ICS 621: Analysis of Algorithms Fall 2024

Self Assessment Exam

Prof. Nodari Sitchinava

To determine if you have enough prerequisite knowledge, solve the following problems. You should work on the problems by yourself, without discussing them with anyone. This is not meant to comprehensively test your knowledge of the prerequisites, but rather the absolute basics that you need to know to undertand the concepts in this class. If you are struggling with these basic problems, you might want to take the undergraduate course in algorithms (ICS 311) before taking this course.

1 Linked List via Arrays (0 pts)

You are hired by a company to implement a singly-linked list that will hold integers. However, the boss of the company insists that you don't use any objects or structures in your implementation. You may use only integer arrays and individual integer variables.

Your implementaion should support the following operations:

- Insertion of the first integer element as the head of the list that will contain at most n elements throughout its lifetime.
- Keeping track of the 'current' element in the list (initially should be the head of the list)
- Advance 'current' to the next element in the list
- Reset 'current' to the head of the list
- Return the integer value of the 'current' element
- Insert a new value **following** the 'current' element in the list (should have no effect if n elements already had been inserted into the list)
- Delete the element **following** the 'current' element in the list (should have no effect if the 'current' element is the last one in the list)

The boss also insists that your implementation is efficient and **every operation should run in $O(1)$ time**, no matter what sequence of operations is performed. This is very important to the success of the company and they will not accept any code that is less efficient!

Fill in the missing code in the following definitions and explain why every one of your operations runs in $O(1)$ time (The code below is given in Java, but you may use C, C++, Java or Python):

```
public class List {

  private int ...

  private int[] ...

  /**********************************************************************
   * Initializes a linked list with integer 'val' as the head of the list
   * and identifies it as the current element
   * Ensures that no more than 'n' elements will be inserted into this list
   **********************************************************************/
  public List(int n, int val) {
```

```
    ...
  }

  /**************************************************************************
   * Advances the current element to the next element in the linked list
   *************************************************************************/
  public void advance() {
    ...
  }

  /**************************************************************************
   * Resets the current element to be the head of the list
   *************************************************************************/
  public void reset() {
    ...
  }

  /**************************************************************************
   * Returns the integer value of the current element
   *************************************************************************/
  public int value() {
    ...
  }

  /**************************************************************************
   * Deletes the element that follows the current element in the list
   * Has no effect if the current element is the last one in the list
   *************************************************************************/
  public void deleteNext() {
    ...
  }

  /**************************************************************************
   * Inserts 'val' into the list as the element that follows the current
   * one. Has no effect if n elements had already been inserted into the list
   * (not counting deletions)
   *************************************************************************/
  public void insertNext(int val) {
    ...
  }
}
```

2 Number of Tree Edges (0 pts)

Prove ***using induction*** that the number of edges in every n-node binary search tree is exactly $n - 1$.

If you need to review inductive proofs, I recommend Jeff Erickson's writeup in the Appendix I of his book.

3 Runtime Analysis (0 pts)

Characterize the running time of the following algorithms in terms of n using Θ notation.

```
TripleLoop(A, n)
  for k = 1 to 10
    for i = 1 to n
      for j = n downto i
        A[i] = A[j] + 1
```

$\Theta($_____$)$

Justification:

```
1: function FOO(n)
2:     if n < 2
3:         return 1
4:     else
5:         for i = 1 to n
6:             x = x + 1
7:         for i = 1 to 2
8:             x = x + FOO(n/4)
9:         return x
```

$\Theta($_____$)$

Justification:

4 Almost Palindromes (0 pts)

A *palindrome* is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, `noon`, and `aibohphobia` (fear of palindromes).

