Date ___/___/_____

Saathi

Assignment 1

Q) What do you understand by Asymptotic notations. Define different Asymptotic notation with examples

- Asymptotic notation is a mathematical tool that describe the limiting behavior of a function as input grows to infinity
It is use to analyze the efficiency of algorithm and to compare their performance

There are three commonly used asympotitic notations

1) Big-O notation (O) : it represent the upper bound of the function's growth rate, i.e, how fast the function grows

It provide the estimate of the worst-case scenario

Example ⇒ if an algorithm has a running time of O(n), it means, that algorithm increase linearly with input size

O(n) ; if we double the size of the input list from 100 to 200, the running time of the algorithm will also be double.

2) Omega Notation (Ω) : It represent the lower bound of the function's growth rate i.e how slow the function grows.
It provide the **best case scenario.**

Example → Suppose we have an algorithm that takes an input size "n" and perform linear search to find element. In best case scenario, the element is found at the beginning of the list.
In this case the algorithm running time $\Omega(1)$

3) Theta Notation (θ) : It represent the right bound of the function's growth rate i.e, the function grows at the same rate both from above and below
It provide the **average case scenario**

Example → Suppose we have an algorithm that takes an input list of size (n) and perform a merge sort to sort a list. The worst case and best case have different running times, but the average-case scenario can be expressed $\theta(n \log n)$

if we double the input size from 100 – 200 the runtime of algorithms will increase approxim

Date ___/___/_____

Saathi

a factor of 2 log 2, which is manageable increase

Q.2) What should be time complexity of

for (i = 1 to n) & i = i + 2)

when i = 1

next turn i = 1 × 2

next turn i = 2 × 2

" " i = 2 × 2 × 2

" " i = 2 × 2 × 2 = 16

⋮

2^k [let assume i run 2^k times]

The process will stop at when

$i >= n$

as $i = 2^k$

$2^k >= n$

$2^k = n$

$K = \log_2 n$

$O(\log_2 n)$ it will execute log n time

Date ___/___/___

Saathi

Q) 5) What should be time complexity

```
int i=1, s=1
While (s<=n)
    { i++
      s = s+i
      printf ("##")
    }
```

Let assume the input size is n

- on first iteration

i	s
2	3
3	6
4	10
5	15
6	21
:	:
k^{th}	

After k^{th} iteration s will become $\frac{k(k+1)}{2}$

loop will stops after ;

$$\frac{k(k+1)}{2} > n$$

$$k^2 + k \geq 2n$$

$$k^2 > 2n.$$

$k = \sqrt{2n}$ after ignoring the constant.

So the loops run till $O(\sqrt{n})$

Q6 Time Complexity of -

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

let assume the input size is n

when

i	after loop i become
1	1
2	4
3	9
4	16
5	25

Date ___/___/___

after k^{th} time i become k^2

So loop will terminate at -

$$k^2 > n$$
$$k > \sqrt{n}$$

So the complexity $\Theta(\sqrt{n})$

Q) = Time Complexity of -

```
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++
}
```

The outermost loop execute $n/2$ to n times which is $O(n)$

The first nested loop execute $\log n$ times since it is multiplied by 2 at each

Similarly the second nested loop also execute $\log_2 n$ times

Date ___/___/_____

Saathi

Time complexity is $(n^2 \times \log n)$

Q)8 Time complexity

```
function(int n){

  if (n==1)
    return;
  for (i=1 to n)
    {
      for (j=1 to n){
          printf("*");
      }
    }
  function (n-3).

}
```

The function is Recursive that is called with argument $(n-3)$, it contain two nested loop that iterate over variable i and j.

i will iterate to n times and inner loop will also n times

Date ___/___/___

for each recursive call, the value of n times. The function will be called n/3 times recursive until n=1. The time complexity $O(n^2(n/3)^2)$

q) Time complexity

```
void function (int n)
{
    for (i to n) {
        for (j=1 to n) {j=j+1}
            printf("*")
    }
}
```

Let the size be n

When

i	j
1	n
2	n
3	n
.	.
.	.
.	.
k^{th}	n^k

let the loop will terminate at

$n^k >= n$

$O(n \log n)$

Q3 $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \end{cases}$

Backward substitution

put $n = n-1$

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

$T(n-1) = 3T(n-2) \quad \text{---(ii)}$

putting (ii) in eq

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

$T(n) = 9T(n-2)$

Now putting n to (n-2)

$T(n-2) = 9T(n-2-2)$

$T(n-2) = 9T(n-4)$

$T(n) = 9 \times 9T(n-4)$

$= 81T(n-4)$

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

$\therefore n - 2^k = 1$

$\Rightarrow n = 1 + 2^k$

$2^k = n-1$

$K \log 2 = \log(n-1)$

$K = \log(n-1)$

$n - 2k = 1$

$n-1 = 2k$

$k = \frac{n-1}{2}$

$3^{\frac{n-1}{2}} [n-2$

$3^{n-1/2} [2n-$

$3^{\frac{n-1}{2}}$

Now

$T(n) = 3^{\log(n-1)} T(n - 2\log(n-1))$

$= 3^{\log(n-1)} \cdot [n - 2\log(n-1)]$

$=$

Q) $T(n) = 2T(n-1) - 1$ if $n > 0$ otherwise

backward substitution

putting $n = (n-1)$

Date ___ / ___ / _____

Saathi

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

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

putting $T(n-1)$

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

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

Using Master Method

$a = 2$

$b = 1$

$c = \log_b(2) = 1$

$f(n) = 1$

$n^c = f(n)$

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