CS361 Homework #5
Due Thursday, December 7th

1. **Cracking the RSA code.** Suppose my public key for the RSA code is $n = 10573$ with an encryption key $e = 5003$. You, the eavesdropper, intercept a message consisting of three four-digit packets:

   $$2222,\ 4370,\ 78$$

   Your job to is the decrypt this message, using the following method.

   (a) First find the primes p and q such that $pq = n$. (Hint: search for primes which are roughly $\sqrt{n}$).)

   (b) Now find $\phi = (p-1)(q-1)$, which is the number of integers which are mutually prime to n.

   (c) Now, using the extended Euclid's Algorithm, find x and y such that $x\phi + ye = 1$, and thus the inverse $y = e^{-1} \bmod \phi$. This is the decryption exponent d. (Show your work; if y is negative, you'll have to find whatever positive number is equivalent to it mod ϕ. Check your work by making sure that $ed \bmod \phi = 1$.)

   (d) Finally, raise each of the intercepted packets to the dth power mod n and write down the result. Feel free to use any method you like to do this; don't worry about the repeated-squaring thing.

   Answer: Heres how Euclids algorithm computes the gcd of $\phi = 10368$ and $e = 5003$, while writing the resulting numbers in the form $x\phi + ye$ at each stage:

10368	5003	
5003	362	$= 10368 - 2 \times 5003$
362	297	$= 5003 - 13 \times 362 = -8 \times 10368 + 17 \times 5003$
297	65	$= 362 - 297 = 14 \times 10368 - 29 \times 5003$
65	37	$= 297 - 4 \times 65 = -69 \times 10368 + 143 \times 5003$
37	28	$= 65 - 37 = 83 \times 10368 - 172 \times 5003$
28	9	$= 37 - 28 = -152 \times 10368 + 315 \times 5003$
9	1	$= 28 - 3 \times 9 = 539 \times 10368 - 1117 \times 5003$

   This gives $y = -1117$, which mod $\phi = 10368$ is equivalent to $d = 9251$, and indeed $edmod\phi = 1$.

2. For each of these, state whether $f = o(g)$, $\Theta(g)$, or $\omega(g)$. Justify your answer.

   (a) $f(n) = 2^{\sqrt{n}}$, $g(n) = 3^{\sqrt[3]{n}}$
   What matters most here is not 2 vs. 3, but $\sqrt{n} = n^{1/2}$ vs. $\sqrt[3]{n} = n^{1/3}$. If we take logs, we get

   $$\log_2 f(n) = n^{1/2}, \quad \log_2 g(n) = (\log_2 3)n^{1/3} = \Theta(n^{1/3})$$

   Since $\log f = \omega(\log g)$, we also have $f = \omega(g)$.

   (b) $f(n) = n^4 3^n$, $g(n) = n^{100}$
   3^n is an exponential, and trumps any polynomial, even n^{100}. So $f = \omega(g)$.

   (c) $f(n) = \log(n^2)$, $g(n) = (\log n)^2$
   $f(n) = \log(n^2) = 2\log n = \Theta(\log n) = o((\log n)^2)$. So $f = o(g)$.

3. Consider the following piece of code:

```
thing(int n) {
  if (n==0) return;
  thing(n-1);
  thing(n-2);
  for (int i=0; i < log(n); i++) print i;
  thing(n-1);
}
```

Let $f(n)$ be the running time of `thing` on input n; to be specific, let $f(n)$ be the number of times `thing` prints a number. Write a recurrence for $f(n)$, and solve it within Θ.

Answer: As always, the running time has a recursive part and a driving term:

$$f(n) = 2f(n-1) + f(n-2) + \log n \ .$$

First we solve this without the driving term. If the recurrence were just

$$f(n) = 2f(n-1) + f(n-2)$$

we would guess an exponential form, $f(n) = a^n$. Plugging this in and simplifying gives a quadratic equation for a,

$$a^2 - 2a - 1 = 0 \ .$$

There are two solutions

$$a = 1 \pm \sqrt{2}$$

and the positive one is the one we care about. This gives

$$f = \Theta((1+\sqrt{2})^n) \ .$$

The driving term $\log n$ is insignificant compared to this exponential growth, so this solution is still correct (within Θ) for the overall problem.

4. Solve each of these recurrences within Θ and justify your answers.

   (a) $f(n) = 8f(n/2) + n$
   Answer: Solving without the driving term gives $f(n) = \Theta(n^3)$ (like the volume of a cube, which is multiplied by 8 whenever the length of the side is doubled), and the driving term is too small compared to this to matter.

   (b) $f(n) = 2f(n-2) + (3/2)^n$
   Answer: Solving without the driving term gives $f(n) = \Theta(2^{n/2}) = \Theta(\sqrt{2}^n)$. But, since $\sqrt{2} < 3/2$, the driving term dominates this, and $f(n) = \Theta((3/2)^n)$.

   (c) $f(n) = 2f(n/2) + \sqrt{n}$
   Answer: Solving without the driving term gives $f(n) = \Theta(n)$, and the driving term is too small to matter.

