# Assignment 1

Name: Tushar Srivastav
University Rollno: 2014920
Section: B
Rollno: 60
Subject name and code: Design & Analysis of Algorithm (TCS 505)

Ans1: Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis. The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that don't depend on machine. Specific constants and doesn't require algorithms to be implemented and time taken by the programs to be compared.

Following are the asymptotic notations that are mostly used :-

(i) Θ Notation: The theta notation bounds a function from above & below, so its defines exact asymptotic behaviour.

(ii) Big O Notation: It defines and upper bound of an algorithm, it bounds a function only from above.

(iii) Ω Notation: Ω Notation provides an asymptotic lower bound.

For example consider Insertion Sort

It takes linear time in best case and quadratic time in worst case.

∴ We can say that Insertion sort have

$O(n^2)$
$\Theta(n^2)$ for worst case
$\Theta(n)$ " best "
$\Omega(n)$

Ans2) $\Theta(\log n)$.

Ans3) $T(n) = \begin{cases} 3T(n-1), & \text{if } n>0 \\ 1 & \text{otherwise.} \end{cases}$

$T(n) = 3T(n-1)$
$3(3T(n-2))$
$3^2 T(n-2)$
$3^3 T(n-3)$
$\vdots$
$3^n T(n-n) = 3^n$

Ans4 $T(n) = \begin{cases} 2T(n-1)-1, & \text{if } n \to 0 \\ 1, & \text{otherwise} \end{cases}$

$T(n) = 2T_1(n-1)-1$

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

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

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

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

$= 2^n T(n-n)-2^{n-1}-2^{n-8}-2^{n-3}\ldots 2^2-2^1-2^0$

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

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

$T(n)=1$

Ans5 ~~S=~~ S=S+i

If k is total number of iterations taken by the program, then while loop terminates.

$1+2+3+\ldots+k = [k(k+1)/2] > n$

$\therefore k = O(\sqrt{n})$

Ans6 $O(\sqrt{n})$

Ans7 j loop is executing logn times

k " " " " "

i " " " $n/2$ " i.e $n/2 \approx n$

Time complexity $= O(n \log 2n)$

Ans8 $O(n^2)$

Ans9 Inner loop will execute $(n+\frac{n}{2}+\frac{n}{3}+\ldots+n/n)$

$n(1+1/2+1/3\ldots+1/n)$

$\therefore$ It is equal to $\theta(n \log n)$

Ans10 $n^k$ $a^n$

$k>=1$ $a>1$

Taking $k=a=2$

$n^2$ $2^n$

$\therefore$ we can say $n^2 = O(2^k)$

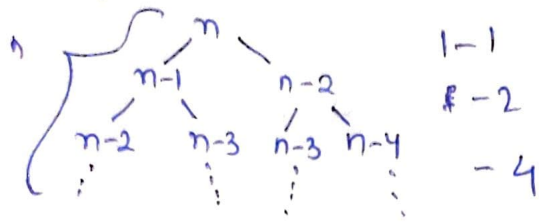
$\therefore n^k = O(a^n)$

Ans11 $O(\sqrt{n})$ Same logic as answer 5.

Ans 12 Recurrence Relation

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

Making Recurrence Tree




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

$a = 1 \quad r = 2$

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

$O(2^{n+1}) = O(2 * 2^n) = O(2^n)$

Space Complexity = $O(n)$

This is because maximum stack frame is equal to n only as function is called like this

$f(n-1) + f(n-2)$

$f(n-2)$ is called when we get the return value from $f(n-1)$

$\therefore$ It is equal to $O(n)$

Ans 13 nlogn

```
for(i=1 ; i<n; i++)
  { for (j=1; j <=n ; j = j+i)
      { printf ("*")
      }
  }
```

$n^3$

```
for (i=1; i<n ; i++)
  { for (j=1 ; j< n ; j++)
      { for (k=1; k<n; k++)
          { Pritf ("#");
          }
      }
  }
```

log logn

```
int fun (int n)
{ if (n<=2)
    return 1;
  else
  } return (fun (floor(sqrt(n))) + n);
```

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

We can assume

$T(n/2) >= T(n/4)$

$\therefore T(n) = 2T(n/2) + cn^2$

Applying Master's Method

$a=2 \quad b=2$

$k = \log_b a = \log_2 2 = 1$

$n^k = n$

$f(n) = n^2$

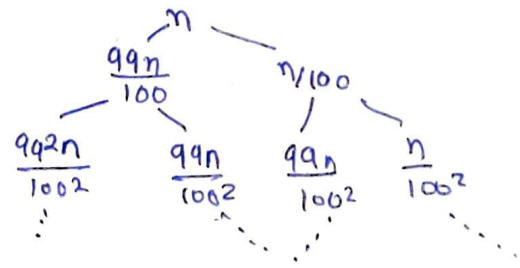
$\therefore$ It is $\theta(n^2)$

But as $T(n) <= \theta(n^2)$

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

Ans 16) If k is a constant greater than 1,
Then T.C = $O(\log \log n)$

Ans 17) $T(n) = T\left(\frac{99n}{100}\right) + T(n/100)$




If we take longer branch i.e $\frac{99n}{100}$

$T.C = \log_{100/99} n \approx \log n$

We can say that the base of log does not matter as it only a matter of constant.

Ans 18) a) 100 log logn logn $\sqrt{n}$ n logn! nlogn $n^2$ $2^n$ $2^{2n}/4^n$ n!

b) 1 loglogn $\sqrt{\log n}$ logn 2logn log2n n 2n 4n logn! nlogn $n^2$ $2(2^n)$ n!

(c) 96 $\log 8^n$ $\log_2 n$ 5n logn! n $\log_6 n$ $n\log_2 n$ $8n^2$ $7n^3$ $8^{2n}$ n!

Ans 19)

```
Linear Search (array, key)
    for i in array
        if value == key
            return i
```

Ans 20) Iterative Insertion Sort

```
insertionSort (arr, n)
    loop from i=1 to i=n-1
        Pick element arr[i] and insert it into
        sorted sequence arr[0....i-1]
```

Recursive Insertion Sort

```
insertionSort (arr,n)
{   if n<=1
      return
    recursively sort n-1 element
        insertion sort (arr,n-1)
    Pick last element arr[i] and insert it into sorted
    sequence arr[0...i-1]
}
```

Insertion sort considers one ~~##~~ input element per iteration and produces a partial solution without considering future elements.

∴ It is called online sorting algorithm.

Ans 20/21/22 Considering only 3 sorting algo now, as we got lectures till this.

| Algo | Best Case | Average Case | Worstcase | S.C | Stable | In place | Online |
|---|---|---|---|---|---|---|---|
| BubbleSort | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | $O(1)$ | ✓ | ✓ | x |
| Selection " | " | " | " | " | x | ✓ | x |
| Insertion " | $O(n)$ | " | " | " | ✓ | ✓ | ✓ |

Ans 23 Binary Search

```
A ← Sorted array
n ← Size of array
x ← Value to be searched
while x not found.
if upper bound < lower bound
    EXIT: x does not exist
Set midpoint = lowerbound + (upperbound-lowerbound)/2
  if A[midpoint] <x
        lowerbound = midpoint +1
```

if A[midpoint] > x

upperbound = midpoint - 1

if A[midpoint] = x

EXIT: x Found at midpoint

| | Time Complexity | Space Complexity |
|---|---|---|
| Linear | O(n) | O(1) |
| Binary Search (Recursive) | O(logn) | O(logn) |
| Binary Search (Iterative) | " | O(1) |

Ans 24 $T(n) = T(n/2) + c$