Name : Nitesh Pant
Sec : DS

vibhu
PAGE No
DATE

Assignment - 1 (DAA)

1> Asymptotic Notation: The study of change in performance of the algorithm with the change in the order of input size.

Different Asymptotic Notations:

1> Big O Notation: It represents the upper bound of the running time or the worst case time for an algorithm.
for example

```
While ( i > a len(arr))
    if (arr[i] % 2 == 0)
        i = i * 2
    else
        i = i + 1
```

for this loop the number of iteration depend on the element of array
So the worst case would be when all elements are odd
O(n)

2> Omega Notation: It represents the lower bound of the running time or the best case complexity

for e the above example: The best case would be when all the elements

in the array are even in this case the time complexity would be $\Omega(\log n)$

3> Theta Notation : This gives us the avg time complexity of an algorithm

$\Omega(\log n) \leq \theta \leq O(n)$

Ans 2> for(i=1 to n) { i= i*2 }
the possible values of i after each iteration would be

1, 2, 4, 8, 16 .... 2^k

$2^k = n$

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

$K = \log_2 n$ (no of iterations)

the time Complexity would be

$O(\log_2 n)$ or $O(\log n)$

Ans 37 $T(n) = \begin{cases} 3T(n-1) & n > 0 \\ 1 & n = 0 \end{cases}$

Using backward Substitution

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

putting $(n = n-1$ in eqn (I)

$T(n-1) = 3T(n-2)$ — (II)

putting (II) in eqn (I) we get

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

$T(n) = 3*3(T(n-2))$ — (IV)

putting $(n = n-2)$ in eqn (I)

$T(n-2) = 3T(n-3)$ — (V)

using eqn (V) in eqn (IV)

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

So on generalization we get

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

$n - K = 0$

$n = K$

$K = n$

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

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

$T(n) = 3^n$

4⇒ $T(n) = \begin{cases} 1 & n \leq 0 \\ 2T(n-1) - 1 & n > 0 \end{cases}$

$T(n) = 2T(n-1) - 1$ — (i)

putting $n = n-1$ in eqn (i)

$T(n-1) = 2T(n-2) - 1$ — (ii)

Using Eqn (ii) in eqn (i)

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

$T(n) = 2 \times 2T(n-2) - 2 - 1$ — (iii)

putting $(n-2)$ in eqn (i)

$T(n-2) = 2T(n-3) - 1$ — (iv)

Using Eqn (iv) in eqn (iii)

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

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

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

$n - k = 0$

$n = k$

$= 2^k - 2^{k-1} - 2^{k-2} \ldots$

Solving this GP

$= 2^n - 2^{n-1} - 2^{n-2} \ldots -1$

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

$O(2^n)$

Ans 5:

```
int i=1, S=1;
while (S<=n){
        i++;
        S = S+i;
        print("#");
}
```

$S_K = 1 + 2 + 3 + 4 \ldots$

$S_K = n$
$\frac{K(K+1)}{2} = n$
$K^2 + K = 2n$
or
$K = \sqrt{2n}$

Time Complexity = $O(\sqrt{n})$

Ans 6:

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

$i * i = n$

$i^2 = n$

$i = \sqrt{n}$ ⇒ no of iteration

Time Complexity = $O(\sqrt{n})$

Ans 7:

```
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++
}
```

No. of iterations for i loop

$i = n - \frac{n}{2} = \frac{n}{2}$ $(O(n/2))$

No. of iterations for j loop

$j = 1, 2, 4, 8, 16 \ldots$

$j_k = 2^k$

$j_k = n$.

$2^k = n$

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

$k = \log_2 n$ no of iterations

Time Complexity $O(\log n)$

No. of iterations for K loop

same as 1 loop so

Time Complexity $O(\log n)$

total time complexity of nested loop

$O\left(\frac{n}{2} * \log n * \log n\right)$

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

Q6 =>

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

for the worst case this function will recurse infinitely

time Complexity O(infinity)

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

for loop i no of iterations n
for loop j no of iterations n
Nested loops so
Time Complexity $O(n^2)$ d

Ans 10) $n^k \leq ca^n$

$a^n + n^k \leq ca^n \rightarrow a^n$

$a^n + n^k \leq a^n(c-1)$

$\frac{a^n + n^k}{a^n} \leq c-1$

$c \geq 1 + \frac{n^k}{a^n} + 1$

$c \geq 2 + \frac{n^k}{a^n}$

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

$n=1$

$c \geq 2 + \frac{1}{1.5}$

$c \geq 3+1$

$c \geq 4$

Ans 11:

