ASSIGNMENT-1

Rahul Negi, Section - C, 2014792, Class Roll No => 39

Q1 Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards

There are mainly three asymptotic notations:

a) Big-O notation - represents the upper bound of the running time of an algorithm. Thus, it gives the worst case complexity of an algorithm.

Eg $f(n) = O(g(n))$

it means that growth rate of $f(n)$ is asymptotically less than or equal to growth rate of $g(n)$.

b) Omega Notation (Ω notation) - represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

Eg $f(n) = \Omega(g(n))$

it means that growth rate of $f(n)$

is asymptotically greater than or equal to growth rate of $g(n)$.

c) Theta Notation (θ-notation) - encloses the function from above and below. Since it represents the upper and the lower band of the running time of an algorithm, it is used for analyzing the average case complexity of an algorithm.

Eg $f(n) = \theta(g(n))$ means that growth rate of $f(n)$ is asymptotically equal to growth rate of $g(n)$.

Q2 T.C of a for loop is no. of times (most inner loop has run) so,

$i = 1, 2, 4, 8, 16, \ldots, n$

$t_k = ar^{k-1}$

$n = 1 * 2^{k-1}$

$n = \frac{2^k}{2}$

$2n = 2^k$

$k = \log_2(2n)$

$k = \log_2 2 + \log_2 n$

$k = 1 + \log_2 n$

so, $O(\log_2 n)$ Ans

Q3

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

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

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

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

$$\vdots$$

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

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

$$= 3^n \times 1 \quad \text{since } T(0) = 1$$

$$= 3^n$$

$O(3^n)$ Ans

Q4

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

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

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

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

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

$$\vdots$$

$$\vdots$$

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

$= 2^n - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots$
$2^2 - 2^1 - 2^0$

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

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

$T(n) = 1$

$O(1)$ Ans

Q5

$i = 1$ 1 time
$s = 1$ 1 time

~~while (s<=n)~~

~~i++~~

$s = s + i$

1, 3, 6, 10, ...

~~sum of first n~~

$1 + 3 + 6 + 10 + \cdots n$ terms $= K$

let $S_n = n$

$$K = \frac{n(n+1)(n+2)}{6}$$

$K =$

$$n = \frac{K(K+1)(K+2)}{6}$$

$n =$

K

let $S_n = n$

K terms, Sum $= K$

Let first $= n$,

$$S_K = \frac{K(K+1)(K+1)}{6}$$

n

$$S_n = n\,\frac{(n+1)(n+2)}{6}$$

$$K = \frac{(n^2+n)(n+2)}{6}$$

$$K = \frac{n^2 + 2n^2 + n^2 + 2n}{6}$$

$$K = n^3$$

$O(n^3)$ Ans

Q@24

$$T(n) = T\left(\frac{n}{2}\right) + 1 \quad \text{— (1)}$$

Put $n = \frac{n}{2}$ in eqn (1)

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

$$= T\left(\frac{n}{4}\right) + 1 \quad \text{— (2)}$$

Put value of $T\left(\frac{n}{2}\right)$ from eqn (2) to (1)

$$T(n) = \left\{ T\left(\frac{n}{4}\right) + 1 + 1 \right.$$

$$= T\left(\frac{n}{4}\right) + 2 \quad \text{— (3)}$$

Put $n = n/4$ in eqn (1)

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1 \quad \text{— (4)}$$

Put value of $T\left(\frac{n}{4}\right)$ from eq (4) to (3)

$$T(n) = T\left(\frac{n}{8}\right) + 1 + 2$$

$$= T\left(\frac{n}{8}\right) + 3 \quad \text{— (5)}$$

$$T(n) = T\left(\frac{n}{2^K}\right) + K \quad - (6)$$

$$\frac{n}{2^K} = 1$$

$$2^K = n$$

Take $\log_2$ both sides

$$\log_2 2^K = \log_2 n \qquad [\log_a b = b \log a)$$

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

$$K = \log_2 n$$

put $K = \log_2 n$ in eqn (6)

$$T(n) = T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n$$

$$T(n) = T\left(\frac{n}{n}\right) + \log_2 n$$

$$= T(1) + \log_2 n$$

$$= 1 + \log_2 n$$

$$= O(\log_2 n)$$

023

Iterative Pseudo code

```
int binarySearch (int[] A, int x)
{
    int low = 0, high = A.length - 1;
    while (low <= high)
    {
        int mid = (low + high) / 2;
        if (x == A[mid])
        {
            return mid;
        }
        else if (x < A[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}
```

Recursive Binary Search Pseudo Code

```
int bS (int [] A, int low, int high, int x)
{
    if (low > high)
        return -1;
    int mid = (low + high) / 2;
    if (x == A[mid])
        return mid;
    else if (x < A[mid])
        return bS (A, low, mid - 1, x);
    else
        return bS (A, mid + 1, high, x);
}
```

	Time Comp	Space Comp
Linear Recursive	O(n)	
Linear Iterative		
BS Recursive	O (log(n))	(log(n))
BS Iterative	O (log(n))	O(1)

