Assignment

Name: Aviral Rana
Setion: A
Subject: Design and Analysis of Algorithms
Roll no: 2015384

Q1: Asymptotic notations are the mathematical notations used to describe running time of an algorithm when the input tends towards a particular limiting value. These notations are generally used to determine the running time of an algorithm ~~comparison~~ and how it grows with the amount of input.

There are 5 types of asymptotic notations:

1) Big Oh (O) Notation: Big Oh notation defines an upper bound of an algorithm it bounds the function only from above.

formally:

$O(g(n)) = \{f(n)$: there exists positive constants c and n_0 such that $0 \le f(n) \le c g(n) \ \forall \ n \ge n_0\}$

2) Small Oh (o) Notation: We denote o-notation to denote an upper bound that is not asymptotically tight.

formally :

Q $o(g(n)) = \{ f(n)$: for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \leq f(n) < c g(n) \ \forall \ n \geq n_0$.

G

3) Big Omega (Ω) Notation : The Big Omega denote asymptotic Lower Bound

more formally :

$\Omega(g(n)) = \{ f(n)$: for any positive constant c and n_0. $0 \leq c g(n) \leq f(n) \ \forall \ n \geq n_0 \}$.

4) Small omega (ω) Notation : By analogy, ω notation is to Ω notation as o-notation is to O-notation. We use ω-notation to denote a lower bound that is not asymptotically tight. Formally :

$\omega(g(n)) = \{ f(n)$: for any positive $c > 0$, there exist $n_0 > 0$ such that $0 \leq c g(n) < f(n) \ \forall \ n \geq n_0 \}$.

5) Theta (Θ) Notation : The theta notation bounds the func. from above and below. So it defines exact asymptotic behavior.

formally :

$\Theta(g(n)) = \{ f(n)$: there exists positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \ \forall \ n \geq n_1 \}$.

Q2: $\Theta(\log n)$

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

By using back substitution.

$T(n) = 3\,T(n-1)$ — 1

$T(n-1) = \cancel{4} \cdot (3(T(n-2)))$ — 2

$T(n-2) = \cancel{3n}\,\cancel{4} \cdot (3\,T(n-3))$ — 3

Putting eqn 2 and 3 in 1

$T(n) = 3 \cdot 3 \cdot 3\,T(n-3)$.

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

Let $K = n$

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

$= O(3^n)$

Q4: $T(n) = \begin{cases} 2(T(n-1)) - 1 & n > 0 \\ 1 & n < 0 \end{cases}$

Using Back substitution.

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

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

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

$$\cancel{T(n) = 2.2.2[T(n-3) - 1]}$$

$$T(n) = 2.2T(n-2) - 2 - 1$$

$$T(n) = 2.2.2T(n-3) - 4 - 2 - 1$$

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

Let $k > n$

$$T(n) = 2^n T(n-n) - [1 + 2 + 4 + \ldots 2^{k-1}]$$

$$= 2^n T(1) - [1(2^{k-1} - 1)]$$

$$2^n - 2^{n-1} - 1$$

$$\Rightarrow O(2^n)$$

Q5: $O(n)$

Q6: $O(\sqrt{n})$

Q7: $O(n \log n \log n)$

Q8: The recurrence relation is :

$$T(n) = T(n-3) + n^2 \qquad n > 0$$

Solving by Back substitution: $\qquad n <= 0$

$T(n-3) = T(n-6) + (n-3)^2$

$T(n-6) = T(n-9) + (n-6)^2$

$\Rightarrow$ Putting back the values

$T(n) = T(n-9) + (n-6)^2 + (n-3)^2 + n^2$

$T(n) = T(n-3k) + (n-3(k-1))^2 + (n-3(k-2))^2 + \ldots n^2$

Let $3k = n$

$k = \frac{n}{3}$

$T(n) = T(n - 3 \times \frac{n}{3}) + \underbrace{n^2 + (n-3)^2 + \ldots + (n - 3(\frac{n}{3} - 1))^2}_{\text{total of } k \text{ terms}}$

$T(n) = T(1) + n^2 + (n-3)^2 + (n-6)^2 + \ldots + (n-n+3)^2$

$\Rightarrow$ Taking only higher order terms

n^2 will be obtained k times

$\Rightarrow$ n^2 will be obtained $\frac{n}{3}$ times

$T(n) = 1 + (n^2 + n^2 + n^2 + \ldots) + (Xn + Yn + Zn + \ldots)$

can be ignored

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

$T(n) = 1 + \frac{n^3}{3}$

$= O(n^3)$

Q9: Inner Loop runs as:

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

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

This sum will converge to $\log n$

Hence $n(\log n)$

~~Q10~~ $O(n \log n)$

Q10: $f(n) = n^K$ $K >= 1$

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

Exponential function grow faster than polynomial functions hence.

$$O(n^K) < O(a^n)$$

for values of $K \geq x$ and $a \geq y$

Let's calculate x and y

Let $K = 2$ and $a = 2$ as well.

$f(n) = n^2$, $g(n) = 2^n$

take log on both sides.

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

$$O(\log n) < O(n)$$

Hence for $K >= 2$ and $a >= 2$ the condition satisfies.

Q11 ⇒ $O(\sqrt{n})$, because the values of i go as follows: 1, 3, 6, 10, 15, 21.....

