CS141, Fall 2020, Assignment #1 **Due: 23:59, Wed 10/06, 2021**
Name: Rishab Dudhia **Student ID: 862141444**

1 A Complex Complexity Problem (25pts)

g_1: 3^n	g_5: $\log^2 n$	g_9: 1	g_{13}: n
g_2: $n!$	g_6: $\log(n!)$	g_{10}: $\ln n$	g_{14}: 2^n
g_3: $n^2 + n$	g_7: 2^{2^n}	g_{11}: $\log \log n$	g_{15}: $n \log n$
g_4: n^3	g_8: $\ln \ln n$	g_{12}: $\sqrt{\log n}$	g_{16}: $5n^2 - 13n + 6$

(1) To be polylogarithmic, a function must be $O(log^k n)$. Any function that is asymptotically less than $O(log^k n)$ is also polylogarithmic since Big-Oh is an upperbound.
Polylogarithmic Funcitons:
g_5, g_8, g_9, g_{10}, g_{11}, g_{12}, g_{13}

(2) To be sublinear, a function must be $o(n)$. For a function to be $o(n)$, the function $f(n)$ must be strictly less than $g(n)$, so $f(n)$ cannot be n for a sublinear function since n is equal to n while o is strictly less than.
Sublinear Functions:
g_5, g_8, g_9, g_{10}, g_{11}, g_{12}

(3) To be superlinear, a function must be $\omega(n)$. For a function to be $\omega(n)$, the function $f(n)$ must be strictly larger than $g(n)$, so $f(n)$ cannot be n for a superlinear function since n is equal to n while ω is strictly greater than.
Superlinear functions:
g_1, g_2, g_3, g_4, g_6, g_7, g_{14}, g_{15}, g_{16}

(4) (a) Compare $f(n) = g_{11}$ and $g(n) = g_{12}$. Which one grows faster?

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{log(log(n))}{\sqrt{log(n)}}$$
$$= 0$$

The function $\sqrt{log(n)}$ can be written as $log(n^{1/2})$ and we know we can drop constants when calculating asymptotic complexities. Therefore, $\sqrt{log(n)}$ is $O(log(n))$ while $log(log(n))$ is $O(log(log(n)))$. Since $log(log(n))$ is less than $log(n)$ when n is sufficiently large, $\sqrt{log(n)}$ will grow faster.

(b) Show that g_8 and g_{11} are asymptotically equivalent.

Let $f(n) = ln(ln(n))$ and let $g(n) = log(log(n))$. For $f(n) = \Theta(g(n))$, the following two conditions must be satisfied:

$$f(n) = O(g(n))$$
$$f(n) = \Omega(g(n))$$

To show this, we must choose two constant values c_1 and c_2 to multiply $g(n)$. We will choose $c_1 = 1$ and $c_2 = 10$ as our constants.
We can then say $f(n) = O(g(n))$ because

$$0 \leq ln(ln(n)) \leq c_2 \cdot log(log(n))$$

$$n \geq n_0$$

where $n_0 = 30$. We have just shown that $g(n)$ is an asymptotic upper bound for $f(n)$. Next, we will show that $g(n)$ is also an asymptotic lower bound of $f(n)$. This is shown through the following,

$$0 \leq c_1 \cdot log(log(n)) \leq ln(ln(n))$$

$$n \geq n_0$$

with n_0 still being 30. Now we have shown $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, so we can say that $log(log(n))$ and $ln(ln(n))$ are asymptotically equivalent.

(c) Show that g_6 and g_{15} are asymptotically equivalent.

