n)$ can never go beyond $g(n)$

Eg $\Rightarrow$ $f(n) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$

$$f(n) = O(x^n)$$

ii) Big Omega (Ω) $\Rightarrow$ $f(n) = \Omega(g(n))$

where $g(n)$ is tight upper bound of $f(n)$ & $f(n)$ is always never perform better than $g(n)$.

Eg ⇒ For the sorting of the array if array is already sorted so insertion sort will take place.

$\Omega(n)$ is the Best time complexity

iii) theta - θ ⇒ $f(n) = \theta(g(n))$ if

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

It gives the time complexity of algorithm at average cases

Eg ⇒ when array is unsorted so all complexity will be $O(n)$.

---

2)

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

The series formed is: 1, 2, 4, 8, 16, ... K^{th} term upto n terms

So by using GP: $a_n = a\,r^{K-1}$

$a_n = n$, $a = 1$, $r = 2$,

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

$n = \frac{2^K}{2}$ ⇒ $2n = 2^K$

taking $\log_2$ on both sid.

$$\log_2 2n = \log_2 2^K$$

$\log_2 2 + \log_2 n = K \log_2 2$

$1 + \log_2 n = K$

K = Time complexity $= O(\log_2 n) = .$

$K = \log n$

---

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

$T(n) = 3T(n-1) \qquad T(0)=1$

Sol. using substitution method

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

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

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

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

for K terms

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

let $T(n-K) = T(0)$

$n = K$

using $n = K$

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

$T(n) = 3^n \cdot 1$

$T(n) = 3^n$

---

4) $T(n) = 2T(n-1)-1 \qquad T(0) = 1$

Sol $T(n) = 2T(n-1)-1 \rightarrow T(n-1) = 2T(n-2)-1$

$T(n) = 2[2T(n-2)-1]-1 \rightarrow T(n-2) = 2T(n-3)-1$

$T(n) = 2[2\{2T(n-3)-1\}-1]-1$

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

$\Rightarrow 8T(n-3) - 7$

For K^{th} term

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

let $T(n-K) = T(0)$

$n = K$

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

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

---

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

Sol. Here no of steps.

	i	s
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15

Order of growth is not constant so general term is $\frac{K(K+1)}{2}$

so $n = \frac{K(K+1)}{2}$

$n = \frac{K^2+K}{2} \Rightarrow 2n = K^2 + K$

ignoring lower order terms

$n = K^2$

$K = \sqrt{n}$

so $T(n) = \sqrt{n}$

---

6) `for (i=1 ; i*i <=n ; i++) count ++;`

sol. so i is moving from 1 to $\sqrt{n}$ with linear growth

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

---

7)
```
for (i = n/2 ; i<=n ; i++)
    for (j=1 ; j<=n ; j=j*2)
        for (K=1 ; K<=n ; K=K*2)
            count ++;
```

Since the j & K loops are running from 1 to n with exponential growth so Time Complexity for these 2 loops will be.

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

& i loop is moving from $\frac{n}{2}$ to n with constant growth.

$T(n) = \Theta\left(\frac{n}{2}\right) = O(n)$

So overall complexity of the f^n will be

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

---

8) $T(n) = T(n-3) + n^2$ & $T(1) = 1$

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

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

$T(n) = T(n-9) + (n-6)^2 + (n-3)^2 + n^2 \Rightarrow T(n-9) = T(n-12) + (n-9)$

$T(n) = T(n-12) + (n-9)^2 + (n-6)^2 + (n-3)^2 + n^2$

general term is

$T(n) = T(n-k) + (n-(k-3))^2 + (n-(k-6))^2 + (n-(k-9))^2 + (n-(k-12))^2 + \dots + (n-(k-k))^2$

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

$n = 1+k \quad \& \quad k = n-1$

$T(n) = T(n-(n-1)) + [n-(n-1-3)]^2 + [n-(n-1-6)]^2 + [n-(n-1-9)]^2 + \dots + n^2$

$T(n) = T(1) + 4^2 + 7^2 + 10^2 + \dots + n^2$

$= 1 + 4^2 + 7^2 + 10^2 + \dots + n^2$

General formula is

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

$n = 9k^2 + 4 - 12k$

by ignoring lower terms

$n = k^2$

$k = \sqrt{n}$.

