University of Nevada, Las Vegas Computer Science 477/677 Fall 2022

Answers to Assignment 2: Due Friday September 16, 2022, Midnight

1. Solve each recurrence, writing O, Ω, or Θ, whichever is most appropriate.

   (a) $F(n) = 4F\left(\frac{n}{2}\right) + 5n^2$

   Master theorem: $F(n) = \Theta(n^2 \log n)$

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

   Antiderivative: $f(n) = \Theta(n^2)$

   (c) $f(n) = f\left(\frac{n}{2}\right) + f\left(\frac{n}{3}\right) + n$

   Master theorem: $f(n) = \Theta(n)$

   (d) $f(n) = f(\sqrt{n}) + 1$.

   Let $m = \log n$, and let $f(n) = g(m) = g(\log n)$. Then $f(\sqrt{n}) = g(\log \sqrt{n}) = g(\frac{1}{2} \log n) = g(m/2)$.

   substituting:

   $$g(m) = g(m/2) + 1$$

   by master theorem:

   $$g(m) = \Theta(\log m)$$

   $$f(n) = g(m) = \Theta(\log \log n)$$

   (e) $f(n) = 2f(\sqrt{n}) + \log n$

   Let $m = \log n$, and let $f(n) = g(m) = g(\log n)$. Then $f(\sqrt{n}) = g(\log \sqrt{n}) = g(\frac{1}{2} \log n) = g(m/2)$.

   substituting:

   $$g(m) = 2g(m/2) + m$$

   by master theorem:

   $$g(m) = \Theta(m \log m)$$

   $$f(n) = g(m) = \Theta(\log n \log \log n)$$

   (f) $H(n) \leq 2H\left(\frac{n}{2}\right) + n$

   Master theorem: $H(n) = O(n \log n)$

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

   Let $m = 2^n$ and let $g(n) = h(m) = h(2^n)$. Then $g(n-1) = h(2^{n-1}) = h(m/2)$.

   substituting:

$$\begin{aligned} h(m) &= 2h(m/2) + 1 \\ \text{by master theorem:} & \\ h(m) &= \Theta(m) \\ g(n) = h(m) &= \Theta(2^n) \end{aligned}$$

(h) $G(n) \geq G(n-1) + \lg n$

Antiderivative: $G(n) = \Omega(n \log n)$

(i) $H(n) \leq 2H(\sqrt{n}) + 4$.

Let $m = \log n$, and let $H(n) = G(m) = G(\log n)$. Then $H(\sqrt{n}) = G(\log \sqrt{n}) = G(\frac{1}{2} \log n) = G(m/2)$.

$$\begin{aligned} \text{substituting:} & \\ G(m) &= 2G(m/2) + 4 \\ \text{by master theorem:} & \\ G(m) &= \Theta(m) \\ H(n) = G(m) &= \Theta(\log n) \end{aligned}$$

(j) $K(n) = K(n - 2\sqrt{n} + 1) + n$.

Antiderivative:

$$\begin{aligned} \frac{K(n) - K(n - (2\sqrt{n} - 1))}{2\sqrt{n} - 1} &= \frac{n}{2\sqrt{n} - 1} \\ K'(n) &= \Theta(\sqrt{n}) \\ K(n) &= \Theta\left(n^{3/2}\right) \end{aligned}$$

(k) $F(n) \leq F\left(\frac{n}{5}\right) + F\left(\frac{7n}{10}\right) + n$

Generalized master theorem: $F(n) = \Theta(n)$

(l) $F(n) = 2F\left(\frac{2n}{3}\right) + F\left(\frac{n}{3}\right) + n$

Generalized master theorem: $2(2/3)^2 + (1/3)^2 = 1$. Thus $F(n) = \Theta(n^2)$

(m) $f(n) = 1 + f(\log n)$

$f(n) = \Theta(\log^* n)$.

2. Find the asymptotic time complexity, in terms of n, for each C++ code fragment. Assume $n \geq 0$. These problems are similar to the ones above, except that you have to read the code, write the recurrence, then solve the recurrence.