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

i = 1, 3, 6, 10, 15
j = 1, 2, 3, 4, 5, a_K

$a_K = K$

$S_K = \frac{K(K+1)}{2} = \frac{K^2+K}{2}$

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

$K^2+K <= 2n$

$K \cong \sqrt{2n}$

Time Complexity $O(\sqrt{2n})$ or $O(\sqrt{n})$

Ans 12

```
fib(int n):
if (n==1 || n==2)
    return 1
if (n==0)
    return 0
return ( fib(n-1) + fib(n-2))
```

$$T(n) = \begin{cases} T(n-1)+T(n-2) & n>2 \\ 1 & 1 \le n \le 2 \\ 0 & n=0 \end{cases}$$

Using backward substitution

$T(n) = T(n-1) + T(n-2)$ — (i)

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

$T(n-1) = T(n-2) + T(n-3)$ — (ii)

Using Eqn (ii) in eqn (i)

$T(n) = T(n-2) + T(n-3) + T(n-2)$ — (iii)

putting $n = n-2$ in eqn (i)

$T(n-2) = T(n-3) + T(n-4)$ — (iv)

Using Eqn (iv) in (iii)

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

$T(n) = 3T(n-3) + 2T(n-4)$ — (v)

putting $n = (n-3)$ in eqn (i)

$T(n-3) = T(n-4) + T(n-5)$ — (vi)

Using (vi) in Eqn (v)

$T(n) = 3T(n-4) + 3T(n-5) + 2T(n-4)$

$T(n) = 5T(n-4) + 3T(n-5)$

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

$T(0) = T(1) = 1$

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

$T(n-1) \approx T(n-2)$

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

$= 2\{2T(n-4) + c\} + c$

$= 4T(n-4) + 3c$

$= 8T(n-6) + 7c$

$= 16T(n-8) + 15c$

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

$n - 2k = 0$

$2k = n$

$k = n/2$

$T(n) = 2^{n/2} T(0) + (2^{n/2} - 1)c$

$= (1+c) \times 2^{n/2} - c$

$T(n) \propto 2^{n/2}$ (Lower bound

$T(n-2) \approx T(n-1)$

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

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

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

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

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

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

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

$T(n) \propto 2^n$ (upper bound)

$O(n) = 2^n$

Space Complexity $O(a1)$ bcz of no value is stored in memory

Ans 13)

$O(n \log n)$

```
int n
for( i=1 to n)
{   for( j=n to 1  j/2)
    {  print("#")
    }
}
```

$O(n^3)$

```
int i, j, k
for(i=1 to n)
{   for(j=1 to n)
    {   for(k=1 to k<=n)
        {   print("#")
        }
    }
}
```

Ans 14)

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + cn^2$$

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

ratio $= 5/16$

$$= \frac{n^2}{1 - 5/16} = O(n^2)$$

Ans 15?

```
int fun (int n){
    for (int i = 1 to int n)
    {
        for (int j = 1 to j <= n ), j += i)
        {
            print ("#")
        }
    }
```

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

$$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right)$$

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

Ans 16?

$$i = 2, 2^c, 2^{c^2}, 2^{c^3} \cdots 2^{c^{\log(\log(n))}}$$

$$2^{c^{\log_c(\log(n))}} = 2^{\log n} = n$$

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

Ans 18?

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

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

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

```
fun (arr[], key){
```

Ans→19→

```
for (i=0 to n-1){
    if (arr[i] = key){
        return
    }
}
return -1
}
```

Ans→20→ Iterative insertion sort

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

Recursive

```
Void insertionSort(int arr[], int n)
{   if (n<2)
        return;
    insertionSort(arr, n-1);
    last = arr[n-1];
    j = n-2;
```

```
while ( j>=0 && arr[j] > temp){
        arr[j+1] = arr[j];
        j = j-1 ; }
    }
    arr [j+1] = last ;
}
```

Ans>21>

Algo	Best	Avg	Worst
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Ans>22>

Algo	Inplace	Stable	Online
Bubble	✓	✓	x
Selection	✓	x	x
Insertion	✓	✓	✓
Merge	x	✓	x
Quick	x	x	x
Heap	✓	x	x

Ans>23> Iterative Binary Search

```
int BinarySearch (int arr[], int l, int r, int n)
{   while ( l <= r)
        int m = ( l+r )//2
        if (arr[m] == n),
            return m
```

```
        else if (arr[m] < n)
                l = m + 1
        else
                r = m - 1
    }
    return -1
}
```

Recursive

```
int BinarySearch (int arr[], int l, int R, int n)
{   if (l > r)
        return -1
    int m = (l + r) / 2
    if (arr[m] == n)
        return m
    else if (arr[m] < n)
        return BinarySearch (arr, m+1, r, n);
    else
        return BinarySearch (arr, l, m-1, n);
}
```

24) Recurrence Relation for Binary Search

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