PRG – ETSInf. THEORY. Academic Year 2013-14
Retake First Partial Exam – June, 23rd, 2014. Duration: 1h 50m.

1. $\boxed{\text{3 points}}$ We need to design a recursive method for determining if a given array of integers `v` contains the three initial numbers from the Fibonacci sequence. The size of the given array is greater than or equal to 3. The Fibonacci sequence begins with 0,1,1,2,3,5,8,...

   The first two values are fixed as 0 and 1, $f(0) = 0$ and $f(1) = 1$. The remaining ones are computed as the sum of the two previous values, i.e. $f(n) = f(n-1) + f(n-2)$.

   (a) (2.5 points). Write the method according to the above considerations.

   Solution: Solution by using an increasing traversal through the array:

   ```
   /** v.length >= 3, 2 <= i <= v.length */
   public static boolean is_fibo (int[] v, int i) {
     if (i==v.length) return true;
     if (v[0]!=0 || v[1]!=1) return false;
     else return (v[i]==v[i-1]+v[i-2]) && is_fibo(v,i+1);
   }
   ```

   Solution by using a decreasing traversal through the array:

   ```
   /** v.length >= 3, 1 <= i <= v.length-1 */
   public static boolean is_fibo (int[] v, int i) {
     if (i<=1) return (v[0]==0 && v[1]==1);
     else return (v[i]==v[i-1]+v[i-2]) && is_fibo(v,i-1);
   }
   ```

   (b) (0.5 points). According to the implementation of the method, what is the initial call in order to verify that the desired property holds for the whole array?

   Solution: Assuming that `v` is an array with at least three elements, the initial call for the recursive solution with ascending traversal is:

   `is_fibo(v,2);`

   And for the solution with descending traversal is:

   `is_fibo(v,v.length-1);`

2. $\boxed{\text{3 points}}$ The following recursive method, `invert()`, returns a `String` that is the reverse `String` of the one received as parameter. For example, given `"hello"` returns `"elloh"`.

   ```
   public static String invert( String s ) {
       if (s.length() <= 1) return s;
       else return invert(s.substring(1)) + s.charAt(0);
   }
   ```

   You have to analyse the temporal cost for each of the following two scenarios:

   1. Operations `substring(int)` and concatenation `+` have a constant running time given the length of the `String s`.
   2. Operation `substring(int)` has a linear temporal cost with respect to the length of `String s` and concatenation `+` has a constant running time given the total number of characters to be concatenated.

   For each scenario you have to do:

   (a) (0.25 points) Describe the input size for the problem and write the expression that best represents it.

   Solution: The input size is the length of `String s` in both scenarios, which is reduced at every recursive call. The expression is $n \equiv$ `s.length()`.

   (b) (0.5 points) Does the algorithm present significant instances? Describe worst and best cases if so.

Solution: There are no significant instances, in both scenarios the algorithm performs a traversal operation through the `String s`.

(c) (1.5 points) Write the recurrence equations for obtaining the temporal cost as function of the input size and solve them by applying the substitution method.

Solution:
In the scenario where `substring()` and concatenation have constant running time:

$$T(n) = \begin{cases} k' & \textit{if } n \leq 1 \\ T(n-1)+k & \textit{if } n > 1 \end{cases}$$

where k and k' are positive coefficients for a given time unit. If we apply the substitution method we get:

$$\begin{aligned} T(n) &= T(n-1)+k = T(n-2)+2k = T(n-3)+3k = \cdots = \\ &= T(n-i)+i\cdot k = \cdots = \\ &(\textit{trivial case}:\ n-i=1,\ i=n-1) \\ &= T(1)+(n-1)k = k'+nk-k \end{aligned}$$

In the scenario where `substring()` has a linear temporal cost and concatenation has constant running time:

$$T(n) = \begin{cases} k' & \textit{if } n \leq 1 \\ T(n-1)+kn & \textit{if } n > 1 \end{cases}$$

where k and k' are positive coefficients for a given time unit. If we apply the substitution method we get:

$$\begin{aligned} T(n) &= T(n-1)+kn = T(n-2)+k(n-1)+kn = T(n-3)+k(n-2)+k(n-1)+kn = \cdots = \\ &= T(n-i)+\textstyle\sum_{j=n-(i-1)}^{n} kj = \cdots = \\ &(\textit{trivial case}:\ n-i=1,\ i=n-1,\ n-(i-1)=2) \\ &= T(1)+\textstyle\sum_{j=2}^{n} kj = k'+k\Big(\frac{n(n+1)}{2}-1\Big) \end{aligned}$$

(d) (0.5 points) Write the result by using of the asymptotic notation.

Solution:
In the scenario where method `substring()` and the concatenation have constant running time: $T(n) \in \theta(n)$.
In the scenario where method `substring()` has linear temporal cost and the concatenation have constant running time: $T(n) \in \theta(n^2)$.

(e) (0.25 points) What is the best scenario regarding to the evaluation of the temporal cost? Support your answer.

Solution: Given the asymptotic behaviour the best scenario is when the `substring(int)` method has constant running time, because in this case the algorithm has a linear temporal behaviour with respect to the input size of the problem.

3. $\boxed{\text{4 points}}$ The following iterative algorithm returns an integer as the maximum sum of values stored in consecutive positions in a given array `a`.

```
/** Precondition: a.length >= 1 */
public static int method( int[] a ) {
    int n = a.length, max = a[0];
    for( int i=0; i <= n-1; i++ ) {
        int sum = 0;
        for( int j=i; j <= n-1; j++ ) {
            sum = sum + a[j];
            if ( sum > max ) max = sum;
        }
    }
    return max;
}
```

To be done:

a) (0.5 points) Describe the input size for the problem and write the expression that best represents it.

Solution: The input size is the length of the array. $n \equiv$ `a.length`.

b) (0.5 points) Does the algorithm present significant instances? Describe worst and best cases if so.

Solution: There are no significant instances because the algorithm performs a traversal across the array.

c) (2 points) Select a basic operation to be considered as critical instruction and according to it obtain an expression for the temporal cost function.

Solution:
Considering `sum = sum + a[j];` as critical instruction with constant running time equal to 1:

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

d) (1 point) Write the result by using of the asymptotic notation.

Solution: The temporal cost of the algorithm is quadratic with respect to the input size, $T(n) \in \theta(n^2)$.