Back To Course (/batchPage.php?batchId=238)

LIVE BATCHES

Learn

Quiz

1 2 3 4 5 6 7 8 9 10

Learn Quiz

Question 1 [1 Marks]

What is the time complexity of fun()?

```
int fun(int n)
{
  int count = 0;
  for (int i = 0; i < n; i++)
     for (int j = i; j > 0; j--)
        count = count + 1;
  return count;
}
```

A Theta (n)

✓ Theta (n^2)

C Theta (n*Logn)

D Theta (nLognLogn)

Explanation

The time complexity can be calculated by counting number of times the expression "count = count + 1;" is executed. The expression is executed 0 + 1 + 2 + 3 + 4 + ... + (n-1) times.

Time complexity = Theta(0 + 1 + 2 + 3 + .. + n-1) = Theta (n*(n-1)/2) = Theta(n^2)

Your submitted response was correct.

Previous

Next

Question 2 [1 Marks]

Let w(n) and A(n) denote respectively, the worst case and average case running time of an algorithm executed on an input of size n. which of the following is ALWAYS TRUE? (GATE CS 2012)
(A) $A(n) = \Omega(W(n))$
(B) $A(n) = \Theta(W(n))$
(C) $A(n) = O(W(n))$
(D) $A(n) = o(W(n))$

A A

B B

✓ C

D D

Explanation
The worst case time complexity is always greater than or same as the average case time complexity.

Your submitted response was correct.

Previous Next

Question 3 [1 Marks]

Which of the following is not O(n^2)?

A (15^10) * n + 12099

B n^1.98

✓ n^3 / (sqrt(n))

D (2^20) * n

Explanation

The order of growth of option c is $n^{2.5}$ which is higher than n^2.

Your submitted response was correct.

Previous Next

Learn

Quiz

1 2 3 4 5 6 7 8 9 10

Learn Quiz

Question 4 [1 Marks]

Which of the given options provides the increasing order of asymptotic complexity of functions f1, f2, f3 and f4?

```
f1(n) = 2^n

f2(n) = n^(3/2)

f3(n) = nLogn

f4(n) = n^(Logn)
```

✓ f3, f2, f4, f1

B f3, f2, f1, f4

C f2, f3, f1, f4

D f2, f3, f4, f1

Explanation

```
f1(n) = 2^n

f2(n) = n^(3/2)

f3(n) = nLogn

f4(n) = n^(Logn)
```

Except f3, all other are exponential. So f3 is definitely first in output. Among remaining, n^(3/2) is next.

One way to compare f1 and f4 is to take Log of both functions. Order of growth of Log(f1(n)) is Θ(n) and order of growth of Log(f4(n)) is Θ(Logn * Logn). Since Θ(n) has higher growth than Θ(Logn * Logn), f1(n) grows faster than f4(n).

Following is another way to compare f1 and f4.
Let us compare f4 and f1. Let us take few values to compare

Learn

Quiz

1 2 3 4 5 6 7 8 9 10

Learn Quiz

Question 5 [1 Marks]

Consider the following program fragment for reversing the digits in a given integer to obtain a new integer. Let n = D1D2...Dm

```
int n, rev;
rev = 0;
while (n > 0)
{
   rev = rev*10 + n%10;
   n = n/10;
}
```

The loop invariant condition at the end of the ith iteration is: (GATE CS 2004)

✓ n = D1D2....Dm-i and rev = DmDm-1...Dm-i+1

B n = Dm-i+1...Dm-1Dm and rev = Dm-1....D2D1

C n != rev

D n = D1D2....Dm and rev = DmDm-1...D2D1

Explanation

We can get it by taking an example like n = 54321. After 2 iterations, rev would be 12 and n would be 543.

Your submitted response was correct.

Previous Next

Question 6 [1 Marks]

Consider the following function

```
int unknown(int n) {

   int i, j, k = 0;

   for (i  = n/2; i <= n; i++)

       for (j = 2; j <= n; j = j * 2)

           k = k + n/2;

   return k;
```

A A

✓ B

C C

D D

Explanation

In the below explanation, '^' is used to represent exponent:
The outer loop runs n/2 or Theta(n) times.
The inner loop runs (Logn) times (Note that j is multiplied by 2 in every iteration).
So the statement "k = k + n/2;" runs Theta(nLogn) times.
The statement increases value of k by n/2.
So the value of k becomes n/2*Theta(nLogn) which is Theta((n^2) * Logn).