Q22

	Inplace	Stable	Online
Bubble Sort	✓	✓	
Insertion Sort	✓	✓	✓
Selection Sort	✓		
Merge Sort		✓	
Quick sort	✓		
Heap Sort	✓		

Q21

	Best Time Comp	Avg Time comp	Worst Time comp	Worst Space comp
Bubble sort	O(n)	O(n²)	O(n²)	O(1)
Selection sort	O(n²)	O(n²)	O(n²)	O(1)
Insertion sort	O(n)	O(n²)	O(n²)	O(1)
Quick sort	O(nlogn)	O(nlogn)	O(n²)	O(log n)

Q20

```
void recursive (int arr[], int n) {
    if (n<=1) return
    recursive (arr, n-1)
    int val = arr[n-1]
    int pos = n-2
    while (pos >=0 && arr[pos] > val) {
        arr[pos+1] = arr[pos]
        pos=pos-1
    }
    arr[pos+1] = val
}
```

insertion sort considers one input elem per iteration and produces a partial solution w/o considering future elements, so it's called online algo

selection sort selects sort an array by repeatedly finding min elem from unsorted part and putt it at beginning which requires access to the entire input, thus it is not online but a offline algorithm.

Q19

```
for (i=0; i<= arr.length(); i++)
{
    if (arr[length] < num ||
            arr[i] >= num)
    {
        if (arr(i) = = num)
            print (num at position, i)
        else
            print (num not present)
        break
    }
}
```

Q13

$(n \log n)$

```
#include <stdio.h>
void swap (int *a , int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
int partition (int array[], int low , int high)
{
    int pivot = array[high];
    int i= low-1 ;
    for (int j =low ; j< high; j++) {
    if (array[j] <=pivot ) {
      i++;
      swap (&array[i], &array[j]);
    }
}
swap (&array [i+1], &array [high]);
 return (i+1);
}
```

```
void quicksort(int array[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(array, low, high);
        quicksort(array, low, pi-1);
        quicksort(array, pi+1, high);
    }
}

void printArray(int array[], int size) {
    for (int i=0; i<size; ++i) {
        printf("%d ", array[i]);
    }
    printf("\n");
}

int main() {
    int data[] = {8, 7, 2, 1, 0, 9, 6};
    int n = sizeof(data) / sizeof(data[0]);
    quickSort(data, 0, n-1);
    printArray(data, n);
    return 0;
}
```

(n^3)

```
#include <stdio.h>
#define N 4
void mult (int mat1[][N], int mat2[][N],
           int res[][N])
{
    int i, j, k;
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            res[i][j] = 0;
            for (k=0; k<N; k++)
                res[i][j] += mat1[i][k] *
                             mat2[k][j];
        }
    }
}
int main()
{
    int mat1[N][N] = { {1,1,1,1},
                       {2,2,2,2},
                       {3,3,3,3},
                       {4,4,4,4} };
    int mat2[N][N] = { {1,1,1,1},
                       {2,2,2,2},
                       {3,3,3,3},
                       {4,4,4,4} };
    int res[N][N];
    int i, j;
```

```
mult (mat1, mat2, res);
printf("Result \n");
for (i=0; i<N; i++) {
    for(j=0; j<N; j++)
        printf ("%d", res[i][j]);
    printf ("\n");
}
return 0;
}
```

$$\left[\log (\log n) \right]$$

This program counts the number of prime numbers less than a non-negative number n.

```
class Solution {
    public int countPrimes (int n) {
        if (n<2) return 0;
        boolean [] nonPrime = new boolean[n];
        nonPrime[1] = true;
```

```
int numNonPrimes = 1;
for (int i = 2; i < n; i++) {
    if (nonPrime[i]) continue;
    int j = i * 2;
    while (j < n) {
        if (!nonPrime[j]) {
            nonPrime[j] = true;
            numNonPrimes++;
        }
        j += i;
    }
}
return (n-1) - nonPrimes;
}
}
```

Q14

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

$T(n) <= 2T(n/2) + cn^2$.

Apply master theorum to right hand side

This gives,

$T(n) <= \theta(n^2) \Rightarrow T(n) = \theta(n^2)$.

Also, $T(n) >= cn^2 \Rightarrow T(n) >= \theta(n^2)$

$\Rightarrow T(n) = \Omega(n^2)$.

since, $T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$

$$T(n) = \Theta(n^2)$$

Q15

Inner for loop dependent on i, so for each i we check no of times inner loop operating.

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

$$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n-1} - \log(n-1)$$

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

$$n\log(n-1) - \log(n-1)$$

$$n\log(n-1)$$

$$n\log n$$

$$O(n\log n)$$

Q16 i takes values

$2, 2^K, (2^K)^K = 2^{K^2}, (2^{K^2})^K = 2^{K^3}, \ldots,$

$2^{K \log_K(\log(n))}$.

last term must be less than or equal to n, and we have $2^{K \log_K(\log(n))} = 2^{\log(n)} = n$, which completely agrees with the value of our last term. So there are in total $\log_K(\log(n))$ many iterations, and each iteration takes constant amount of time to run, i.e.,

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