CS:3330 Homework 9, Spring 2017

Due at the start of class on Thursday, April 27th

---

1. (a) Let $T(n)$ denote the running time of StoogeSort on an input of size n. This running time is given by the following recurrence relation –

$$T(2) = 1$$
$$T(n) = 3T(2n/3) + 1$$
(the three recursive calls and the 1 for the divide and combine)

(b) Here, $a = 3, b = 3/2$, and $f(n) = 1$. We have $3 \cdot f(2n/3) = 3 \cdot f(n)$. Therefore, $\alpha = 3 > 1$. So, we are in case 3 of the Master theorem discussed in lecture, so–

$$T(n) = O(n^{\log_{3/2}(3)})$$
$$\approx O(n^{2.709})$$

2. (a) Let $U(n)$ denote the running time of Unusual on an input of size n. This running time is given by the following recurrence relation –

$$U(2) = 1$$
$$U(n) = 3U(n/2) + n \qquad \text{(the three recursive calls and the swapping)}$$

(b) Here, $a = 3, b = 2$, and $f(n) = n$. We have $3 \cdot f(n/2) = \frac{3}{2} \cdot f(n)$. Therefore, $\alpha = 3/2 > 1$. So, we are in case 3 of the Master theorem discussed in lecture, so–

$$U(n) = O(n^{\log_2(3)})$$
$$\approx O(n^{1.58})$$

3. (a) $T(n) = T(n-2) + 2^n$ for $n \geq 2$, $T(1) = 1$, $T(0) = 0$.
After the first unroll step, we get

$$T(n) = T(n-4) + 2^{n-2} + 2^n.$$

After the second unroll step, we get

$$T(n) = T(n-6) + 2^{n-4} + 2^{n-2} + 2^n.$$

After the third unroll step, we get

$$T(n) = T(n-8) + 2^{n-6} + 2^{n-4} + 2^{n-2} + 2^n.$$

We will now *guess* that for any integer $k \geq 1$,

$$T(n) = T(n-2k) + 2^{n-2(k-1)} + 2^{n-2(k-2)} + \cdots + 2^{n-2} + 2^n.$$

This guess can be *confirmed* using the following inductive proof.
Base Case: $k = 1$. For $k = 1$, the guess is simply the original recurrence $T(n) = T(n-2) + 2^n$, which is obviously correct.
Inductive hypothesis: Suppose that the guess is correct for some $k \geq 1$. So let us start with the guess

$$T(n) = T(n-2k) + 2^{n-2(k-1)} + 2^{n-2(k-2)} + \cdots + 2^{n-2} + 2^n$$

and unroll it once more using $T(n-2k) = T(n-2k-2) + 2^{n-2k}$. Plugging this for $T(n-2k)$ gives us

$$T(n) = T(n-2(k+1)) + 2^{n-2k} + 2^{n-2(k-1)} + 2^{n-2(k-2)} + \cdots + 2^{n-2} + 2^n.$$

This confirms the guess for $k+1$ and completes the inductive proof.

Now we use the guess to solve for $T(n)$. If n is even, we pick k such that $n - 2k = 0$, implying that $k = n/2$. Then,

$$T(n) = T(0) + 2^2 + 2^4 + \cdots + 2^{n-2} + 2^n.$$

We plug $T(0) = 0$ and verify using the geometric series formula that $T(n) = O(2^n)$.

If n is odd, we pick k such that $n - 2k = 1$, implying that $k = (n-1)/2$. Then,

$$T(n) = T(1) + 2^3 + 2^5 + \cdots + 2^{n-2} + 2^n.$$

We plug $T(0) = 1$ and verify using the geometric series formula that $T(n) = O(2^n)$.

(b) $T(n) = (T(n-2))^2$ for $n \geq 1$, $T(0) = 2$.

