ANALYSIS OF ALGORITHMS

(SET: 2)

<u>Solutions</u>

Q.1 What is time complexity of fun( )?

```
int fun(int n)
{
   int count=0;
   for (int i= n; i> 0; i/=2)
      for(int j=0; j< i; j++)
          count+= 1;
   return count;
}
```

(a) O(n^2)
(b) O(nlog n)
(c) O(n)
(d) O(nlog n log n)

Solution: Option (c)

Explanation:
For a input integer n, the innermost statement of fun( ) is executed following times. So time complexity T(n) can be written as:
T(n) = O(n + n/2 + n/4 + … 1) = O(n)

The value of count is also: n + n/2 + n/4 + .. + 1

Q.2 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)
(b) Theta(n^2)

(c) Theta(n*log n) (d) Theta(nlog n log n)

Solution: Option (b)

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)

Q.3 The recurrence relation capturing the optimal time of the Tower of Hanoi problem with n discs is—

(a) T(n) = 2T(n – 2) + 2
(b) T(n) = 2T(n – 1) + n
(c) T(n) = 2T(n/2) + 1
(d) T(n) = 2T(n – 1) + 1

Solution: Option (d)

Explanation:
Following are the steps to follow to solve Tower of Hanoi problem recursively:
Let the three pegs be A, B and C. The goal is to move n pegs from A to C.

To move n discs from peg A to peg C:

move n-1 discs from A to B. This leaves disc n alone on peg A
move disc n from A to C
move n/1 discs from B to C so they sit on disc n

The recurrence function T(n) for time complexity of the above recursive solution can be written as following:

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

Q.4 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)=Ω(n)
(b) A(n)=theta(n)
(c) A(n)=O(n)
(d) A(n)=o(n)

Solution: Option (c)

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

Q.5 Which of the following is not O(n^2)?

(a) (15^10) * n + 12099
(b) n^1.98
(c) n^3 / (sqrt(n))
(d) (2^20) * n

Solution: Option (c)

Explanation:
The order of growth of option c is n^2.5 which is higher than n^2.

Q.6 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)

(a) f3, f2, f4, f1
(b) f3, f2, f1, f4
(c) f2, f3, f1, f4
(d) f2, f3, f4, f1

Solution: Option (b)

Explanation:
f1(n) = 2^n
f2(n) = n^(3/2)
f3(n) = nLogn
f4(n) = n^(Logn)
Except f2, all other are exponential. So f2 is definitely first in output. Among remaining, n^(3/2) is next.
Let us compare f4 and f1. Let us take few values to compare:
n = 32, f1 = 2^32, f4 = 32^5 = 2^25
n = 64, f1 = 2^64, f4 = 64^6 = 2^36
....
....

Q.7 Consider the following program fragment for reversing the digits in a given integer to obtain a new integer. Let $n = D_1D_2\ldots D_m$

```
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)
(a) $n = D_1D_2\ldots.D_{m-i}$ and $rev = D_mD_{m-1}\ldots D_{m-i+1}$
(b) $n = D_{m-i+1}\ldots D_{m-1}D_m$ and $rev = D_{m-1}\ldots.D_2D_1$
(c) n!= rev
(d) $n = D_1D_2\ldots.D_m$ and $rev = D_mD_{m-1}\ldots D_2D_1$

Solution: Option (a)

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

Q.8 What is the time complexity of the below function?

```
void fun(int n, int arr[ ])
{
   int i = 0, j = 0;
   for(; i < n; ++i)
      while(j < n && arr[i] < arr[j])
         j++;
}
```

(a) O(n)
(b) O(n^2)
(c) O(nlog n)
(d) O(n(log n) ^2)

Solution: Option (a)

Explanation:
In the first look, the time complexity seems to be O(n^2) due to two loops. But, please note that the variable j is not initialized for each value of variable i. So, the inner loop runs at

most n times. Please observe the difference between the function given in question and the below function:

```
void fun(int n, int arr[ ])
{
   int i = 0, j = 0;
   for(; i < n; ++i)
   {
      j = 0;
      while(j < n && arr[i] < arr[j])
         j++;
   }