(a)
```
void f(int i)
 {
  for(int j = 0; j < i; j++)
   cout << "hello world" << endl;
  if(i > 0) f(i/2);
  if(i > 0) f(i/2);
 }
int main()
 {
  f(n);
  return 1;
 }
```

The recurrence is $T(n) = n + 2T(n/2)$. The solution is $T(n) = \Theta(n \log n)$.

(b)
```
void f(int i)
 {
  if(i > 0)
   {
    for(int j = 0; j < i*i; j++);
     cout << "hello world" << endl;
    f(2*i/3);
    f(i/3);
    f(2*i/3);
   }
 }
int main()
 {
  f(n);
  return 1;
 }
```

The recurrence is $T(n) = n^2 + 2T(2n/3) + T(n/3)$. By the generalized master theorem, $T(n) = \Theta(n^2 \log n)$.

(c) You have nine coins, one of which is counterfeit. The eight good coins all weigh the same, but the counterfeit coin is slightly lighter. You have a balance scale. How can you find the counterfeit coin with at most two weighings?

1. The first step is to identify a set S of three coins, such that the bad coin is guaranteed to be a member of S. Place three coins on each side of the scale; the other three coins are not used. If the scale balances, we know that S consists of the coins not used. Otherwise, one side of the scale goes up, the other down. In that case S is the set of coins that go up.

2. We know the bad coin is in S. Place one of the coins in S on each side of the scale. Do not use the other coin in S. If the scale balances, the bad coin is the one you didn't use. Otherwise, one side of the scale goes up, and the bad coin is on that side.

3. The Coin-row problem: there is a row of n coins whose values are positive integers $C_0, C_1, C_2, \ldots, C_{n-1}$, not necessarily distinct. The goal is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

   Pick **one** of the three paradigms listed below, and write an algorithm for the coin-row problem using that paradigm.

   1. Greedy
   2. Dynamic Programming
   3. Recursion

   Use dynamic programming. Here is C++ code for the DP algorithm. Computation of each $X[i]$ is a subproblem, and $X[n-1]$ is the output.

   You didn't have to have written code for the algorithm; you could have simply explained it in words.

```
X[0] = C[0];
X[1] = C[1];
X[2] = C[0]+C[2];
for(int i = 3; i < n; i++)
 X[i] = max(X[i-3],X[i-2])+C[i];
cout << "The maximum value is " << max(X[n-2],X[n-1]);
```

   I didn't ask you to compute the maximum value set of coins, but I'll give you the algorithm for that anyway.

```
void writelistofcoins(int k);
 {
  if(k >= 2)
   {
    writelistofcoins(back[k]);
    cout << ",";
   }
  cout << k;
 }

int main()
 {
  X[0] = C[0];
  X[1] = C[1];
  X[2] = C[0]+C[2];
  back[2] = 0;
  for(int i = 3; i < n; i++)
   if (X[i-3] > X[i-2])
    {
     back[i] = i-3;
     X[i] = X[i-3]+C[i];
    }
```

```
  else
   {
    back[i] = i-2;
    X[i] = X[i-2]+C[i];
   }
 if(X[n-2] > X[n-1]) writelistofcoins(n-2);
 else writelistofcoins(n-1);
 return 1;
}
```

The loop invariant of the following code is:

`sum == A[0] + A[1] + ... + A[i]`

```
int sum = A[0];
int i = 0;
while(i < n)
 {
  i = i+1;
  sum = sum+A[i];
 }
cout << "The sum is " << sum << endl;
```

4. What is the loop invariant of the following code?

   The loop invariant is: `m = max(A[0], ... , A[i])`

   What is the output? The output is: `max(A[0], ... , A[n-1])`

```
int m = A[0];
int i = 0;
while(i < n)
 {
  i = i+1;
  if(A[i] > m) m = A[i];
 }
cout << m << endl;
```

5. What is the loop invariant of the following code?

   The loop invariant is: $y = x^i$

   What is the output? The output is x^k.

```
float x;
cin >> x;
int k;
cin >> k;
assert(k >= 0);
int i = 0;
float y = 1;
while(i < k)
  {
   i = i+1;
   y = y*x;
  }
cout << y << endl;
```