Also we know that

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

for the sum of series 1+2+3+4.....

So the series 1, 3, 6, 10, 15 will stop when a_n becomes equal to or greater than n

$$\therefore \frac{n(n+1)}{2} = n_0 \leftarrow \text{final value of } \underline{n}$$

$$n \simeq \underline{\sqrt{n_0}}$$

Q12 :

$$T(n) = \begin{cases} T(n-1) + T(n-2) + 1 & n \geq 2 \\ 1 & 0 \leq n < 2 \end{cases}$$

Assume time taken by $T(n-2) \simeq T(n-1)$

So

$$T(n) = 2T(n-2) + C \quad [\text{C is a constant}]$$

Solving we get:

$$T(n) = 2.2.2T(n-2.3) + 3C + 2C + 1C$$

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

$$n - 2k = 0 \Rightarrow k = \frac{n}{2}.$$

$$T(n) = 2^{\frac{n}{2}} T(0) + (2^{\frac{n}{2}} - 1)C$$

$$T(n) = O(2^{\frac{n}{2}}) \simeq O(2^n)$$

Space Complexity will be O(n) for the recursion stack. which go to size n in worst case.

Q13 : a) n log n

Program :

```
for (int i=0; i<n; i++)
{
    for (int j=1; j<n; j*=2)
    {
        cout << i << j;
    }
}.
```

b) n^3

Program :

```
for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
        for (int k=0; k<n; k+1)
            cout << i << j << k;
```

c) log(log(n))

Program :

```
for(int i=1; i<=n; i=i*2)
{
    for(int j=1; j<=i; j=j*2)
    {
        cout << i << j;
    }
}
```

```
void func(int n)
{       int c = 0;
    while(n >0)
    {
        c++;
        n/=2
    }
}
:
int x = func(n)
for (int i=1; i<=x; i = i*2)
{
    cout << i << x;
}
```

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

we can assume $T\left(\frac{n}{2}\right) \geq T\left(\frac{n}{4}\right)$

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

Using Master theorem.

$a = 2, \quad b = 2.$

$\log_b a \Rightarrow \log_2 2 \Rightarrow 1$

$f(n) = cn^2$

$n^k = n^2$

$\Rightarrow k = 2.$

$\because \log_b a < k$

$1 < 2$

$\Rightarrow$ Complexity $= O(n^k)$

$\Rightarrow O(n^2)$

15) $\because$ Inner loop will run n/i times.

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

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

~~n(?)~~ θ $n \log n$

$\Rightarrow$ ~~$O(n)$~~ $O(n \log n)$

16). Assuming pow (i, k) works in log(K) time.

we can express the runtime as

$$\sqrt[k]{\sqrt[k]{\sqrt[k]{\ldots\sqrt[k]{n}}}} <= \sqrt[k]{2}$$

$$\Rightarrow \quad n^{\frac{1}{k^m}} \leq 2^{\frac{1}{k}}$$

raise both sides to k^m

$$n^{\frac{k^m}{k^m}} \leq 2^{\frac{k^m}{k}}$$

$$n \leq 2^{k^{m-1}}$$

take Log $\quad \log(n) \leq k^{m-1} \underbrace{\log(2)}_{\text{a constant}}$

take Log again

$$\log(\log(n)) \leq (m-1) \underbrace{\log k}_{\text{a constant}}$$

$$\log(\log(n)) + 1 \leq m$$

$\because$ pow (i, k) takes log(K) time

Complexity $\Rightarrow$ $O(\log(K) \cdot \log(\log(n)))$.

Q18).

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

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

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

Q19).

```
for (int i=0; i<n; i++)
{
        if (arr[i] is equal to key)
            print index and break
        else
            continue.
}.
```

Q20): Iterative :
void insert(i

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

It is called online sorting algorithm because it does not have the constraint of having the entire input available at the beginning like other sorting algorithms as bubble sort or ~~xxx~~ selection sort. It can handle data piece by piece.

21) Quicksort : $O(n \log n)$
Mergesort : $O(n \log n)$
Bubblesort : $O(n^2)$
Selectionsort : $O(n^2)$
insertionsort : $O(n^2)$

22) Inplace: Bubblesort, Selectionsort, Quicksort, Insertionsort
Stable: Bublesort, Insertionsort, Mergesort
Online: Insertionsort

Q23) Iterative :

```
int low = 0, high = n-1
while ( low <= high) {
        mid = (low + high)/2
        if ( key = = a[mid])
                print mid and break
        if ( key > a[mid]) low = mid+1;
        else high = mid -1
}
```

Recursive

```
int BS(arr, low, high, key) {
    if (low > high) return -1;
    mid = (low + high)/2
    if (arr[mid] == key) return mid;
    if (arr[mid] > key) return BS(arr, low, mid-1);
    else return BS(arr, mid+1, high);
}
```

Time Complexity of BS:

Iterative = $O(\log n)$

Recursive = $O(\log n)$.

Time Complexity of Linear Search

Iterative = $O(n)$

Recursive = $O(n)$

Space Complexity of BS:

Iteration: $O(1)$.

Recursive: $O(\log n)$

Space Complexity of Linear Search

Iteration = $O(1)$

Recursive : $O(n)$

24) Recurrence relation for B.S:

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