Quiz-1 (E): Design and Analysis of Algorithms (Spring-2024)

Solution

Question-1: Write the time complexities of the following C++ codes in terms of Big-O notation. Assume that there is no error in the codes.

Code	Time complexity

int isPrime (unsigned int n)
// return true if the argument value is prime and false otherwise
{
for (unsigned int i = 2; i * i < n; i++)
{
// if i is a factor, then not prime
if (0 == n % i)
return 0;
}
 	
// if we end loop without finding factor then number must be Prime
return 1;
}
	O($n^{0.5}$)

int Sum = 0;
for(int i=1; i<N; i += 2)
{
for(int j=N; j>0; j /= 3)
Sum++;
}
	O(nlogn)

int Fibonacci(int n)
{
if (n == 0)
return 1;
else if (n == 1)
return 1;
else
return Fibonacci(n-1) + Fibonacci(n-2);
}
	O(2^n)

Question-2: Which **famous** problem is solved by the following C++ code? Write the time complexity of the code in terms of Big-O notation.

```
int abc(int n)
{
    int prev1=0, prev2=1;
          for(int i=0; i<n; i++)
          {
                    int savePrev1 = prev1;
                    prev1 = prev2;
                    prev2 = savePrev1 + prev2;
           }
   return prev1;
}
```

Finding n-th term of Fibonacci series. O(n)

Question-3: Write an algorithm in pseudocode form to find the number of distinct elements in an array A[1…n] in O(n) time; where "n" is the array size. For example, if the array is {3,1,3,8,2,1,8,2}, the number of distinct elements is 4 (to be returned from the procedure) as the distinct elements are {1,2,3,8}. All the elements of the array are in the range [1,100]. Also, n >> 100 (n is significantly greater than 100). You are not allowed to use more than constant extra space i.e. use of a few variables is allowed. There will be no credit for a solution that will take more than linear time or more than constant extra space.

Note: Use of any implicit procedure call (for procedure not made by you) is not allowed.

```
Procedure DistinctElementsInArray
BEGIN
Inputs: List A[1…n]  and n
Output: count    {Number of  Distinct Elements}
    count = 0
    counts_Array[101]    {for storing the frequencies of distinct elements}
    FOR (i = 1 to 100)
        counts_Array[i] = 0
    END FOR

    FOR (i = 1 to n)
        counts_Array [ array[i] ] = counts_Array [ array[i] ] + 1
    END FOR
    FOR (i = 1 to 100)
        IF ( counts_Array[i] >  0) THEN
            count = count + 1
        END IF
    END FOR
    RETURN count
END PROCEDURE
```