After the first unroll step, we get

$$T(n) = ((T(n-4))^2)^2 = T(n-4)^4.$$

After the second unroll step, we get

$$T(n) = ((T(n-6))^2)^4 = T(n-6)^8.$$

After the third unroll step, we get

$$T(n) = ((T(n-8))^2)^8 = T(n-8)^{16}.$$

We will now *guess* that for any integer $k \geq 1$,

$$T(n) = T(n-2k)^{2^k}.$$

This guess can be *confirmed* using an inductive proof – I am going to skip this proof for now. Like in part (a), since k can only take on integer values, we should really be separately considering odd and even values of n. But, we will "fudge" this and just pick k such that $n - 2k = 0$, implying that $k = n/2$. This gives us

$$T(n) = T(0)^{2^{n/2}} = 2^{2^{n/2}}.$$

(c) $T(n) = T(n/2) + n$ for $n \geq 2$, $T(1) = 1$.

After the first unroll step, we get

$$T(n) = T(n/4) + n/2 + n.$$

After the second unroll step, we get

$$T(n) = T(n/8) + n/4 + n/2 + n.$$

After the third unroll step, we get

$$T(n) = T(n/16) + n/8 + n/4 + n/2 + n.$$

We will now *guess* that for any integer $k \geq 1$,

$$T(n) = T\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}} + \frac{n}{2^{k-2}} + \cdots + \frac{n}{4} + \frac{n}{2} + n.$$

This guess can be *confirmed* using an inductive proof – I am going to skip this proof for now. We now set $n/2^k = 1$, implying that $k = \log_2 n$. (There is some "fudging" going on here also because for $k \geq 1$ to be an integer, n needs to be a power of 2.) This gives us

$$T(n) = T(1) + \frac{n}{2^{\log_2 n - 1}} + \frac{n}{2^{\log_2 n - 2}} + \cdots + \frac{n}{4} + \frac{n}{2} + n.$$

Plugging in $T(1) = 1$ and evaluating the geometric series gives us $T(n) = O(n)$.

4. (a) 16. **Explanation:** The first call to `strangeSum` has parameters $(L, 0, 1)$ and it returns the sum of the first two elements; so `leftSum` is 5. The second call to `strangeSum` has parameters $(L, 1, 2)$ and it returns the sum of the middle two elements; so `midSum` is 6. The third call to `strangeSum` has parameters $(L, 2, 3)$ and it returns the sum of the last two elements; so `rightSum` is 5.

   (b) $T(n) = 3T(n/2) + 1$ for $n > 2$ and $T(n) = 1$ for $n = 1, 2$.

   (c) This recurrence has a form for which the Master Theorem applies. So $a = 3$, $b = 2$, $f(n) = 1$, and $af(n/b) = 3f(n)$. Therefore, $T(n) = O(n^{\log_2 3})$.

5. (a)

```
# Given numbers a and n and a nonnegative integer d, this function
# computes a^d mod n, i.e., the remainder one gets when a^d is
# divided by n.

# The function uses recursion to speedup computation. The function
# performs about log(d) multiplications, rather than d-1
# multipications. So it should have no trouble handling d with 100s
# or even 1000s of digits. Also, in order to keep intermediate values
# small, performs the modulo operation as soon as possible, rather
# than wait for a^d to be computed.
def fastPowerMod(a, d, n):
    # Base cases of the recursion
    if(d == 0):
        return 1 % n
    elif(d == 1):
        return a % n
    # Recursive case
    else:
        temp = fastPowerMod(a, d/2, n)
        if(d%2 == 0): # if d is even
            return (temp*temp) % n
        else: # d is odd
            return (((temp*temp)%n)*a)%n
```

   (b) 1

   (c) The algorithm performs $O(\log n)$ multiplications and $O(\log n)$ mod operations. Each of these operations is performed on numbers that are at most n in value and therefore at most $O(\log n)$ bits in size. Each multiplication and each mod takes time that is quadratic in the size of the given numbers. Therefore, each of these arithmetic operations takes $O(\log^2 n)$ time and altogether the entire running time of the algorithm is $O(\log^3 n)$.

6. $T(n) = T(n-1) + n$ for $n \geq 2$ and $T(1) = 1$. We can use the unroll, guess, and confirm method to show that $T(n) = O(n^2)$.

7. (a) $T(n) \leq T(|L_1|) + T(|L_2|) + n$, where $n/3 \leq |L_1|, |L_2| \leq 2n/3$ and $|L_1| + |L_2| = n$.
   (b) Using the recursion tree method, we can solve this to get $T(n) = O(n \log n)$ as the expected running time.
