University of Nevada, Las Vegas Computer Science 477/677 Fall 2021

Assignment 3: Due Wednesday September 15, 2021 8:30 AM

Name:_____

You are permitted to work in groups, get help from others, read books, and use the internet. Do not turn this homework in. Please bring a printed copy to class on September 15, and we will go over the problems.

The exam on September 20 will be based on the problems in the first two homework assignments and on the problems in this assignment.

1. The following function computes the product of two positive integers. Verify that that `p + cd = ab` is a loop invariant for this code, and use the loop invariant to show that `product(a,b)` returns `a*b`.

```
1  int product(int a, int b)
2   {
3    assert (a > 0 and b > 0);
4    int c = a;
5    int d = b;
6    int p = 0;
7    while(d > 0)
8     {
9      if(d % 2) p = p+c;
10     c = c+c;
11     d = d/2;
12    }
13   return p;
14  }
```

There are a number of known techniques for verifying loop invariant, but I will use the technique I showed you in class. To make the discussion clearer, I have numbered the lines. There are two conditions for a loop invariant.

1. It holds before the first iteration of the loop, that is, before line 7.
2. If it holds before at the beginning of an iteration, that is, before line 9, it also holds at the end of that iteration, that is, after line 11.

We can then conclude, by the inductive principle, that the loop invariant hold after the last iteration.

We first prove the condition 1. Before line 7, $p + c * d = 0 + a * b = a * b$, hence the loop invariant holds.

For condition 2., Use c, d, and p for the values of those variables before one iteration, and c', d', and p' for the values of c, d, and p after the iteration. We are given that $p + c * d = a * b$, and we need to prove that $p' + c' * d' = a * b$. From the code, we know that $c' = c * c$ and $d' = d/2$, with truncation as in C++. We also know that $p' = p$ if d is even, and $p' = p + c$ if d is odd.

Case I: d is even. Then the division $d/2$ is not truncated, so $d' = \frac{1}{2}d$, hence $p' + c' * d' = p + (c + c) * \frac{1}{2}d = p + (2c) * \frac{1}{2}d = p + c * d$, and we are done.

Case II: d is odd. Then $d' = \frac{1}{2}(d-1)$. We have $p'+c'*d' = p+c+(c+c)*\frac{1}{2}(d-1)/2 = p+c+(2c)*\frac{1}{2}(d-1) = p+c+c*(d-1) = p+c+c*d-c = p+c*d = a*b$, and we are done.

We now prove correctness of the code. By the inductive principle, $p + c * d = a * b$ after line 12, which means that $p = a * b$, since $d = 0$. Thus the value returned is $a * b$.

2. The following function computes x^b for a real number x and a positive integer b. What is its loop invariant? Hint: Addition is to multiplication as multiplication is to exponentiation. The loop invariant is analogous to the loop invariant of the function `product` in the previous problem.

```
1  float power(float x, int b)
2   {
3    assert(b > 0);
4    float y = x;
5    int d = b;
6    float z = 1.0;
7    while(d > 0)
8     {
9      if(d % 2) z = z*y;
10     y = y*y;
11     d = d/2;
12    }
13   return z;
14  }
```

Our loop invariant is $z * y^d = x^b$.

We first prove the condition 1. Before line 7, $z * y^d = 1.0 * x^b = x^b$, and we are done.

For condition 2., Use y, d, and z for the values of those variables before one iteration, and y', d', and z' for the values of y, d, and z after the iteration. We are given that $z * y^d = x^b$, and we need to prove that $p' * (y')^{d'} = x^b$. From the code, we know that $y' = y * y$ and $d' = d/2$, with truncation as in C++. We also know that $z' = z$ if d is even, and $z' = z * y$ if d is odd.

Case I: d is even. Then the division $d/2$ is not truncated, so $d' = \frac{1}{2}d$. We have $z' * (y')^{d'} = z * (y * y)^{\frac{1}{2}d} = z * (y^2)^{\frac{1}{2}d} = z * y^d$ by the multiplication of exponents rule. We are done.

Case II: d is odd. Then $d' = \frac{1}{2}(d-1)$, hence $z' * (y')^{d'} = z * y * (y * y)^{\frac{1}{2}(d-1)} = z * y * (y^2)^{\frac{1}{2}(d-1)} = z * y * y^{d-1} = z * y^{1+d-1} = z * y^d = x^b$, and we are done.

We now prove correctness of the code. By the inductive principle, $z * y^d = x^b$ after line 12, which means that $z = x^b$, since $d = 0$ and $y^0 = 1$. Thus the value returned is x^b.

3. Most of the following recurrences can be solved using either the master theorem, the generalized master theorem, or the anti-derivative method. Use Θ in your answer if appropriate, otherwise O or Ω.

4. $F(n) = F(n-1) + n^2$

By the anti-derivative method: $F(n) = \Theta\left(\frac{n^3}{3}\right) = \Theta(n^3)$.

5. $F(n) = 3F(n/3) + n$

   $\log_3 3 = 1$, hence $F(n) = n \log n$ by the master theorem.

6. $F(n) = 3F(n/3) + 3F(2n/3) + n^3$

   Note that $\gamma = 3$. We need to find d such that $3(1/3)^d + 3(2/3)^d = 1$. We always try $d = \gamma$ first. $3(1/3)^3 + 3(2/3)^3 = 3/27 + 24/27 = 1$, hence $F(n) = \Theta(n^3 \log n)$ by the generalized master theorem.

