TCS - 505 Name: Vijit Kala Sec: A

Assignment 1 Roll No: 2014941

Q1.) Asymptotic Notations are mathematical tools to represent the time & space complexities of algorithms for asymptotic analysis.

The different asymptotic notations :→

① Big O (O) :-

⇒ $f(n) = O(g(n))$

⇒ $g(n)$ is the "tight" upper bound of $f(n)$.

for ex:

$T(n) = 3n + 2$

$\Rightarrow O(n)$

② Big omega (Ω) :-

⇒ $f(n) = \Omega(g(n))$

⇒ $g(n)$ is the "tight" lower bound of $f(n)$.

for ex:

$f(n) = 4n + 3$

$g(n) = n$

⇒ $f(n) = \Omega(g(n))$

let's see if $f(n) \geq cg(n)$

⇒ $4n + 3 \geq cn_0$ for some $c > 0$ & $n_0 \geq 1$

when $c = 1$ & $n_0 = 1$ for any $n \geq 1$

$4n + 3 \geq n_0$ is true

Thus ⇒ $4n + 3 = \Omega(n)$

③ Theta(Θ) :-

⇒ Θ gives "tight" upper & lower bound of function.

ex:- $f(n) = 3n+2$ & $f(n) = \Theta(g(n))$ & $g(n) = n$

$3n+2 = \Theta(n)$

as

$3n+2 \geq 3n$ ⇒

$3n+2 \leq 4n$, for n

$\forall k_1 = 3$, $k_2 = 4$ & $n_0 = 2$

⇒ Complexity of $f(n)$ can be represented as $\Theta(n)$.

Q2.) Time complexity ⇒ $O(\log n)$

Q3.)

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

let

$T(n) = 3T(n-1)$ — (I)

⇒ $T(n+1) = 3T(n-2)$ — (A)

using value of $T(n-1)$ in (I)

⇒ $T(n) = 3^2 T(n-2)$ — (II)

⇒ $T(n-2) = 3T(n-3)$ — (B)

using value of $T(n-2)$ in (II)

⇒ $T(n) = 3^3 T(n-3)$ — (III)

⇒ Gen form:

$T(n) = 3^k T(n-k)$ — (IV)

$T(0) = 1$

⇒ $n-k = 0$

$n = k$

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

$\Rightarrow T(n) = 3^n$

$\Rightarrow O(3^n)$

Q4.)

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

let $T(n) = 2T(n-1) - 1$ — (I)

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

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

$\Rightarrow T(n) = 2^2 T(n-2) - 2 - 1$ — (II)

from (I)

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

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

$\Rightarrow T(n) = 2^3 T(n-3) - 2^2 - 2 - 1$ — (III)

$\Rightarrow$ Gen form:

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

$T(0) = 1 \Rightarrow n-k = 0 \Rightarrow k = n$

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

$\Rightarrow 2^n - 1 \left[2^{n-1} \left(\frac{1 - (1/2)^n}{1 - 1/2} \right) \right]$

$\Rightarrow 2^n - 1 \left[\frac{2^n}{2} \left(\frac{2^n - 1}{\frac{2^n}{\frac{1}{2}}} \right) \right]$

$\Rightarrow 2^n - 2^{n+1} \Rightarrow 1 \Rightarrow T(n) = 1 \Rightarrow O(1)$

Q5.) Time Complexity: $O(\sqrt{n})$

Q6.) Time Complexity: $O(\sqrt{n})$

Q7.) Time Complexity: $O(n \log^2 n)$

Q8.) Time Complexity: $O(n^2)$

Q9.) Time Complexity: $O(n \log n)$

Q10.)

$f(n) = n^k \quad k >= 1$

$g(n) = a^n \quad a > 1$

Since, exponential funcs. grow faster than polynomial functions

$O(n^k) < O(a^n)$ for all $k >= x \geq 1$, $a >= y$

⇒ Solving for x & y

Assuming $k = 2$ & $a = 2$

$f(n) = n^2 \qquad g(n) = 2^n$

Take log on both sides

$\log(f(n)) = 2 \log_2 n$

$\log(g(n)) = n \log 2$

$\rightarrow O(\log_2 n) \quad < \quad O(n)$

∴ Condition satisfies for all $k >= 2$ & $a >= 2$.

Q11.) $O(\sqrt{n})$ as i goes as follows:

1, 3, 6, 10, 15, 21 ...

$f(x) = \frac{n(n+1)}{2}$

$\Rightarrow$ 1, 3, 6 .... will stop when a_n becomes equal to or greater than n.

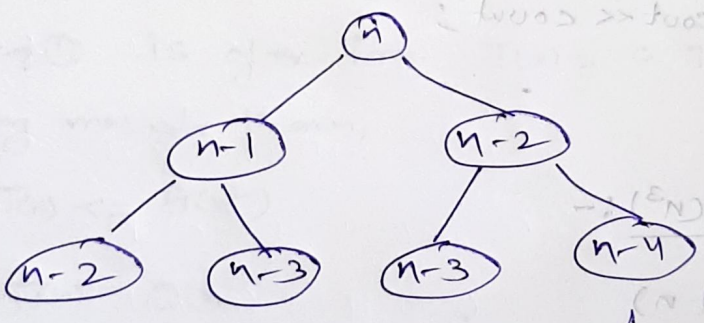
$\because \frac{n(n+1)}{2} = n_0$

$n \geq \sqrt{n_0}$

Q12.) Recurrance relation:

$$T(n) = T(n-1) + T(n-2) + 1 \quad \text{— (1)}$$

Solving using Tree method:



(1)

2^1

2^2

2^n

$T(n) = 1 + 2 + 4 + \ldots 2^n$

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

$\Rightarrow O(2^n)$

If we consider function call stack size it will have space complexity: O(n), else O(1).