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

---

a)
```
for (i=1 to n)
    for (j=1 to n)  {j = j+i}
        print ("*")
```

sol: steps =>

i	1	2	3	4	5	.....	n
j	n	$\frac{n}{2}$	$\frac{n}{3}$	$\frac{n}{4}$	$\frac{n}{5}$	.....	$\frac{n}{n} = 1$

$$TC = n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \frac{n}{5} + \dots + \frac{n}{n}$$

$$TC = n\left[1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{n}\right]$$

$$TC = n \int_1^n \frac{1}{x}\, dx$$

$$TC = n [\log x]_1^n = n[\log n - \log 1]$$

$$TC = n \log n$$

---

10) if $c > 1$, then the exponential c^n for sufficiently large, so that c^n is n^k is $O(c^n)$

---

11)
```
int j=1, i=0;
while(i<n){
    i=i+j;
    j++;
}
```

sol.

i =	0	1	3	6	10	15	.....
j =	1	2	3	4	5	6	.....

Here 'i' will go till n & general formula for K^{th} term is $n = \frac{K(K+1)}{2}$

so $TC = O(\sqrt{n})$

12) So the relation for the fibonacci series is

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

let $T(n-1) \cong T(n-2)$

$= T(n) = 2T(n-1) + C \rightarrow T(n-1) = 2T(n-2) + C$

$T(n) = 2[2T(n-2) + C] + C \rightarrow T(n-2) = 2T(n-3) + C$

$T(n) = 4T(n-2) + 3C$

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

$T(n) = 8T(n-3) + 7C$

So for the K^{th} term until the

$T(n) = 2^K T(n-K) + (2^K - 1)C$

& $T(n-K)$ for $T(0) = 1$

$= n - K = 0 \Rightarrow n = K$

$T(n) = 2^n \cdot 1 + (2^n - 1)C$

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

$= 2^n [1 + C] - C$

neglect the C

$T \cdot C = O(2^n)$

And max depth for recursive function is based on n so space complexity is $O(n)$

---

13) i) $n \cdot \log n$

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

ii) n^3

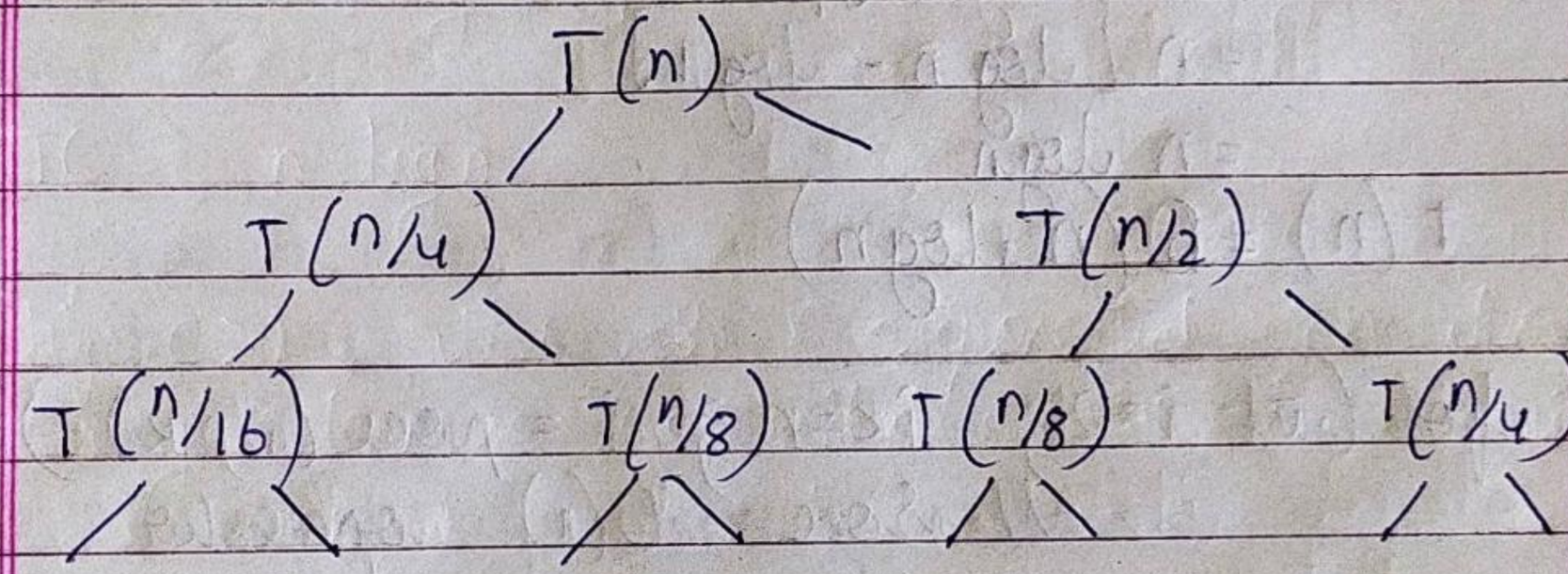
Eg => for (int i=0 ; i<n ; i++)
for(int j=0 ; j<n ; j++)
for(int k=0 ; k<n ; k++)
printf("#");

iii) log (log n)

