12-08-2021

Assignment 1

1. what do you mean by Asymptotic Notation. Define different asymptotic notations.

1. Asymptotic notations are the mathematical representation of order of growth of any mathematical function.

There are mainly 3 types of notations:

1. Big O notation: It takes exact or upper bound.
2. Theta notation: It only takes the exact bound
3. Omega notation: It takes lower or exact bound.

** The word asymptotic means toward infinity

2. What should be the complexity of:

```
for (i=1 to n){
  i=i*2; }
```

Time complexity: $\Theta(\log(n))$

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

using Backward substitution method,

$T(n) = 3T(n-1)$ — (1)

$T(n-1) = 3T(n-2)$ — (2)

Putting value of $T(n-1)$ in the eq (1)

$T(n) = 3 \cdot 3T(n-2)$ — (3)

$T(n-2) = 3 \cdot T(n-3)$ — (4)

Putting value of $T(n-2)$ in eqn 3.

$T(n) = 3^3 T(n-3)$ — (5)

At some point

$T(n) = 3^k T(n-k)$ — (6)

$T(n-k) = 0$

$n = k$

Putting $k = n$ in eq 6

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

$T(n) = 3^n T(0) \Rightarrow T(n) = 3^n \cdot x1$

Time complexity : $O(3^n)$

4. $T(n) = \begin{cases} 2T(n-1) - 1 & , n > 0 \\ 1 & , n \leq 0 \end{cases}$

Using Backward substitution

$T(n) = 2T(n-1) - 1$ — (1)

$T(n-1) = 2T(n-2) - 1$ — (2)

Putting the value of $T(n-1)$ in (1)

$T(n) = 2[2T(n-2) - 1] - 1 \Rightarrow 4T(n-2) - 3$ — (3)

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

Putting the value of $T(n-2)$ in (3)

$T(n) = 4[2T(n-3) - 1] - 3 \Rightarrow 8T(n-3) - 7$

$T(n) = 8T(n-3) - 7$ — (4)

At some point

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

4. $T(n) = \begin{cases} 2T(n-1)-1 & , n>0 \\ 1 & , n \leq 0 \end{cases}$

using Backward substitution method

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

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

Putting Eq. 2 in Eq. 1

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

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

$T(n-2) = 2T(n-3) - 1 \quad -(4)$

Putting (4) in (3)

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

$T(n) = 2^3 T(n-3) - 2^2 - 2^1 - 1 \quad -(5)$

At some point:

$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} \text{ ---- } 2^0$

Now, assume

$n-k=0 \Rightarrow k=n$

$T(n) = 2^n T(n-n) - [2^0 + 2^1 \text{ ---- } 2^{k-2} + 2^{k-1}]$

$= 2^n T(0) - [2^0 + 2^1 \text{ ---- } 2^{n-2} + 2^{n-1}]$

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

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

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

Time complexity: $\Theta(2^n)$

Class Friends
DATE.....
PAGE.....

Q.5 What should be the complexity of:

```
int i = 1, s = 1;
while (s<=n){
  i++;
  s = s+i;
  print("#");
}
```

assume n=5

i=1	i=2	i=3	i=4	i=5
s=1	s=3	s=6	s=10	s=15

n=8 s=1, i=1
s=3, i=2 #
s=6, i=3 #
s=10, i=4 #

assume n=5

i=1	i=2	i=3
s=1	s=3	s=6
	#	#

Time complexity $\Theta(\lfloor \log(n) \rfloor)$

Q.6 Time complexity of

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

Time complexity $O(\sqrt{n})$

7. Time complexity of:

```
void function (int n){
int i, j, k, count=0;
for (i=n/2; i<=n; i++){          //O(n)
   for(j=1; j<=n; j=j*2)         //O(log n)
      for(k=1; k<n; k=k*2)       // O(log n)
          count++;
```

Time complexity: $O(n * \log n * \log n)$

8. Time complexity of:

```
function (int n) {
 if (n==1)              — O(1)
   return;              — O(1)
 for (i=1, to n) {      — O(n)   } O(n^2)
    for (j=1 to n) {    — O(n)   }
       printf ("*"),; }
}
function (n-3),
}
```

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

Using Backward substitution method

$T(n) = T(n-3) + c(n^2)$ — (1)

$T(n-3) = T(n-4) + c((n-1)^2)$ — (2)