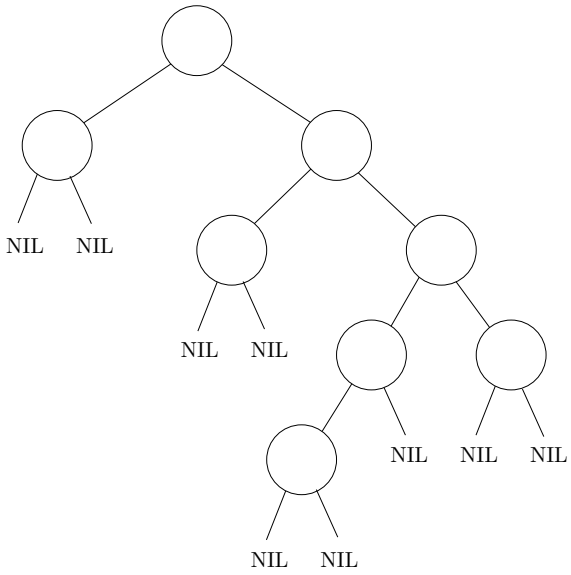
You may assume that in the problems below, an input string is given as an array of characters. For example, input string `noon` is given as an array $s[1..4]$, where $s[1] =$ `n`, $s[2] =$ `o`, $s[3] =$ `o`, and $s[4] =$ `n`.

(a) Give a ***recursive*** algorithm that tests if the input string is a palindrome. Write down the pseudocode and use induction to prove that your algorithm correctly identifies palindromes. Analyze your algorithm's running time.

(b) Sometimes removing a single character from the string, turns it into a palindrome. For example, if we remove `l` from the `revolver`, we will get a palindrome. We call such strings ***almost palindromes***. Write down the pseudocode for an algorithm that tests if the input string is an almost palindrome. Analyze your algorithm's running time.

5 Binary Search Trees (0 pts)

(a) - (5 pts) Assign the keys 3, 7, 12, 21, 27, 46, 49, 51 to the nodes of the binary search tree on the right so that they satisfy the binary search tree property.



(b) - (5 pts) Explain why this tree cannot be colored to form a legal red-black tree.

(c) - (5 pts) The binary search tree can be transformed into a red-black tree by performing a single rotation. Using the keys from part (a), draw the resulting red-black tree and label each node with "red" or "black".

6 Graph ADT (0 pts)

(a) Implement a Directed Graph abstract data type using adjacency list representation in a programming language of your choice from among Java, C, C++, or Python. Your ADT should support insertions and deletions of arbitrary nodes, as well as accessing a given node's outgoing neighbors.

(b) Consider the following pseudocode:

```
1:  function ProcessGraph(G)
2:      for each vertex u ∈ G.V
3:          u.color = WHITE
4:          u.π = NIL
5:      time = 0
6:      for each vertex u ∈ G.V
7:          if u.color == WHITE
8:              Visit(G, u)
9:  function Visit(G, u)
10:     time = time + 1
11:     u.d = time
12:     u.color = GRAY
13:     for each v ∈ G.Adj[u]
14:         if v.color == WHITE
15:             v.π = u
16:             Visit(G, v)
17:     u.color = BLACK
18:     time = time + 1
19:     u.f = time
```

(i) Analyze the running time of the above pseudocode.

(ii) Do you recognize this code? What does it do?

(iii) Implement the above pseudocode using your Graph ADT from part (a).

7 Coin tossing (0 pts)

Consider a random experiment of tossing a biased coin three times. We assume that the tosses are independent of each other and the probability of a head is 0.3. Let X be the random variable whose value represents the number of heads obtained.

(a) List the possible outcomes of three coin tosses. What are the probabilities $Pr[X = 0]$, $Pr[X = 1]$, $Pr[X = 2]$, $Pr[X = 3]$?

(b) Compute $E[X]$ using the definition of expected value (i.e., without the use of indicator random variables). *Show your work.*

(c) Let X_i be an indicator random variable, whose value is 1 if the i-th toss resulted in heads. Define X in terms of X_i's and compute $E[X]$ using this expression. *Show your work.*