Let $f(n) = log(n!)$ and $g(n) = nlog(n)$. We know

$$log(n!) = log(n) + log(n-1) + ... + log(2) + log(1) \quad (1)$$
$$nlog(n) = log(n) + log(n) + ... + log(n) \quad (2)$$

On the right hand side of (1), there are n terms with a logarithm in each. The first logarithm is the logarithm of n and each concurrent term takes the logarithm of one less than the term before. Then, on the right hand side of (2), there are n terms of $log(n)$ being added together. Therefore, since $nlogn$ is the sum of n logarithms of n, $log(n!)$ is less than $nlogn$ for sufficiently large n because it is only the sum of n logarithms of values starting at n all the way down to 1. We have just proven that $f(n) = O(g(n))$ as $g(n)$ is an upper bound for $f(n)$. To show that $g(n)$ is also a lower bound, we will modify (1),

$$\begin{aligned} log(n!) &= log(n) + log(n-1) + ... + log(2) + log(1) && (3)\\ &\geq log(n) + log(n-1) + ... + log(n/2) && (4)\\ &\geq log(n \cdot (n-1) \cdot ... \cdot \frac{n}{2}) && (5)\\ &\geq log((\frac{n}{2})^{\frac{n}{2}}) && (6)\\ &\geq \frac{n}{2} \cdot log(\frac{n}{2}) && (7)\\ &\geq n \cdot log(n) && (8) \end{aligned}$$

From line (7) to (8) we just dropped the constants to have a final form of $nlog(n)$. We now have that $log(n!)$ is asymptotically greater than $nlog(n)$, so we have shown $f(n) = \Omega(g(n))$. Since we have now shown that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, we can conclude that $f(n) = \Theta(g(n))$.

(5)

$$g_9 < g_8 = g_{11} < g_{12} < g_{10} = g_5 < g_{13} < g_6 = g_{15} < g_3 = g_4 = g_{16} < g_1 = g_{14} < g_2 < g_7$$

2 Solve Recurrences (15pts)

(1) $T(n) = T(n-1) + n^2$

The master theorem cannot be used for this problem since the recurrence is not in the form required for master theorem. Thus, to solve the recurrence we will expand it with each call to $T(n)$ enclosed in a pair of brackets, [ ]. The final line will occur when the base case of the recursion is reached and all

calls to $T(n)$ have been made. We will start with the recurrence given

$$\begin{aligned} T(n) &= [T(n-1) + n^2] \\ &= [[T(n-2) + (n-1)^2] + n^2] \\ &= [[[T(n-3) + (n-2)^2] + (n-1)^2] + n^2] \\ &... \\ &= T(0) + 1^2 + 2^2 + ... + (n-1)^2 + n^2 \end{aligned}$$

From the problem statement we know that the base case will also be a constant so the first term will take $\Theta(1)$ time and it can be ignored. Discarding the first term, we have

$$1^2 + 2^2 + ... + (n-1)^2 + n^2$$

We can see that this is a sum of squares from 1 to n. The formula for a sum of squares from 1 to n is

$$\frac{n(n+1)(2n+1)}{6} = \frac{2n^3 + 3n^2 + n}{6}$$

We can see that this equation is of degree 3 so we think that the recurrence is $\Theta(n^3)$. To prove our educated guess, we must prove that our expanded function, let's call it $f(n) = \frac{2n^3+3n^2+n}{6}$, has an asymptotic complexity of $O(n^3)$ and $\Omega(n^3)$. First, we will show that $f(n)$ is of complexity $O(n^3)$. This is simply shown by the following:

$$1^2 + 2^2 + ... + (n-1)^2 + n^2 < n^2 + n^2 + ... + n^2 + n^2$$

Let's say that there are n terms on the Right Hand Side (RHS) just as there are n terms on the Left Hand Side (LHS). Obviously the RHS will be greater as the first term is larger than the first term of the LHS. In fact, every term on the RHS is larger than its partner term on the LHS except for the final term for both sides which are equal. RHS has n n^2 terms, or n^3 terms, so asymptotically it can said to be $O(n^3)$. Since we just showed that our function, which was on the LHS, is less than the RHS, which is of complexity $O(n^3)$, we can say that our function is also of $O(n^3)$ complexity.
We will use the definition of Big Omega to prove that $f(n)$ is also of complexity $\Omega(n^3)$. The definition states that a function $f(n)$ should be greater than or equal to a function $c \cdot g(n) \geq 0$ for $c > 0$ and $n \geq n_0 > 0$. We will choose $c = 0.1$ and $g(n) = n^3$. Then

$$\begin{aligned} \frac{2n^3 + 3n^2 + n}{6} &\geq 0.1 \cdot n^3 \\ 2n^3 + 3n^2 + n &\geq 0.6n^3 \end{aligned}$$