Replace $T(n-1$

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

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

~~If (6)~~ Putting value of $T(n-3)$

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

9. Time complexity

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

The first loop has $\Theta(n)$

The second loop has $\Theta(\lceil n/i \rceil)$

$\Rightarrow \Theta(n * \lceil n/i \rceil)$

$\Rightarrow O(n^2)$

$i=x^2$

$j=x^2 \; x^3 \; x^4 \; x^5 \; x'$

* * * * * *

i = 1

j = 1 2 3 4 5

i = 2

j = 1 3 5

i = 3

j = 1 3

i = 4

j = 1 4

j = 5

j = 1

11. what is the time complexity of below code and why.

```
void fun(int n){
int j=1, i=0;
while(i<n){
i=i+j;
j++;
}
}
```

let n = 5

i	j	
0	1	
1	2	— ①
3	3	— ②
6	4	— ③

Loop run 3^2 times

12 write the recurrence relation for the recursive function that prints Fibonacci series. solve the recurrence relation to get time complexity of the program. what will be the space completely complexity of this program and why.

$$\begin{cases} T(n) = T(n-1) + T(n-2) + c \\ T(0) = T(1) = 1 \end{cases}$$

Solution : $T(n-1) \approx T(n-2)$ → assume

$T(n) = 2T(n-2) + c$ — (1)

$T(n-2) = 2T(n-4) + c$ — (2)

Putting (2) in (1)

$T(n) = 2(2T(n-4) + c) + c$

$T(n) = 4T(n-4) + 3c$ — (3)

$T(n-4) = 2T(n-6) + c$ — (4)

Putting (4) in (3)

$T(n) = 4(2T(n-6) + c) + 3c$

$T(n) = 8T(n-6) + 7c$ — (5)

So, $T(n) = 2^k T(n-2k) + (2^k - 1)c$ — (6)

- Now, $n - 2k = 0 \Rightarrow k = n/2$

Putting the value of k in eq. 6

$T(n) = 2^{n/2} \cdot 1 + (2^{n/2} - 1)c \Rightarrow 2^{n/2}(1 + c) - c$

Now assume $T(n-2) \simeq T(n-1)$

$T(n) = 2T(n-1) + c \quad -(1)$

$T(n-1) = 2T(n-2) + c \quad -(2)$

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

$T(n) = 4T(n-2) + 3c \quad -(3)$

$T(n-2) = 2T(n-3) + c \quad -(4)$

Putting 4 in 3

$T(n) = 4(2T(n-3) + c) + 3c$

$T(n) = 8T(n-3) + 7c$

so $T(n) = 2^k T(n-k) + (2^k - 1)c$

$n - k = 0 \quad n = k$

$T(n) = 2^n \cdot T(0) + (2^n - 1)c$

$= 2^n + 2^n c - c \Rightarrow 2^n(1+c) - c$

so,

for lower bound $T(n) \propto 2^{n/2}$

for upper bound $T(n) \propto 2^n$

$O(2^n)$ is the time complexity of recursive fibonacci series.

space complexity of fibonacci series is $O(n)$

13. write programs which have complexity:

(a) nlogn

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

(b) n^3

```
void three(int n){
for(int i=0; i< n; i++){
 for(int j=0; j<n ; j++){
   for(int k=0; k<n; k++){
   // some O(1) work; }
}}
```

14. solve the following recurrence relation

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

case 1:

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

$T(n) = 2T(n/2) + c(n^2)$ —①

$T(n/2) = 2T(n/4) + c(n/2)^2$ —②

Putting value of $T(n/2)$ in ①

$T(n) = 2[2T(n/4) + c(n/2)^2] + cn^2$

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

$T(n/4) = 2T(n/8) + c(n/4)^2$

$T(n) = 4[2T(n/8) + c(n/4)^2] + 2c(n/2)^2 + cn^2$

$T(n) = 8T(n/8) + 4c(n/4)^2 + 2c(n/2)^2 + c(n^2)$

$= 2^3 T(n/2^3) + 2^2 c(n/2^2)^2 + 2^1 c(n/2^1)^2 + c(n^2)$

$T(n) = 2^k T(n/2^k) + 2^{k-1} c(n/2^{k-1})^2 + 2^{k-2} c(n/2^{k-2})^2 + cn^2$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \boxed{k = \log_2 n}$