7. $F(n) = F(n/2) + 2F(n/3) + n^2$

   Note that $\gamma = 2$. We need to find d such that $(1/2)^d + 2(1/3)^d = 1$. We try $d = \gamma$ first. We get $(1/2)^2 + 2(1/3)^2 = 1/4 + 2/9 < 1$. Thus, $d < 2$, which is all we need to know to get $F(n) = \Theta(n^2)$.

8. $F(n) = F(n/2) + 1$

   In this case, $A = 1$, $B = 2$, and $C = 0$, hence $\log_B A = C$, and we have $F(n) = \Theta(n^C \log n) = \Theta(\log n)$ by the master theorem.

9. $F(n) = F(\sqrt{n}) + 1$ (Hint: Use substitution: $m = \log n$ and $G(m) = F(n)$.) $F(\sqrt{n}) = G(\log(\sqrt{n})) = G(\frac{1}{2}(\log n)) = G(m/2)$. Thus, we have $G(m) = G(m/2) + 1$. By the result of the previous problem, $F(n) = G(m) = \Theta(\log m) = \Theta(\log \log n)$.

10. $F(n) = F(\log n) + 1$ (Hint: this was on a previous homework.) $F(n) = \Theta(\log^* n)$

11. $F(n) \leq F(n/5) + F(7n/10) + n$ (Hint: this should look familiar!) It is the recurrence we need to find the complexity of the median of medians algorithm. Since $(1/5) + (7/10) < 1$, $F(n) = O(n)$ by the generalized master theorem.

12. $F(n) = F(n - \sqrt{n}) + \sqrt{n}$

    By the anti-derivative method:

$$\begin{aligned} F(n) - F(n - \sqrt{n}) &= \sqrt{n} \\ \frac{F(n) - F(n - \sqrt{n})}{\sqrt{n}} &= 1 \\ F'(n) &= \Theta(1) \\ F(n) &= \Theta(n) \end{aligned}$$

13. $F(n) = F(n - \log n) + n \log n$

    By the anti-derivative method:

$$\begin{aligned} F(n) - F(n - \log n) &= n \log n \\ \frac{F(n) - F(n - \log n)}{\log n} &= n \\ F'(n) &= \Theta(n) \\ F(n) &= \Theta(n^2) \end{aligned}$$

14. $F(n) = F(n-1) + F(n/2) + 1$

    I cannot solve this one. (Seriously.)[1]

    Of course, I do not expect you to find a solution. But there are two facts which can be proved. (This is not an assignment, though.)

    (a) For any constant k, $F(n) = \Omega(n^k)$, that is, F grows faster than any polynomial function.

    (b) For any constant $k > 1$, $F(n) = O(k^n)$. That is, F grows slower than any strictly exponential function.

15. Name each of these seven algorithms.

    (a) Pick an element P from a set S, then partition S into two parts: those items which are less than P and those greater than P. Recorsively sort each part, and combine them to form a sorted list.

    Quicksort. $O(n^2)$ time, but if items are initially permuted randomly, the expected time is $\Theta(n \log n)$.

    (b) Divide a set S into two roughly equal parts. Recursively sort each part, then combine the two sorted parts to obtain a sorting of S.

    Mergesort. $\Theta(n \log n)$ time.

    (c) Given a sorted set S and an item x, you need to determine whether $x \in S$. Pick one element, say P, out of S. If $x = P$, you are done. If $x < P$, discard all items of S which are greater then P, while if $x > P$, all items of S which are less then P. Keep doing this until you either find x or you have discarded all items of S.

    Binary search. $O(\log n)$ time if you always pick P to be the median.

    (d) Given a set S, create an empty binary search tree T. Insert the items of S into T one at a time. Finally, visit and print the items of T in left-to-right order, also called inorder.

    Treesort, which is a sophisticated version of insertion sort. The worst case time complexity is quadratic, but if the items are permuted randomly, the expected time complexity is $O(n \log n)$, just like quicksort.

[1] You might wonder what the purpose of this problem is. I want you to realize that the world is not neatly tied up in blue ribbons. Not not every question has an answer, and not every question that has an answer has an answer that anyone can figure out.

(e) Given a set S, delete the least element of S and print it. Then delete the least remaining element of S and print it. Keep going until you have deleted and printed all elements of S.

Selection sort. The simple-minded implementation of this algorithm takes $\Theta(n^2)$ time. However, if the items are kept in a heap, you can do it in $\Theta(n \log n)$ time. (Heapsort.)

(f) All items of S are in a row. Agents run up and down the row, swapping any two adjacent items if the one on the right is less than the one on the left. Keep going until no more swaps are possible.

Bubblesort.

(g) You are grading a large number of exams, each of which is labeled with the ID of a student. Each ID consists of five numerals, and no two students have the same ID. You separate them into ten piles, based on the last digit of the ID. You then combine the piles in order, and separate them again into piles, based on the second to last digit of the ID. Do this five times. What will you accomplish?

Radix sort. They will be sorted, if you combine the piles in the correct order at every phase. This algorithm does not use the comparison model of computation.