Clearly the above inequality is true for all $n > 0$ since the n^3 term is larger on the Left Hand Side (LHS), and the LHS has additional positive terms while the RHS only has the one n^3 term. Thus, $f(n) > c \cdot g(n)$ for $n > 0$, and $f(n) = \Omega(n^3)$.
Since $f(n)$ is of complexities $O(n^3)$ and $\Omega(n^3)$ and $f(n)$ is just the expanded representation of $T(n)$, we can confidently say that the recurrence is

$$T(n) = \Theta(n^3)$$

(2) $T(n) = 3T(n/2) + n^3$
Since the recurrence is in the correct form for master theorem, master theorem can possibly be used to solve the equation. To confirm that master theorem can be used, we must look at $f(n) = n^3$. For master theorem, we list the following values to confirm that master theorem can be used:

$$a = 3$$

$$b = 2$$
$$\log_b a = \log_2 3 \approx 1.584963$$
$$\epsilon = 0.5$$

From these values, we can deduce

$$\begin{aligned} f(n) &= \Omega(n^{\log_b a + \epsilon}) \\ &= \Omega(n^{\log_2 3 + 0.5}) \\ &= \Omega(n^{1.584963 + 0.5}) \\ n^3 &= \Omega(n^{2.085}) \end{aligned}$$

We have just proved the first requirement of case 3 of the master theorem which requires that $f(n)$ is polynomially larger than $n^{\log_b a}$. The second requirement, the regularity condition, requires

$$a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$$

for $c < 1$ and sufficiently large n. If we choose $c = 0.9$, the inequality becomes

$$\begin{aligned} 3(\frac{n}{2})^3 &\leq 0.9n^3 \\ \frac{3n^3}{8} &\leq \frac{9n^3}{10} \\ 15n^3 &\leq 36n^3 \\ 0 &\leq 21n^3 \\ n &\geq 0 \end{aligned}$$

This shows that the second requirement for case 3 of the master theorem holds since $c \cdot f(n)$ is larger than $a \cdot f(\frac{n}{b})$ when $n \geq 1$. Since all requirements of case 3 are true, we can conclude that

$$T(n) = \Theta(n^3)$$

(3) $T(n) = 2T(n/5) + \log\log n$
The recurrence is in the form accepted by master theorem so we will try case 1 of the master theorem to see if the recurrence applies. Case 1 states that $f(n)$ must be of complexity $O(n^{(\log_b a) - \epsilon})$ where $\epsilon > 0$. To prove that the theorem applies, we will take the following values

$$a = 2$$
$$b = 5$$
$$\log_b a = \log_5 2 \approx 0.43068$$
$$\epsilon = 0.1$$
$$(\log_b a) - \epsilon = 0.33068$$
$$f(n) = \log\log n$$

We have just shown that the recurrence has a leaf cost of $\Theta(n^{0.33068})$. In the recurrence, the root cost is $\log\log n$. From the long inequality comparing many of the common functions from class, we know that $\log\log n$ is asymptotically less than $n^{0.33068}$. Therefore, $f(n) = O(n^{0.33068})$, so the requirement of case 1 of the master theorem holds true. We can conclude that

$$T(n) = \Theta(n^{log_5 2}) = \Theta(n^{0.33068})$$

(4) $T(n) = 2T(n/2) + n \log n$

First, we will attempt to solve the recurrence using the expansion tree. The height of the expansion tree must be $\log_b n = \log_2 n$. We also know that the nodes in each level will add up to $n \log_2 n$. Therefore, we predict that the complexity of the recurrence is