}
```

Q.9 In a competition, four different functions are observed. All the functions use a single for loop and within the for loop, same set of statements are executed. Consider the following for loops:

(A) for(i = 0; i < n; i++)
(B) for(i = 0; i < n; i += 2)
(C) for(i = 1; i < n; i *= 2)
(D) for(i = n; i > -1; i /= 2)

If **n** is the size of input (positive), which function is most efficient(if the task to be performed is not an issue)?

(a) A
(b) B
(c) C
(d) D

Solution: Option (c)

Explanation:
The time complexity of first for loop is O(n).
The time complexity of second for loop is O(n/2), equivalent to O(n) in asymptotic analysis.
The time complexity of third for loop is O(logn).
The fourth for loop doesn’t terminate.

Q.10 What does it mean when we say that an algorithm X is asymptotically more efficient than Y?

(a) X will be a better choice for all inputs
(b) X will be a better choice for all inputs except small inputs
(c) X will be a better choice for all inputs except large inputs

(d) Y will be a better choice for small inputs

Solution: Option (b)

Explanation:
In asymptotic analysis we consider growth of algorithm in terms of input size. An algorithm X is said to be asymptotically better than Y if X takes smaller time than y for all input sizes n larger than a value n_0 where $n_0 > 0$.

Q.11 What is the time complexity of Floyd–Warshall algorithm to calculate all pair shortest path in a graph with n vertices?

(a) O(n^2logn)
(b) Theta(n^2logn)
(c) Theta(n^4)
(d) Theta(n^3)

Solution: Option (d)

Explanation:
Floyd–Warshall algorithm uses three nested loops to calculate all pair shortest path. So, time complexity is Thete(n^3).

Q.12 Consider the following functions:

f(n) = 2^n
g(n) = n!
h(n) = n^logn

Which of the following statements about the asymptotic behavior of f(n), g(n), and h(n) is true?
(a) f(n) = O(g(n)); g(n) = O(h(n))
(b) f(n) = **Ω** (g(n)); g(n) = O(h(n))
(c) g(n) = O(f(n)); h(n) = O(f(n))
(d) h(n) = O(f(n)); g(n) = **Ω** (f(n))

Solution: Option (d)

Explanation:
According to order of growth: h(n) < f(n) < g(n) (g(n) is asymptotically greater than f(n) and f(n) is asymptotically greater than h(n).

We can easily see above order by taking logs of the given 3 functions:
logn logn < n < log(n!) (logs of the given f(n), g(n) and h(n)).

<u>**Note**</u>: log(n!) = (nlogn)

Q.13 The minimum number of comparisons required to find the minimum and the maximum of 100 numbers is _________________.

(a) 147.1 to 148.1
(b) 145.1 to 146.1
(c) 140 to 146
(d) 140 to 148

Solution: Option (a)

Q.14 In the following C function, let n >= m.

```
int gcd(n,m)
{
 if (n%m ==0) return m;
 n = n%m;
 return gcd(m, n);
}
```

How many recursive calls are made by this function?
(a) theta(log n)
(b) Ω(n)
(c) theta(log log n)
(d) theta(sqrt(n))

Solution: Option (a)

Explanation:
Above code is implementation of the Euclidean algorithm for finding Greatest Common Divisor (GCD).

Q.15 Consider the following functions:

$f(n) = 3n^{\sqrt{x}}$
$g(n) = 2^{\sqrt{x \log_2 n}}$
$h(n) = n!$

Which of the following is true? (GATE CS 2000)
(a) h(n) is O(f(n))
(a) h(n) is O(f(n))
(c) g(n) is not O(f(n))
(d) f(n) is 0(g(n))

Solution: Option (d)

Explanation:
g(n) = 2^(sqrt(n)logn) = n^(sqrt(n))

f(n) and g(n) are of same asymptotic order and following statements are true:
f(n) = O(g(n))
g(n) = O(f(n))

(a) and (b) are false because n! is of asymptotically higher order than n^(sqrt(n))

Q.16 Consider the following three claims:

I) (n + k)^m =   (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)
(a) I and II
(b) I and III
(c) II and III
(d) I, II and III

Solution: Option (a)

Explanation:
(I) (n+m)^k = n^k + c1*n^(k-1) + ... k^m =   theta(n^k)
(II) 2^(n+1) = 2*2^n = O(2^n)

Q.17
```
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;
 }
```

What is the returned value of the above function? (GATE CS 2013)
(a) $\Theta(n^2)$
(b) $\Theta(n^2 \log n)$
(c) $\Theta(n^3)$
(d) $\Theta(n^3 \log n)$

Solution: Option (b)

Q.18 Consider the following two functions. What are time complexities of the functions?

```
int fun1(int n)
{
```

```
   if (n <= 1) return n;
   return 2*fun1(n-1);
}

int fun2(int n)
{
   if (n <= 1) return n;
   return fun2(n-1) + fun2(n-1);
}
```

(a) O(2^n) for both fun1() and fun2()
(b) O(n) for fun1() and O(2^n) for fun2()
(c) O(2^n) for fun1() and O(n) for fun2()
(d) O(n) for both fun1() and fun2()

Solution: Option (b)

Explanation:
Time complexity of fun1() can be written as:
T(n) = T(n-1) + C which is O(n)

Time complexity of fun2() can be written as:
T(n) = 2T(n-1) + C which is O(2^n)

Q.19 Consider the following segment of C-code:

```
int j, n;
 j = 1;
 while (j <= n)
       j = j*2;
```

The number of comparisons made in the execution of the loop for any n > 0 is:
Base of Log is 2 in all options.

(a) CEIL(logn) + 1
(b) n
(c) CEIL(logn)
(d) FLOOR(logn) + 1

Solution: Option (a)

Explanation:
We can see it by taking few examples like n = 1, n = 3, etc.

Q.20 Consider the following C-program fragment in which i, j and n are integer variables.

```
for (i = n, j = 0; i >0; i /= 2, j += i);
```

Let val(j) denote the value stored in the variable j after termination of the for loop. Which one of the following is true?

(a) val(j) = theta(log n)
(b) val(j) = theta(sqrt(n))
(c) val(j) = theta(n)
(d) val(j) = theta(nlog n)

Solution: Option (c)

<u>Note</u>: The semicolon after the for loop, so there is nothing in the body. The variable j is initially 0 and value of j is sum of values of i. i is initialized as n and is reduced to half in each iteration.

j = n/2 + n/4 + n/8 + .. + 1 = theta(n)