Q13.)

(I) Complexity of (nlogn) :-

```
void fun(int n)
{
    int count = 0;
    for(int i = 0; i<n; i++)
    {
        for(int j=1; j<=n; j=j*2)
            count++;
    }
    std::cout << count;
}
```

(II) Complexity of (n^3) :-

```
void fun(int n)
{
    int count = 0;
    for(int i = 0; i<n; i++)
    {
        for(int j=1; j<=n; j++)
        {
            for (int k = 1; k+n/2 <= n; k++)
                count++;
        }
    }
}
```

(11b) Complexity of (log(log(n)):

```
void fun( int n)
{
    for( int i= 2; i<=n;   i*= i)
    {
        std:: cout << i << " ";
    }
}
```

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

⇒ Assuming $T(n/2) >= T(n/4)$

⇒ $T(n) <= 2T(n/2) + cn^2$ — (II)

Now eq (II) is of the form: $T(n) = a\,T(n/b) + \Phi(n)$

⇒ Applying master's theorem,

⇒ $T(n) <= \theta(n^2)$

⇒ $T(n) = O(n^2)$

and

$T(n) >= cn^2$

⇒ $T(n) >= \theta(n^2)$

⇒ $T(n) = \Omega(n^2)$

Since $T(n) = \Omega(n^2)$ & $T(n) = O(n^2)$

⇒ $T(n) = \theta(n^2)$

Q15) outer loop runs: n-times

inner runs: 1/i-times

$\Rightarrow 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \frac{1}{n+1}$-times

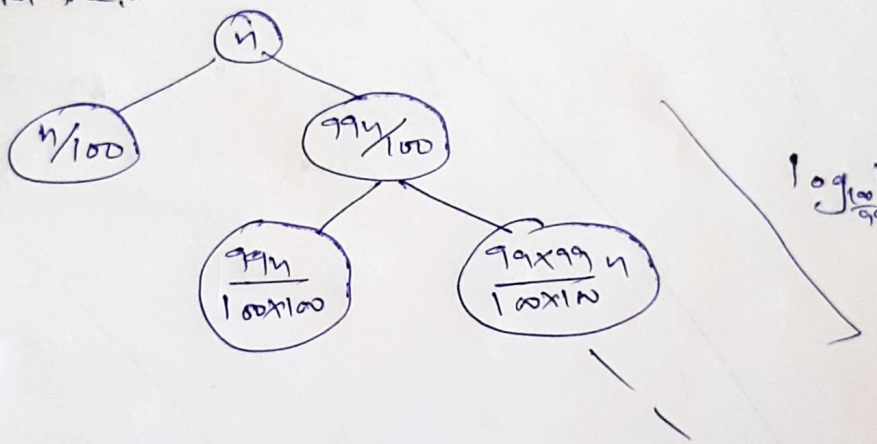
$\Rightarrow \ln(n)$

$\Rightarrow$ Time complexity: $O(n \lg n)$

Q16) The loop grows exponentially, time complexity: $O(\lg(\lg n))$

Q17) $T(n) = T(n/100) + T(99N/100) + N$

Recursion tree:-



at each level, we have to go through N values

therefore complexity is: $N \times \log_{\frac{100}{99}} N$

$\Rightarrow N * \frac{\lg_2 N}{\lg_2 \frac{100}{99}}$

$\Rightarrow$ neglecting $\lg_2 100/99$

$\Rightarrow$ Time Complexity: $(N \lg N)$

The analysis shows that, quicksort on average takes $O(N \lg N)$ ~~comparisons~~ time to sort n items.

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

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

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

Q19)
```
for(int i=0; i<n; i++){
    if (arr[i] == key){
        cout<< index
        break;
    }
}
```

Q20) Iterative :
```
void insertionSort(int arr[], int n){
    ~~int n =~~
    for(int i=0; i<n; i++)
    {
        int j=i;
        while (j>0 && arr[j] < arr[j-1])
        {
            swap(arr[j], arr[j-1]);
            j--;
        }
    }
}
```

Recursive:

```
void insertionSort(vector<int> arr, int i)
{
    if(i<=0)
        return;
    insertionSort(arr, i-1);
    int j = i;
    while(j>0 && arr[j] < arr[j-1]){
        swap(arr[j], arr[j-1]); j--;
    }
}
```

It is called online sorting algo because it doesn't have the constraint of having entire input available at the beginning like sorting algos like bubble or selectionsort. Can handle data piece by piece.

Q21) Quicksort: O(nlogn)
Merge ": O(nlogn)
Bubble: $O(n^2)$
Selection: $O(n^2)$
insertion: $O(n^2)$

Q22) Inplace: Bubble, Selection, Quick, Insertion
Stable: Bubble, Insertion, Merge
Online: Insertion

Q23) Iterative:

```
low = 0
high = n-1
while low <= high:
    mid = (low + high) // 2
    if key == a[mid]:
        print(mid)
        break
```

```
elif  key > a[mid]:
        low = mid+1
else :
        high = mid-1
```

Recursive:

```
def  BS (arr, low, high, key) :
    if (low > high) :
        return -1
    mid = (low+high) //2
    if  arr[mid] == key:
        return mid
    elif  arr[mid] > key:
        return  BS(arr, low, mid-1, key)
    else:
        return  BS(arr, mid+1, high, key)
```

Time Complexity of BS :
Iterative : O(log n)
Recursive : O(log n)

Space :
Iterative : O(1)
Recursive : O(log n)

Time Complexity of LS
Iterative : O(n)
Recursive : O(n)

Space :
Iterative : O(1)
Recursive : O(N)

Q24.

Recurrence relation for Bin. Search :

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