   (d) $f(n) = 5f(n/5) + n$
   Answer: Just like Mergesort: solving without the driving term gives $f(n) = \Theta(n)$, and since the driving term is also $\Theta(n)$, we get $f(n) = \Theta(n \log n)$.

   (e) $f(n) = f(n/2) + 4$
   Answer: If this 4 were a 1, this would be exactly the definition of $\log_2 n$, which increases by 1 each time n doubles. In fact, $f(n) = 4 \log_2 n$ is an exact solution, so $f(n) = \Theta(\log n)$.

   (f) $f(n) = f(n-1) + f(n-2) + f(n-3) + \cdots + f(1) + f(0)$
   Answer: The best way to solve this is to try it out and see what happens. But, you should guess an exponential form, $f(n) = a^n$. Plugging this in and dividing both sides by a^{n-1} gives

   $$a = 1 + \frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^3} + \cdots$$

   and treating the right-hand side as a geometric series, we get

   $$a = \frac{1}{1 - 1/a} = \frac{a}{a-1}$$

   and solving this gives $a = 2$, so $f(n) = \Theta(2^n)$. In fact, if we use the base case $f(0)$, then the first few values of $f(n)$ are exactly

   $$1, 1, 2, 4, 8, 16, \ldots,$$

   doubling at each step.

5. Solve this recurrence exactly; start with a reasonable guess, and then substitute it back into the recurrence and base case to adjust the constants and make sure it works. (Assume n is a power of 2.)

   $$f(n) = 4f(n/2) + 3n^2, \ f(1) = 0$$

   Answer: First let's solve it within Θ. Without the driving term, f quadruples when n doubles, so $f(n) = \Theta(n^2)$. But the driving term is also $\Theta(n^2)$, so $f = \Theta(n^2 \log n)$. To get an exact solution, we need to discover the constant hidden in Θ. So, we write

   $$f(n) = An^2 \log_2 n$$

   and substitute this into the recurrence. This gives

   $$\begin{aligned} An^2 \log_2 n &= 4A(n/2)^2 \log_2(n/2) + 3n^2 \\ &= An^2((\log_2 n) - 1) + 3n^2 \\ &= An^2 \log_2 n - An^2 + 3n^2 \end{aligned}$$

   Canceling, we get $A = 3$. Thus our exact solution is

   $$f(n) = 3n^2 \log_2 n \ .$$

6. Suppose I have n people in a room. Assuming that all birthdays are equally likely, what is the average number of triplets of people A, B, C such that their birthdays fall on three consecutive days? How large does n have to be before the probability that there is such a triplet becomes fairly large?

   Answer: Just as in the birthday problem, the average number of triplets for which this happens is the number of possible triplets, times the probability that a given triplet falls on consecutive birthdays. The number of triplets is

   $$\binom{n}{3} = \frac{n(n-1)(n-2)}{3!}$$

   and the probability that the second person's birthday is after the first persons, and the third's is after the second's, is

   $$\frac{1}{365^2} \ .$$

   But, there are 6 ways for this to happen, since three people could have their birthdays in $3! = 6$ different orders. So the probability is 6 times this, and this cancels the 6 in the denominator of $\binom{n}{3}$. Thus the total average number of triplets is

   $$\frac{n(n-1)(n-2)}{365^2}$$

   and this is roughly 1 when $n = 53$.

   Another way to think about this factor of 6: when we define $\binom{n}{3}$, we divide by $3! = 6$ because when choosing a set of three people, we don't care in what order they were chosen. But in this case we *do* care about the order, so we shouldn't divide by 6 in the first place.

   To put it yet another way, having three people fall on consecutive birthdays is 6 times as likely as having them all fall on the same birthday (as in the midterm question), since there are 6 times as many ways for it to happen.

7. If I call `downSift` on a heap containing n items, the worst-case running time is $O(\log n)$. But for most nodes, the running time is only $O(1)$. Explain why, and give a clearer meaning to what "most" and $O(1)$ mean here.

   Answer: The shape of a heap is a balanced binary tree, except that the last row may be incomplete. For simplicity, let's assume the last row is complete. Then 1/2 of the nodes are leaves, so `downSift` has nothing to do; 1/4 are one level above the leaves, so `downSift` causes at most one swap; 1/8 are two levels above the leaves, so we need at most two swaps; and so on. Adding these up, we see that only a 2^{-k} fraction of the nodes could need k swaps, and only the root could need all $\log_2 n$ swaps.

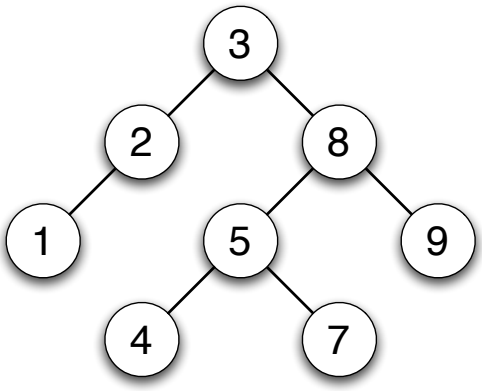

Figure 1: An AVL tree. What happens when we insert 6?