$$\log_2 n \cdot n \log_2 n = n \log_2^2 n = \Theta(n \log_2^2 n)$$

We can see that the recurrence is in the accepted form for master theorem. To attempt to solve the recurrence, we will look at case 2 of the master theorem. Case 2 states that $f(n)$ must have an asymptotic complexity of $\Theta(n^{\log_b a} \log^k n)$ where $k \geq 0$. We set the following values

$$\begin{aligned} a &= 2 \\ b &= 2 \\ k &= 1 \\ n^{\log_b a} \log^k n &= n^{\log_2 2} \log^1 n \\ &= n \log n \end{aligned}$$

Now we have to prove that $f(n) = \Theta(n \log n)$ by showing that $f(n) = O(n \log n)$ and also that $f(n) = \Omega(n \log n)$. First, we will show that $f(n) = O(n \log n)$ by choosing a constant value c that can be multiplied to $n \log n$ to have an upper bound on $f(n)$ on a range $n \geq n_0$. Let us choose $c = 2$, then we have

$$\begin{aligned} f(n) &\leq c \cdot g(n) \\ n \log n &\leq 2 \cdot n \log n \end{aligned}$$

Clearly, this inequality will always hold true for, so we can say that $c \cdot g(n)$ is an upper bound on $f(n)$ for $n > 1$. Thus, we have shown that $f(n) = O(n \log n)$.

Now we will show that $f(n) = \Omega(n \log n)$. Once again we will choose c, but this time to be a lower bound on $f(n)$ on a range $n > n_0$. Let us choose $c = 0.5$, then we have

$$\begin{aligned} f(n) &\geq c \cdot g(n) \\ n \log n &\geq 0.5 \cdot n \log n \end{aligned}$$

Now it is obvious that the inequality is always true for $n > 1$ since the Right Hand Side will always be half of the Left Hand Side. Now we have shown that $f(n)$ also has an asymptotic complexity of $\Omega(n \log n)$.

Now, since we have shown that $f(n)$ is of complexities $O(n \log n)$ and $\Omega(n \log n)$, we can say that $f(n) = \Theta(n \log_2 n)$. Finally, we have satisfied the condition for case 2 of the master theorem, and we can conclude that

$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n) = \Theta(n \log^2 n)$$

3 Test the Candies (10pts)

(1) Each call to the following algorithm divides the input array or subarray in half and then checks each pile to see if a BAD, or $false$, is returned on the pile. If a BAD is returned, the function is now called on the subarray, or pile, that was just checked. As the program recurses, and there is at least one bad candy, eventually there will be a call to the function when i and k are equal, and this is the bad candy. Its array location (either i or k as they are equal and point to the same element) will be pushed onto to the vector of bad candies and the program will recursively add all of the bad locations onto the bad vector. We assume *bool checkPile* (*int*[*n*] *candies*, *int i*, *int j*) is the function that is called to use the device that returns GOOD = true or BAD = false. We also assume that this function uses constant time which will be important for part 2 of the problem.

```
check(int[n] candies, int i, int k, vector<int>& badCandies) {
  //parameters: candies: an array containing all candy,
  //i: points to front of array/subarray,
  //k: points to last element of array/subarray,
  //badCandies: vector of locations of bad candies in candies array
  if (i == j) {
      badCandies.push_back(i);
  }
  else {
      int j = (i + k) / 2;

      bool checkGood = checkPile(candies, i, j);
      if (!checkGood) {
          check(candies,i,j,badCandies);
      }

      checkGood = checkPile(candies, j+1, k);
      if (!checkGood) {
          check(candies,j+1,k,badCandies);
      }
  }
}
```