Your submitted response was correct.

Previous Next

Question 7 [1 Marks]

The recurrence equation

```
T(1) = 1

T(n) = 2T(n - 1) + n, n ≥ 2
```

evaluates to

✓ 2^{n + 1}- n - 2

B 2ⁿ - n

✗ 2^{n + 1} - 2n - 2

D 2ⁿ + n

Explanation

If draw recursion tree, we can notice that total work done is,
T(n) = n + 2(n-1) + 4(n-2) + 8(n-3) + 2^{n-1} * (n - n + 1)
T(n) = n + 2(n-1) + 4(n-2) + 8(n-3) + 2^{n-1} * 1

To solve this series, let us use our school trick, we multiply T(n) with 2 and subtract after shifting terms.

2*T(n) = 2n + 4(n-1) + 8(n-2) + 16(n-3) + 2^n

T(n) = n + 2(n-1) + 4(n-2) + 8(n-3) + 2^{n-1} * 1

We get

2T(n) - T(n) = -n + 2 + 4 + 8 + ... 2^n
T(n) = -n + 2^{n+1} - 2 [Applying GP sum formula for 2, 4, ...]

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

Question 8 [1 Marks]

Consider the following three claims
I (n + k)^m = θ(n^m), where k and m are constants
II 2^(n + 1) = 0(2^n)
III 2^(2n + 1) = 0(2^n)
Which of these claims are correct? (GATE CS 2003)

✓ I and II

B I and III

C II and III

D I, II and III

Explanation

```
(I)  (n+m)^k = n^k + c1*n^(k-1) + ... k^m = θ(n^k)

(II)  2^(n+1) = 2*2^n = O(2^n)
```

Your submitted response was correct.

Learn

Quiz

1 2 3 4 5 6 7 8 9 10

Learn Quiz

Question 9 [1 Marks]

Consider the following C code segment

```
int f (int x)

{

      if (x < 1)  return 1;

      else return (f(x-1) + g(x))

}
```

A Linear

✓ Exponential

C Quadratic

D Cubic

Explanation

f(n) = f(n-1) + g(n) ---- 1
g(n) = f(n-1) + g(n/2) ---- 2

Putting the value of g(n) in equation 1,
f(n) = 2*f(n-1) + g(n/2)

So, we can derive the below equation,
f(n) > 2f(n-1)
=> f(n) > 2*2*f(n-2) ---- because f(n) > 2*f(n-1), so, f(n-1) > 2*2*f(n-2).... so on
=> f(n) > (2^n)f(1) --- here '^' denotes the exponent.
So, f(n) > Theta(2^n)

So, option B is true, exponential growth for f(x).

Learn

Quiz

1 2 3 4 5 6 7 8 9 10

Learn Quiz

Question 10 [1 Marks]

What is the time complexity of following function fun()? Assume that log(x) returns log value in base 2.

```
void fun()

{

   int i, j;

   for (i=1; i<=n; i++)

      for (j=1; j<=log(i); j++)

         printf("GeeksforGeeks");
```

A Θ(n)

✓ Θ(nLogn)

C Θ(n^2)

D Θ(n^2(Logn))

Explanation

The time complexity of above function can be written as: Θ(log 1) + Θ(log 2) + Θ(log 3) + ... + Θ(log n) which is Θ (log n!)

Order of growth of 'log n!' and 'n log n' is same for large values of n, i.e., Θ (log n!) = Θ(n log n). So time complexity of fun() is Θ(n log n).

The expression Θ(log n!) = Θ(n log n) can be easily derived from following Stirling's approximation (or Stirling's formula) (https://en.wikipedia.org/wiki/Stirling%27s_approximation)

```
log n! = n log n - n + O(log(n))
```

Option (B) is correct.

Your submitted response was correct.