This test consists of 2 questions on 4 pages (including this one). When you receive the signal to start, please make sure that your copy of the test is complete.

Please answer questions in the space provided. You will earn 20% for any question you leave blank or write “I cannot answer this question,” on. You will earn substantial part marks for writing down the outline of a solution and indicating which steps are missing.

Good Luck!
QUESTION 1. [8 marks]

Read over this short recursive program and its pre- and post-condition. Then prove that if \( \text{pow}(x, y) \) is executed with any \( x \) and \( y \) that satisfy the precondition, the postcondition must be satisfied. You should prove this using complete induction on \( y \). You may assume, without proof, that for any natural number \( y \), \( y = 2(y/2) + (y \% 2) \), and that for any natural number \( y \) greater than 1, \( 0 \leq (y/2), (y \% 2) < y \).

\( y/2 \) means \( y \) integer-divided by 2, or \( \lfloor y/2 \rfloor \).

```python
def pow(x, y):
    if y == 0:
        return 1
    elif y == 1:
        return x
    else:
        return pow(x, y//2) * pow(x, y//2) * pow(x, y%2)
```

**Precondition:** \( x \in \mathbb{R}, y \in \mathbb{N} \)

**Postcondition:** \( \text{pow}(x, y) \) terminates and returns \( x^y \).

**Sample solution:** Define \( P(y) \) : as "If \( x \in \mathbb{R} \) and \( \text{pow}(x, y) \) is executed, then it terminates and returns \( x^y \)." I prove that \( \forall y \in \mathbb{N}, P(y) \) using complete induction.

**Induction step:** Assume \( y \) is an arbitrary natural number, and that \( P(i) \) is true for natural numbers \( 0 \leq i < y \). There are two cases to consider.

**Case 1, \( y < 2 \):** In these cases, I can verify the claim directly. \( P(0) \) claims that if \( x \) is a real number, then \( \text{pow}(x, 0) \) returns \( x^0 = 1 \), and this is clearly what occurs, since \( y = 0 \) satisfies the if statement on the first line. \( P(1) \) claims that if \( x \) is a real number, then \( \text{pow}(x, 1) \) returns \( x^1 = x \), and this is clearly what occurs, since \( y \neq 0 \) falsifies the if statement on the first line, and then satisfies the elif test on the second line. So, in both cases, \( P(y) \) holds.

**Case 2, \( y > 1 \):** Since \( 0 \leq (y/2), (y \% 2) < y \), the induction hypothesis assumes \( P(y/2) \) and \( P(y/2) \). Also, \( y > 1 \) means that both the if and elif tests fail, so the else branch executes, returning:

\[
\text{pow}(x, y/2) \times \text{pow}(x, y/2) \times \text{pow}(x, y \% 2) = x^{y/2} \times x^{y/2} \times x^{y \% 2} \quad \# \text{by IH}
\]

\[
= x^{y/2+y/2+y \% 2} \quad \# \text{adding exponents}
\]

\[
= x^y \quad \# \text{by assumption}
\]

So, \( P(y) \) holds in this case also.

In all cases, if \( y \in \mathbb{N} \) and \( P(i) \) is true for natural numbers \( 0 \leq i < y \), then \( P(y) \) follows.

I conclude, by the principle of complete induction, \( \forall y \in \mathbb{N}, P(y) \).

Combining \( \forall y \in \mathbb{N}, P(y) \) with the fact that I assumed only that \( x \in \mathbb{R} \), this establishes that whenever \( y \in \mathbb{N} \) and \( x \in \mathbb{R} \) (i.e. the precondition is satisfied), then \( \text{pow}(x, y) \) terminates and returns \( x^y \) (the postcondition).
**Question 2.** [8 marks]

The definition of `pow` suggests the following recurrence for its time complexity, based on the value of `y`:

\[
T(y) = \begin{cases} 
1 & \text{if } y < 2 \\
1 + 2T([y/2]) & \text{otherwise}
\end{cases}
\]

Notice that the recursive call `pow(x, y%2)` takes constant time with respect to `y`, so it is represented in the recurrence by 1.

**Part (a) [2 marks]**

Use the Master Theorem (reprinted for your reading pleasure at the end of this test) to find the complexity class of `T`. Show your work.

**Sample solution:** `T(y)` has the form of a recurrence covered by the Master Theorem, where \( a = 2 \) (number of recursive calls), \( b = 2 \) (number of pieces the problem is divided into), and \( d = 0 \) (degree of polynomial bounding the amount of work to divide and later re-combine the problem). According to the Master Theorem, this means that:

\[
a = 2 > 1 = 2^0 = b^d
\]

...so `T` is in complexity class \( \Theta(y \log_2 y) = \Theta(y) \), or linear in `y`.

**Part (b) [2 marks]**

What would the complexity class of `T` be if it represented just 1 recursive call `pow(x, y/2)` rather than 2? Show your work.

**Sample solution:** Again, \( T(y) \) has the form of a recurrence covered by the Master Theorem, except now \( a = 1 \), \( b = 2 \), and \( d = 0 \), so

\[
a = 1 = 2^0 = b^0
\]

...and `T` is in complexity class \( \Theta(y \log y) = \Theta(\log y) \).
PART (c) [4 marks]

Modify the definition of \(\text{pow}\) so that it returns the same result as before, but with just one recursive call \(\text{pow}(x, y/2)\)

**Sample solution:** The idea is to avoid evaluating \(\text{pow}(x, y/2)\) twice, so that the cost of the recursive calculation is incurred only once. There are several ways to do this, and I choose to save the value

```python
def pow(x, y):
    if y == 0: return 1
    elif y == 1: return x
    else:
        p = pow(x, y//2)
        return p * p * pow(x, y%2)
```

---

**Master Theorem**

Suppose a recurrence expressing the complexity of a divide-and-conquer algorithm has the following form:

\[
T(n) = \begin{cases} 
  k & \text{if } n \leq B \\
  a_1T(\lceil n/b \rceil) + a_2T(\lfloor n/b \rfloor) + f(n) & \text{if } n > B
\end{cases}
\]

where \(B, k > 0, a_1, a_2 \geq 0,\) and \(a_1 + a_2 > 0,\) and \(f(n)\) is the cost of splitting and recombining.

If \(f \in \Theta(n^d),\) and \(a = a_1 + a_2,\) then

\[
T \in \begin{cases} 
  \Theta(n^d) & \text{if } a < b^d \\
  \Theta(n^d \log n) & \text{if } a = b^d \\
  \Theta(n^{\log_b a}) & \text{if } a > b^d
\end{cases}
\]

---

# 1: _____/ 8
# 2: _____/ 8

TOTAL: _____/16