(2) The problem statement says, "when the number of bad candies is small," so we assume *checkPile*() returns GOOD at least once per call to *check*() when the difference between i and k is large. Therefore we will say that our function has the recurrence equation

$$T(n) = T(\frac{n}{2}) + c$$

Looking at the recurrence, we can see that it is in the acceptable form for master theorem. Master theorem requires a final term after the the recursive call, but in our case that term is just a constant c for all arithmetic and other constant time operations. In other words, of the recurrence, $f(n) = c$. Since n does not change c we know that $f(n)$ is of time complexity $\Theta(1)$. For master theorem, lets mark down some values:

$$a = 1 \tag{9}$$
$$b = 2 \tag{10}$$
$$\log_b a = 0 \tag{11}$$
$$n^{\log_b a} = 1 \tag{12}$$
$$\log_b^0 n = 1 \tag{13}$$

According to master theorem case 2, if $f(n) = \Theta(n^{\log_b a} \log^k n)$, then the recurrence is of complexity $\Theta(n^{\log_b a} \log^{(k+1)} n)$. Now we have already shown that $f(n) = \Theta(1)$, and we already have the values calculated to confirm case 2 of master theorem in lines (12) and (13) if we choose $k = 0$. The last step is simply to multiply $n^{\log_b a}$ and $\log_b^0 n$. Above we found that both of these values are 1 so the product will be equal to 1. Now we have shown

$$f(n) = \Theta(n^{\log_b a} \log^k n) = \Theta(1)$$

Now we have proven that $f(n)$ is of complexity $\Theta(1)$. Now we have confirmed that the master theorem applies to this recurrence and we can deduce

$$T(n) = \Theta(n^{\log_b a} \log^{(k+1)} n) = \Theta(\log n)$$

(3) If every candy was bad, our recurrence equation from part 2 would not hold because our assumption that most candy is good would not be true. Therefore, since *check*() needs to be called in every case and there are two calls to the test functions per call to *check*, our new recurrence will be

$$T(n) = 2T(\frac{n}{2}) + 2n$$

To solve this recurrence, we will try to apply case 2 of the master theorem. To apply case 2, we will have to show that $f(n) = \Theta(n^{\log_b a} \log^k n)$. We know from the form of master theorem that $f(n) = 2n$. First, we will have to declare some values:

$$\begin{aligned} a &= 2 \\ b &= 2 \\ \log_b a &= 1 \\ n^{\log_b a} &= n \end{aligned}$$

With these values, if we choose $k = 0$ we need to prove

$$f(n) = \Theta(n^{\log_b a} \log^k n) = \Theta(n)$$

To prove that $f(n) = \Theta(n)$ we will first show that $f(n) = O(n)$. To do this we will choose a constant $c = 10$ and function $g(n) = n$, and then we can see

$$\begin{aligned} f(n) &\leq c \cdot g(n) \\ 2n &\leq 10 \cdot n \end{aligned}$$

This inequality will always hold when $n > 0$, so on a range $n \geq 1$, $f(n)$ is of complexity $O(n)$. Next, we will show that $f(n) = \Omega(n)$. To do this, we will choose $c = 1$ for the following inequality:

$$\begin{aligned} f(n) &\geq c \cdot g(n) \\ 2n &\geq 1 \cdot n \end{aligned}$$

This inequality will always hold when $n > 0$, so $f(n)$ is strictly greater than $c \cdot g(n)$ on the range $n \geq 1$ and $f(n) = \Omega(n)$. Now we have shown that $f(n)$ is indeed $\Theta(n)$, so we can apply case 2 of the master theorem to the recurrence:

$$T(n) = \Theta(n^{\log_b a} \log^{(k+1)} n) = \Theta(n \log n)$$

(4) (bonus) Looking at the recursion tree of $T(n) = 2T(\frac{n}{2}) + 2n$, the tree will have height $\log_b n = \log n$. We know that there are two calls to the test function in each call to *check*(). Also, we know that there will be m final tests on the bad candies. Therefore, there will be $O(m \cdot 2logn)$ tests ran.