Eg => for(int i=0 ; i<n ; i=i*i)
printf("Hello");

---

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



$$T(n) = C\left[n^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \dots\right]$$

so here

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

---

15) for (int i=1 ; i<=n ; i++)
for (int j=1 ; j<n ; j+=i)
//some O(1) operation

sol.

i =	1	2	3	4	5	.....	n
j =	n	$\frac{n}{2}$	$\frac{n}{3}$	$\frac{n}{4}$	$\frac{n}{5}$		$\frac{n}{n} = 1$

So, here,

$$T(n) = n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \dots + \frac{n}{n}$$

$$= n\left[1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{n}\right]$$

$$= n \int_1^n \frac{1}{x}\, dx$$

$$= n [\log x]_1^n$$

$$= n [\log n - \log 1]$$

$$= n \log n$$

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

---

16) `for (int i=2; i<=n; i = pow(i, k))`
`// some O(1) operation`

Sol For any function increasing with exponential rate the $T(n)$ become

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

---

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

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

b) $2(2^n)$, $4n$, $2n$, 1, $\log(n)$, $\log(\log n)$,

$\sqrt{\log(n)}$, $\log 2n$, $2\log n$, n, $\log n!$, $n!$, n^2, $n \log k$.

Sol> $1 < \log(\log n) < \sqrt{\log(n)} < \log n < \log 2n < \log(n!) < 2\log(k) < 2n < 2n < 4k < n\log(n) < n^2 < 2(2^n) < n!$

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

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

19-

Sol>
```
Search (A, n)
  low = 0   high = n-1,   k
  while (low ≤ high)
      mid = (low + high)
                 2
      if A[mid] = key
          return mid
      else if (A[mid < key)
          low = mid + 1
      else
          high = mid - 1
```

20-

Sol> Iterative method ->
```
insertion sort (A)
  for j = 2 to A.size
      key = A[j];
```

```
i = j - 1
while i > 0 AND A[i] > key
        A[i+1] = A[i]
A[i+1] = key
```

Recursive Method ⇒

```
Insertion_sort (A, n)
  if (n ≤ 1)
      return
  Insertion_sort (A, n-1)
  key = A[n-1]
  j = n-2
  while j ≥ 0 AND A[j] > key
      j = j - 1
  A[j+1] = key
```

This sort considers one i/p element per iteration and produces a partial solution without considering future elements that's why it is called online sorting.

Other sorting algo's are.

Bubble sort	$O(n^2)$	selection sort	$\Theta(n^2)$
Insertion sort	$O(n^2)$	merge sort	$O(n \log n)$
Heap sort	$O(n \log n)$	quick sort	$O(n \log n)$

Q21-
sol.

Algos.	Best case	Average case	Worst case.
Bubble sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Selection sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Insertion sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Merge sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$
Heap sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$
Quick sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$
Counting sort	$\Omega(n+K)$	$\Theta(n+K)$	$O(n+K)$

Q22.
Sol.

Algos	Inplace	stable	Online
Bubble sort	Yes	Yes	Yes
Insertion sort	Yes	Yes	Yes
selection sort	Yes	No	Yes
Merge sort	No	Yes	Yes
Quick sort	Yes	No	Yes
Heap sol	Yes	No	Yes
Count sort.	Yes	Yes	Yes.

Q23.
Sol. ~~Linear~~ Iterative method :

```
BinarySearch (A, n, Key)
    i = 0 , j = n-1
    while (i <= j)
        mid = (i + j) / 2
        if A[mid] == Key
            return mid
        else if A[mid] < Key
            i = mid + 1
        else
```

```
        j = mid - 1
    return -1
```

Time complexity = $O(\log_2 n)$
Space complexity = $O(1)$

Recursive method ⇒

```
Binary_search (A, i, j, Key)
    if (i > j)
        return -1
    mid = (i + j) / 2
    if A[mid] == Key
        return mid
    else if A[mid] < Key
        return binary_search (A, mid+1, j, Key)
    else
        return binary_search (A, i, mid-1, Key)
```

Time complexity ⇒ $O(\log n)$
Space complexity ⇒ $O(1)$

Linear search ⇒ Time Complexity ⇒ $O(n)$
Space Complexity ⇒ $O(1)$

24)
Sol Recursion relation for binary search is

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

X ⟶ X