CS210 Data Structures and Algorithms
Second Concept Assignment (20 marks)
Due date: October 25 at 11:55 pm
Important: No late concept assignments will be accepted

Please submit on OWL a pdf file with your solution to the assignment. You are encouraged to type your answers. If you decide to submit hand-written answers to the questions please make sure that the TA will be able to read your solutions. If the TA cannot read your answers you will not be given credit.

Remember that concept assignments must be submitted by the due date.

1. (1 mark) Consider a hash table of size $M = 7$ where we are going to store integer key values. The hash function is $h(k) = k \bmod 7$. Draw the table that results after inserting, in the given order, the following values: 5, 12, 1, 36, 26. Assume that collisions are handled by separate chaining.

2. (1 mark) Show the result of the previous exercise, assuming collisions are handled by linear probing.

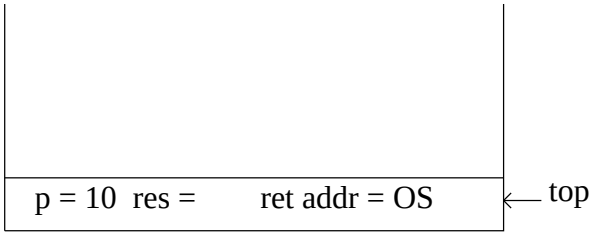
3. (3 marks) Repeat exercise (1) assuming collisions are handled by double hashing, using a secondary hash function $h'(k) = 5 - (k \bmod 5)$.

4. (4 marks) Consider the following algorithms.

Algorithm main()
  p $\leftarrow$ 10
  res = proc(p,2,1) (A1)
  print(res)

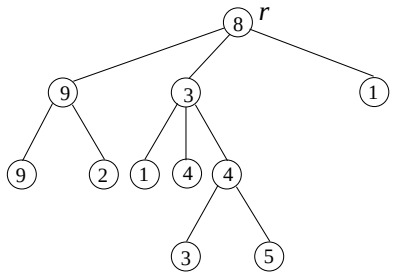
Algorithm proc(c, x, t)
  if $x = 0$ **then**
    return c (***)
  $m \leftarrow c - 5$
  if $m > t$ **then**
    return proc$(m, x - 1, t)$ (A2)
  else return proc$(2 * m, x - 1, t)$ (A3)

Draw the execution stack up to the moment just before the instruction marked (***) is about to be executed, i.e. run the algorithm and draw the activation records in the execution stack and stop the execution of the algorithm just before the value of c is to be returned. Show what the execution stack looks like at that moment. The addresses of statements where algorithms are invoked have been marked in the pseudocode as A1, A2, and A3. To help you out we have drawn the first activation record, for algorithm main.



Execution Stack

5. (6 marks) Write in detailed pseudocode, like the one used in class, an algorithm `find(r,k,v)` that receives as input the root `r` of a tree and two integer values `k` ≥ 0, and `v`, and it returns the number of nodes at level `k` storing the value `v`. For example, for the tree below with root `r` the invocation `find(r,2,4)` must return the value 2 as there are two nodes as level 2 that store the value 4, the invocation `find(r,3,9)` must return 0 as no node at level 3 stores the value 9, and `find(r,5,4)` must also return 0 as there are no nodes at level 5 in this tree.



For a node r let r.isLeaf() return *true* if r is a leaf and *false* otherwise. To get the value stored in a node r use r.getValue(). To access the children of a node r use the following pseudocode:

for each child c of r **do**

Note. A node does not store its level, so **you cannot** use in your algorithm something like r.getLevel() to get the level of node r.

Hint. In the initial call to the algorithm the value of the first parameter is the root of the tree and the value of the second parameter is the target level. Every time that the algorithm is invoked recursively the value of the second parameter must be updated how? [1]

Algorithm `find(r,k,v)`
In: Root `r` of a tree, value `k` ≥ 0, and value `v`
Out: Number of nodes at level `k` storing the value `v`

6. (5 marks) Compute the time complexity of the following algorithm. You must explain how many operations are performed per call, how many calls to the algorithm are made, how many operations the algorithm performs in total, and what the order of the time complexity is.

   Algorithm `func`(n) performs $c_1 \log n$ operations, where c_1 is a constant.

   Algorithm `algo`(r)
   In: Root r of a **proper binary tree** storing n nodes
   if r is a leaf **then return** n + `func`(n)
   else {
   $v \leftarrow$`func`(n)
   $v \leftarrow v$ + `algo`(r.leftChild()) + `algo`(r.rightChild())
   return v
   }

[1] *(Hint. It is not increased.)*