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + \frac{2^k}{2}\left(\frac{2n}{2^k}\right)^2 + \frac{2^k}{4}\left(\frac{4n}{2^k}\right)^2$ —— n^2

$= 2^{\log n} \cdot 1 + \frac{2^{\log n}}{2}\left(\frac{2n}{2^{\log n}}\right)^2 + \frac{2^{\log n}}{4}\left(\frac{4n}{2^{\log n}}\right)^2$ —— n^2

$= n + \frac{n}{2}\left(\frac{2n}{n}\right)^2 + \frac{n}{4}\left(\frac{4n}{n}\right)^2 +$ —— n^2

$= n + 2n + 4n$ —— n^2

$n[1 + 2 + 4 \text{ —— } n]$

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

Time complexity : $O(2^n)$

15. What is the time complexity of the following:

```
int fun (int n){
  for (int i=1; i<=n; i++){
    for (int j=1; j<n; j+=i){
      //some O(1) work; }
}}
```

i	j
① 1	1
1	2
② 2	1
2	—
③ — 3	1

n=3

$$\sum_{i=1}^{n} \sum_{j=1}^{n-1} (1)$$

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

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

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

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

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

16. for(int i=2; i<=n; i=pow(i,k))
{ //some O(1) work
}

n=33, k=2

i=2
i=4
i=16

$2 \quad 2^k \quad 2^{k^2} \quad 2^{k^3} \quad \cdots \quad 2^{k^{x-1}} = n$

~~k=2, n=8~~

~~$2^{k^x} = n \Rightarrow \log(n) = k^x \log 2$~~

~~$2^{k^x} = n \Rightarrow \log n = k^x \times \log k =$~~

$2^{k^x} = n \quad \log n = k^x \log 2$ ~~$x \log k = \log$~~

$x = \log_k \log n$

$O(\log \log n)$

17. $T(n) = \begin{cases} T((99n)/100) + n/100 & , n > 1 \\ 0 & , n = 1 \end{cases}$

Putting $n = \frac{99n}{100}$ in eq (1)

$$T\left(\frac{99n}{100}\right) = T\left(\left(\frac{99}{100}\right)^2 n\right) + \frac{n}{100} \quad - (2)$$

$$T(n) = \left(\left(\frac{99}{100}\right)^k n\right) + \frac{kn}{100} \quad - (3)$$

$$\left(\frac{99}{100}\right)^k n = 1 \qquad n = \left(\frac{100}{99}\right)^k$$

$$k = \log \frac{100n}{99}$$

$$T(n) = n\left(\frac{\log_{\frac{100}{99}} n}{100}\right) \Rightarrow T(n) = O(12 \log n)$$

18.

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

(b) $1 < \log\log n < \sqrt{\log n} < \log n < 2n < 4n < n < \log(2n) < 2\log n < n < n\log n = \log n! < 8n^2 < n!$

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

19.

```
for(i=0 to n)
  if (arr[i] == key)
     return i;
return -1;
```

20 Recursive

```
void insertion_sort (int arr[], int size)
{
if (size<1)
    return;
insertion_sort (arr, size-1);
int last = arr [size-1];
int j = size - 2;
while(j>=0 && arr[j]>last ){
 arr[j+1] = arr[j] ;
     j--; }
arr[j+1] = last;
}
```

Iterative

```
void insertion_sort(int arr[], int n)
{
for(int i=1; i<n; i++)
   int value = arr[i];
   int j=i;
while(j>0 && arr[j-1]>value)
{
 arr[j] = arr[j-1];
  j--;
}
arr[j] = value;
}
}
```

Online algorithm: An online algorithm is one that can process its input piece by piece in serial fashion.

23. Binary search (Recursive)

```
void b_search(int arr[], int size, int k)
{
int
void b_search( int arr[], int low, int high, int k) {
if (low <= high) {
   if (k ==
if (low <= high) {
  int mid = (low + high) / 2;
  if (arr[mid] == k)
      return mid;
  if (arr[mid] < key)
      b_search(arr, mid+1, high, k);
  else
    b_search (arr, low, mid);
  }
```

Iterative

```
int b_search (int arr[], int low, int high, int key)
{
while(low <= high){
 int mid= (low+high)/2;
 if (arr[mid] == key)
   return mid;
 if (arr[mid] < key)
    low = ~~big~~ mid +1;
 else
    high = mid;
}
}
```