COSC 6320
Spring 2018
Homework 1

All work submitted should be done by the student alone. If you may use any reference material (books and journal articles), you must provide proper citation. You are not allowed to copy the solutions from any source. Collaboration between students is not allowed. On problems selected from the textbook, use the definitions from the book. Definitions from different books may be slightly different.

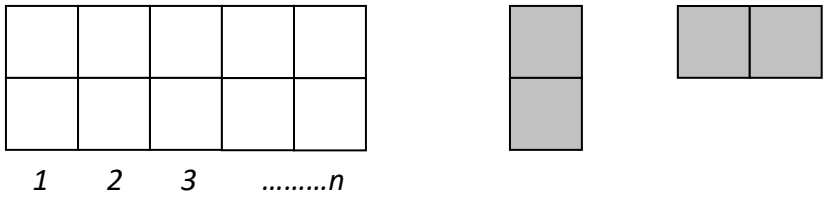
[1] (Generating Function) Solve the following recurrence relation using the generating function method.

$$h_n = 2h_{(n-1)} + n - 1, \text{ for } n > 0, \text{ and } h_0 = 0$$

[2] (Transformation) Solve the following recurrence relation by using range transformation.

$$n^2T(n) = 3(n-1)^2T(n-1) + 1, \quad T(1) = 1$$

[3] (Recurrence) We have a *2*x*n* chess board as shown in the figure below. We would like to cover such a board with *2*x*1* (vertical) dominoes or *1*x*2* (horizontal) dominoes completely. How many different ways can we do it? Derive a recurrence relation first and solve it using characteristic equation method.



[4] (Huffman's Tree) We talked about the Huffman's algorithm is great detail including the proof. This is a slight modification of the Huffman's problem. The longest code produced by the algorithm can be pretty bad (depending on the input data of course). Propose an efficient way to reduce the height of the Huffman tree by one. The code produced from this tree should still be valid and optimal under the restriction. This, of course, assumes that the height can be reduced.

[5] (ADT, Lists) The purpose of this assignment is to emphasize the separation of algorithm using a data type and the implementation of the data type. We would like to hide the implementation as much as possible (think object-oriented). Write two implementations of linear lists ("Array" and "Linked list") so that they are interchangeable, i. e., when you use the data structure in an application, you don't have to know how it is implemented. To test them, write two algorithms

(such as the purge function in the notes) that use the implementations. The function, say purge, should be able to run with either implementation of the list. The TA will test your program using a different operation (in addition to purge). You have to define a "position" that works for both cases and an "elemType" first before the list class.

List ADT Functions: p is a position, list is a list, x is an element

- list.get(p) returns the element if p is a valid position;
- list.add(x, p) add x to the list at position p (if p is a valid position);
- list.remove(p) delete element at position p (if p is a valid position);
- list.find(x) returns a position of x if it is in the list;
- list.next(p): returns the next position of p;
- list.first(): returns the first position of the list;
- list.empty(): is list empty?

Add additional functions and definitions as needed. You have to come up with one design that can be implemented in both cases. As for the elemType, try int first. Then if you can, generalize it to any other type. How?

This homework is now complete.