CS 561, HW2

Prof. Jared Saia, University of New Mexico

Due: Sept. 13

1. In this problem you will use Chernoff bounds to show that for most of the levels of a skip list, the size of the level is very tightly bounded around its expectation.

   Chernoff Bounds: Assume you have n independent, indicator random variables $X_1, X_2, ..., X_n$ and let $X = \sum_{i=1}^{n} X_i$ and $\mu = E(X)$. Then Chernoff bounds tell us that for any $0 \leq \delta \leq 1$:

   $Pr(X \leq (1-\delta)\mu \text{ or } X \geq (1+\delta)\mu) \leq 2e^{-\mu\delta^2/4}$

   Use Chernoff and Union bounds to show that with probability at least $1-1/n$, for all $0 \leq j \leq \log n - \log\log n - 5$, List j in a skip list contains between $n/2^{j+1}$ and $3n/2^{j+1}$ nodes. Let List 0 be the bottom list, List 1 be the next higher up, etc. You may assume that n is sufficiently large, e.g. n is larger than some constant n_0.

   Note: Chernoff bounds are more powerful than bounds on Binomial distributions since X need not be binomially distributed. The only requirement is that the X_i be independent. Hint: Remember that $e^{-x} \leq 2^{-x}$.

2. You toss m pebbles onto the nodes of a k-regular undirected graph (recall that a graph is k-regular if every node has degree k). Each pebble lands on a node selected uniformly at random. A pair of pebbles is said to "collide" if they fall on the same node or on two nodes that are neighbors. What is the expected number of pairs of pebbles that collide? About how large must m be before you would expect at least 1 pair of pebbles to collide?

3. Consider the recurrence $T(n) = 3T(n/4) + \log^2 n$

   (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 please note that your two bounds should match)

4. Consider the following function:

```
int f (int n){
  if (n==0) return 3;
  else if (n==1) return 5;
  else{
    int val = 3*f (n-1);
    val = val - 2*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. 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 integer x, x factors into at most $\log x$ 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 numbers less than n 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 databases x and y where x and y have value no more than n, for n a very large number (think terabytes). They want to check to see if their databases are consistent (i.e. they want to check if they are the same) but Alice does not want to have to send her entire database to Bob. What is an algorithm Alice and Bob can use to check consistency with reasonably good probability by sending a lot fewer bits? 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 the databases are the same but in fact they are different?