PRG - ETSInf. THEORY. Academic year 2012/2013. Partial exam 1
April 22nd, 2013. Duration: 2 hours.

1. $\boxed{\text{4 points}}$ Let `a` be a one dimensional array of integers and `m` an integer. Write a `recursive` method for checking if there exists a pair of values `a[left]` and `a[right]` located in symmetric positions whose sum is equal to `m`. Following conditions must be met:

$$0 \leq \texttt{left} \leq \texttt{right} < \texttt{a.length} \qquad \texttt{left} == \texttt{a.length-1-right}$$

If there is at least one pair that meets the condition `a[left]+a[right]==m`, then the method must return the index of the leftmost value of that pair (the value of `left`). Otherwise -1 must be returned.

For example, if `m`=4 and `a`={1,4,5,9,6,0,-8} the result must be 1, if `m`=18 and `a` ={1,4,5,9,6,0,2} the result must be 3, and if `m`=25 and `a`={1,3,2,5,4,6} the method should return -1. We consider the method is called initially by taking into account the whole array in the three examples.

To be done:

a) Write the profile of the method you are going to implement, i.e., the header.
b) Describe the trivial and the general cases.
c) Write the method using Java.
d) First call to the method, i.e., how the method should be called from other one.
   Use a wrapper method if you want.

---

2. $\boxed{\text{3 points}}$ Let `m` be a squared two-dimensional array of integers, let `a` be a one dimensional array of integers, and it holds `m.length == a.length`. The following iterative method returns the position (row number) where the array `a` is inside `m`, otherwise it returns -1.

```
/**
 * Returns the position where "a" is inside "m" as a row or -1 otherwise.
 * PRECONDITION: m is a squared array and m.length == a.length
 */
public static int searchRow( int[][] m, int[] a )
{
    boolean found = false;
    int i = 0;
    while( i < m.length && !found ) {
        found = true;
        for( int j=0; j < m.length && found; j++ )
            found = ( m[i][j] == a[j] );
        if ( !found ) i++;
    }
    return ( found ) ? i : -1;
}
```

To be done:

1) Determine the input size of the problem and write the expression that represents it.
2) Choose a unit of measure for estimating the cost (program step or critical instruction), determine which elementary operations can be considered as critical instruction and select one of them.
3) Identify if there exist significant instances. In the affirmative case indicate the best and worst cases.

4) Obtain the mathematical expression as precise as be possible for the temporal cost function $T(n)$. If there exist significant instances then both functions must be obtained, one for the best case $T^b(n)$ and another for the worst case $T^w(n)$.

5) Express the results using asymptotic notation.

---

3. 2 points The following method checks if a slice (`a[b..e]`) of the array `a` is sorted in ascending order. The method splits the received slice into two halves, checks if each half is sorted and checks if the connection between both halves is in ascending order.

```
/** Returns true iff a[b..e] is in ascending order. Precondition: b<=e */
public static boolean isSorted( int[] a, int b, int e )
{
     if ( b == e )
        return true;
     else {
         int m = (b+e)/2;
         return isSorted(a,b,m) && isSorted(a,m+1,e) && a[m]<=a[m+1];
     }
 }
```

To be done:

i) The same steps described in the previous problem.

ii) Apply the substitution method for obtaining the temporal cost function.

---

4. 1 point Is it possible to modify the previous algorithm in order to have $\Omega(1)$ as the lower bound? If your answer is affirmative, which is the instruction to be modified and how?

Solution: The instruction to be modified is the one with the recursive calls. The comparison must be before the recursive calls:

```
return a[m]<=a[m+1] && isSorted(a,i,m) && isSorted(a,m+1,f);
```