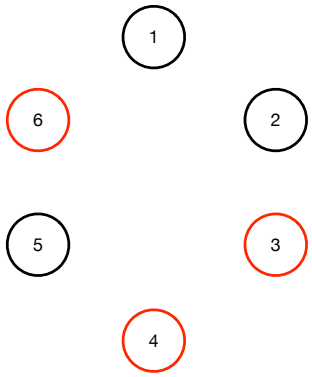
CS 362, HW2

Prof. Jared Saia, University of New Mexico

Due: Feb. 18

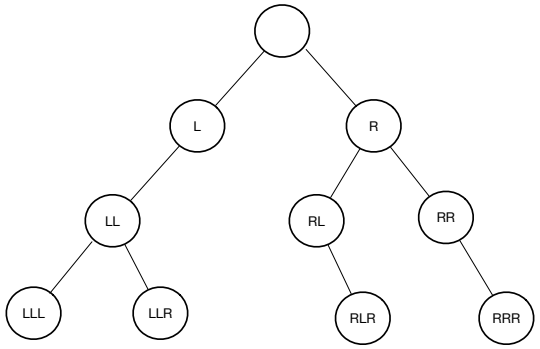
1. A cat hops on posts arranged in a circle. There are $2n$ posts, with n red and n black. The cat can start at any post, and always hops to the next post in the clockwise direction, until it visits all posts. It "wins" if, at every point during its trip, the number of red posts visited so far is always at least the number of black posts visited so far. In the figure below, the cat wins by starting at post 3, but loses if it starts at any other post.



   Prove that the cat can always win, if it starts on the right post. Prove this by induction on n for $n \geq 1$. Let your IH be that the cat can always win in the case where there are between 2 and $2(n-1)$ posts.

2. 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 give a recurrence relation for $f(v)$.

3. Consider the recurrence $f(n) = 3f(n/2) + \sqrt{n}$

   (a) Use the Master method to solve this recurrence

   (b) Now use annihilators (and a transformation) to solve the recurrence. Show your work. (This is perhaps stating the obvious, but note that your two bounds should match)

4. Consider the following function:

```
int f (int n){
  if (n==0) return 2;
  else if (n==1) return 5;
  else{
    int val = 2*f (n-1);
    val = val - f (n-2);
    return val;
  }
}
```

   (a) Write a recurrence relation for the *value* returned by f. Solve the recurrence exactly. (Don't forget to check it)

   (b) Write a recurrence relation for the *running time* of f. Get a tight upperbound (i.e. big-O) on the solution to this recurrence.

5. *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, heapsort and quicksort.

6. There are two bins: Bin 1 initially has 3 white balls and 1 red ball. Bin 2 has 4 white balls. In every round, a ball is selected uniformly at random from each bin and these two balls are swapped. Let $p(n)$ be the probability that the red ball is in bin 1 at the beginning of the n-th round.

   (a) Write a recurrence relation for $p(n)$.

   (b) (8 points) Use "guess and check", and proof by induction to solve this recurrence. Don't forget to label BC, IH and IS and clearly say where you are using the IH. Hint: Compute the first few values of $p(n)$ to spot the pattern.

   (c) If this is repeated for m rounds, what is the expected number of rounds that the red ball is in bin 1?

7. **Primes and Probability**. In this problem, you will use the following facts: (1) any integer can be uniquely factored into primes; (2) the number of primes less than any number m is $\Theta(m/\log m)$ (this is the prime number theorem).

   We will also make use of the following notation for integers x and y: 1) $x|y$ means that x "divides" y, which means that there is no remainder

when you divide y by x. and 2) $x \equiv y \pmod{p}$ means that x and y have the same remainder when divided by p, or in other words, $p|(x-y)$.

(a) Show that for any positive integer x, x factors into at most $\log x$ unique primes. Hint: 2 is the smallest prime.

(b) Let x be some positive integer and let p be a prime chosen uniformly at random from all primes less than or equal to m. Use the prime number theorem to show that the probability that $p|x$ is $O((\log x)(\log m)/m)$.

(c) Now let x and y both be positive integers less than n, such that $x \neq y$, and let p be a prime chosen uniformly at random from all primes less than or equal to m. Using the previous result, show that the probability that $x \equiv y \pmod{p}$ is $O((\log n)(\log m)/m))$.

(d) If $m = \log^2 n$ in the previous problem, then what is the probability that $x \equiv y \pmod{p}$. Hint: If you're on the right track, you should be able to show that this probability is "small", i.e. it goes to 0 as n gets large.

(e) Finally, show how to apply this result to the following problem. Alice and Bob both have large numbers x and y where x and y are both at most n, for n a very large number. They want to check to see if their numbers are the same, but Alice does not want to have to send her entire number to Bob.[1]
What is an efficient randomized algorithm for Alice and Bob that has "small" probability of failure? How many bits does Alice need to send to Bob as a function of n, and what is the probability of failure, where failure means that this algorithm says x and y are equal, but in fact they are different?

[1]For example, x and y represent large binary files (think terabytes), and Alice and Bob want to check that their files are equal, without Alice having to send her entire file to Bob