Homework 7

Due date: Mar 5, 2020, 9:30am

Objective

- Deduce recurrence relations that describe the time complexity of recursively defined algorithms.
- Solve elementary recurrence relations.
- Use mathematical induction to prove the closed form of a recursively defined function.

Exercises

1. (5 points)
   Consider the sequence $(a_i)_i$ of numbers $(a_1, a_2, a_3, \ldots)$, which is defined by the following recurrence relation:

   $a_1 = 5$
   $a_n = 3a_{n-1}$ for all $n > 1$

   Calculate the values $a_2, a_3, \ldots, a_6$.

   $a_2 =$ _____ $a_3 =$ _____ $a_4 =$ _____ $a_5 =$ _____ $a_6 =$ _____

2. (10 points)
   Ackermann's function is defined as follows:

   $A(m, n) = 2n$ if $m = 0$
   $A(m, n) = 0$ if $m \geq 1$ and $n = 0$
   $A(m, n) = 2$ if $m \geq 1$ and $n = 1$
   $A(m, n) = A(m - 1, A(m, n - 1))$ if $m \geq 1$ and $n \geq 2$

   a) Find these values of Ackermann's function:
   $A(1, 0) =$ $A(0, 1) =$

   $A(1, 1) =$ $A(2, 2) =$

   $A(1, 2) =$ $A(1, 3) =$

   b) Determine a closed form of $A(1, n)$ for all $n \geq 1$.
   $A(1, n) =$

3. (10 points)
   The following questions refer to the algorithm `maximum` that determines the greatest element in the array A[p. . . q], where p is the lowest index in the array A and q is the highest index in A. (See also Homework 6, Exercise 4.)

```
int maximum (int[] A, int p, int q)

   if (p == q)
      return A[p];

   int k, l, max1, max2, max3;
   k = p + roundDown((q-p+2)/3);
   l = k + roundDown((q-p+2)/3);
   max1 = maximum(A, p, k-1);
   max2 = maximum(A, k, l-1);
   max3 = maximum(A, l, q);
   if (max1 >= max2 && max1 >= max3)
      return max1;
   else if (max2 >= max1 && max2 >= max3)
      return max2;
   else
      return max3;
```

   a) Use a recurrence relation to specify the runtime function for the algorithm `maximum`. You can assume that the number q - p + 1 of element of the array A, is a power of 3, i.e. $q - p + 1 = 3^k$ for some integer value $k \geq 0$. Make sure your function specification is formally correct.

   b) Determine the time complexity of the algorithm `maximum`. (no proof required)

4. (15 points)
   Consider the following recursive version of insertion sort. The algorithm assumes 0-based indexing. The original call to sort an array, say `data`, of size n is `insertionSort(data, n)`.

```
int insertionSort (ComparableObject[] a, int size)
   if (size <= 1)
      return;

   insertionSort(a, size - 1);

   ComparableObject obj = a[size - 1];
   int j = size - 1;
   while (j > 0  &&  a[j – 1] > obj)
      a[j] = a[j – 1];
      j--;
   a[j] = obj;
```

a) Use a recurrence relation to specify a runtime function of the recursive version of insertion sort. Assume the worst case scenario.

b) Determine a closed form of the recurrence relation.

c) What is the time complexity of the recursive version of insertion sort?

5. (10 points)
   Consider the following definition of t_{Hanoi}:

   $t_{Hanoi}(0) = 1$

   $t_{Hanoi}(n) = 2 \cdot t_{Hanoi}(n-1) + 1$ for $n > 0$

   Use mathematical induction to prove that

   $t_{Hanoi}(n) = 2^{n+1} - 1$ for $n \geq 0$.

Submission

Turn in a hard-copy with your solutions at the beginning of the class meeting on Mar 5. The solutions can be hand-written or typed.