CS 362, HW4

Prof. Jared Saia, University of New Mexico

1. Your boss asks you to decide which of two algorithms to use in a new software system. The runtimes of the two algorithms are given by the following recurrences (remember that when the base case of a recurrence is not given, assume $T(c) = \Theta(1)$ for any constant c):

   - Algorithm 1: $T(n) = 5T(n/2) + n$
   - Algorithm 2: $T(n) = 3T(n/2) + n^2$

   Which algorithm has the better asymptotic cost? Justify your answer by solving both recurrences (using recursion trees) and comparing the solutions.

2. A frog is jumping across a line of lily pads. It starts at lily pad 1. When the frog is at lily pad i for any $i \geq 1$, it jumps to lily pad $i + 1$ with probability $1/2$ and to lily pad $i + 2$ with probability $1/2$.

   (a) Let $p(i)$ be the probability that the frog ever visits lily pad i, for any $i \geq 1$. Write a recurrence relation for $p(i)$. Don't forget the base case(s).

   (b) Use annihilators to solve for a general solution to your recurrence relation.

   (c) Use the base case(s) of your recurrence to solve for an exact solution.

   (d) Now, let X be a random variable giving the number of lily pads between lily pad 1 and n that the frog visits, for some fixed number n. Compute $E(X)$ by using: linearity of expectation, indicator random variables, and your solution to the recurrence $p(i)$ that you found above.

3. *Silly-Sort* Consider the following sorting algorithm

```
Silly-Sort(A,i,j)
  if A[i] > A[j]
    then exchange A[i] and A[j];
  if i+1 >= j
    then return;
  k = floor((j-i+1)/3);
  Silly-Sort(A,i,j-k);
  Silly-Sort(A,i+k,j);
  Silly-Sort(A,i,j-k);
```

(a) Argue (by induction) that if n is the length of A, then Silly-Sort(A,1,n) correctly sorts the input array A[1...n]

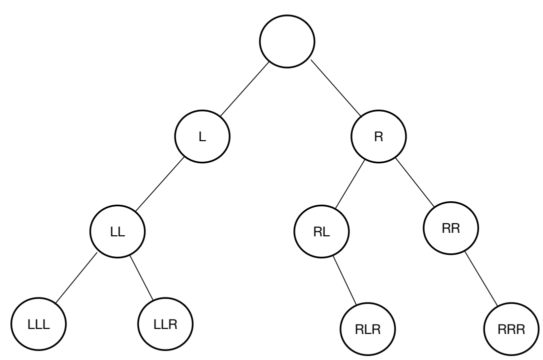
(b) Give a recurrence relation for the worst-case run time of Silly-Sort and a tight bound on the worst-case run time

(c) Compare this worst-case runtime with that of insertion sort, merge sort and quicksort.

4. *Note: In this problem, you'll be writing - but not solving - a recurrence relation over a data structure. When we get to dynamic programming in class, we'll see how to solve these types of recurrences.*

   Consider a rooted binary tree with nodes are labelled as follows. The root node is labelled with the empty string. Then, any node that is a left child of a node with name σ receives the name σL and any node that is the right child of that node receives the name σR.

   Give a recurrence relation returning the number of R's in all labels of all nodes. For example, the following tree has 10 R's.



   Hint: For a node v, let $f(v)$ be the number of R's in the tree rooted at v, if the naming started at v. Also, let $\ell(v)$ (resp. $r(v)$) be the left (resp. right) child of v if it exists or NULL otherwise. Finally, let $s(v)$ be the number of nodes in the subtree rooted at v and assume this value is stored at each node. Now write a recurrence relation for $f(v)$. Don't forget to include the base case and to test it on some examples.