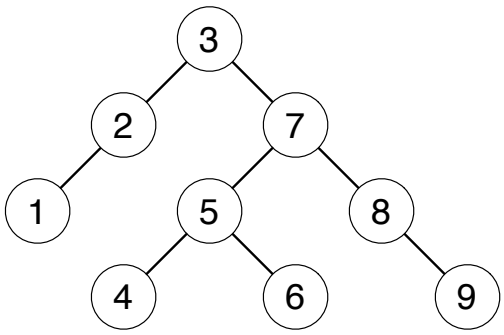

Figure 2: The final state after inserting 6.

8. Figure 1 shows an AVL tree. What are the balance factors? If I insert a new item with key 6, what will the balance factors be, and which nodes will be "too unbalanced"? What rotations will it do to repair itself, and what will be its final state?

   Answer: Initially, the balance factor on 3 is $+1$, the ones on 2 and 8 are -1, and all others are 0. After adding 6, the balance factors of 7, 5, 8 and 3 are -1, $+1$, -2 and $+2$ respectively. The tree responds by rotating 5 to the left and down, and then rotating 8 to the right and down, giving the final state shown in Fig. 2.

9. Figure 2 shows a 2-3-4 tree. If I insert a new item with key 11, what will happen, and what will be the final state of the tree? Assume that full nodes get split on the way down the search path.

   Answer: On the way down we split the node $(6, 8, 13)$, pushing 8 up to join $(4, 15)$, and creating two nodes (6) and (13) with two children each. We then split $(9, 10, 12)$, pushing 10 up to join (13) and creating two nodes (9) and (12). Finally, the new item 11 joins (12). The final state is shown in Fig. 4.

10. Prove that for any t it is possible to calculate $x^t \bmod p$ with less than $2 \log_2 t$ multiplications. For which values of t is this the hardest to do?

    Answer: Suppose t has $k+1$ binary digits. By starting with x and squaring k times, we obtain

    $$x, x^2, x^4, x^8, \ldots, x^{2^k} \quad .$$

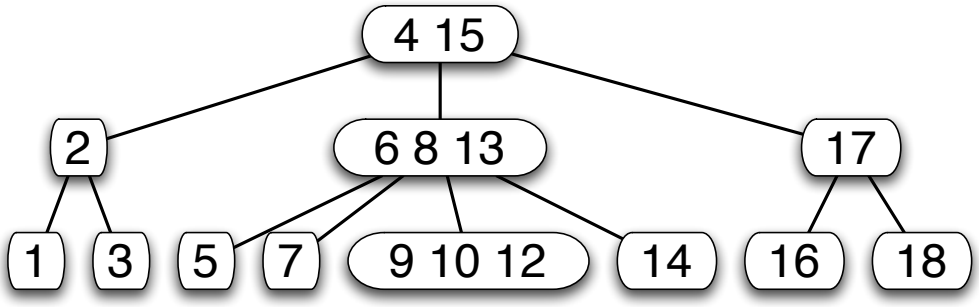

Figure 3: A 2-3-4 tree. What happens when we insert 11?

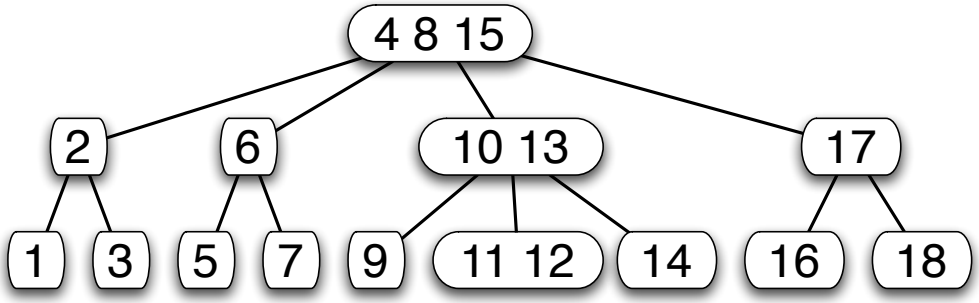

Figure 4: The 2-3-4 tree after we insert 11.

Now, since t can be written as the sum of powers of 2 up to 2^k (this is its binary digit expansion), we just multiply the corresponding powers of x together and get x^t. The total number of multiplications is k (for the squaring) plus k (for combining powers of 2 to get t) for a total of at most $2k$. Finally, $k \leq \log_2 t$.

The worst case for this approach is when t's binary expansion is all 1s, which happens when t is of the form $2^{k+1} - 1 = 1 + 2 + 4 + 8 + \cdots + 2^k$. Then we need to combine all these powers of x together, and we can't skip any of them. On the other hand, if t has just j 1s in its binary expansion, we only need $t + j$ multiplications.