1 Induction

(25 marks) Prove the following by induction. Given the general form of the geometric series:

\[ \sum_{i=0}^{\infty} ax^i = a + ax + ax^2 + \ldots + ax^n + \ldots \]

prove that when \( x \neq 1 \)

\[ \sum_{i=0}^{n} ax^i = a \frac{x^{n+1} - 1}{x - 1} \]

Answer:

Let \( P(n) = \text{“} \sum_{i=0}^{n} ax^i = a \frac{x^{n+1} - 1}{x - 1} \text{”} \).

Base case: for \( n = 0 \), prove \( P(0) = \text{“} \sum_{i=0}^{0} ax^i = a \frac{x^1 - 1}{x - 1} \text{”} \). Since \( x \neq 1 \), this is equivalent to \( a = a \frac{1}{1} \), which is true.

Inductive step: Assume \( P(n) \) is true, then show \( P(n+1) \).

\[
\sum_{i=0}^{n+1} ax^i = \sum_{i=0}^{n} ax^i + ax^{n+1} \\
= a \frac{x^{n+1} - 1}{x - 1} + ax^{n+1} \\
= a \frac{x^{n+1} - 1}{x - 1} + a \frac{x^{n+1}(x - 1)}{x - 1} \\
= a \frac{x^{n+1} - 1 + x^{n+1}(x - 1)}{x - 1} \quad \text{since } x - 1 \neq 0 \\
= a \frac{x^{n+1} - 1 + x^{n+2} - x^{n+1}}{x - 1} \\
= a \frac{x^{n+2} - 1}{x - 1}
\]

which proves the theorem.
2 Big-O, big-Ω, and big-Θ

(5 marks) (a) Show that $2^{n/2} \in O(2^n)$.

**Answer:**
Find constants $c$ and $n_o \geq 1$ such that $\forall n \geq n_o$ we have:

$$2^{n/2} \leq c \cdot 2^n$$

Let $c = 1$ and $n_o = 1$, then for all $n \geq n_o$:

$$2^{n/2} \geq 1$$

therefore:

$$2^n \geq 2^{n/2}$$

and:

$$c \cdot 2^n \geq 2^{n/2}$$

which is what we needed to prove.

(5 marks) (b) Show that $2^n \not\in O(2^{n/2})$.

**Answer:**
Show that there does not exist constants $c$ and $n_o \geq 1$ such that $\forall n \geq n_o$ we have:

$$2^n \leq c \cdot 2^{n/2}$$

Prove this by contradiction: suppose that there exists such constants. Therefore:

$$2^{n/2} \leq c$$

Choose $n = \max(n_o, 2 \log_2 c + 1)$, then

$$2^{n/2} \geq 2^{\log_2 c + 1} = c + 1 > c$$
(15 marks) (c) Indicate, for each pair of expressions \((A, B)\) in the table below, whether \(A\) is \(O\), \(\Omega\), or \(\Theta\) of \(B\). Assume that \(k \geq 1\) and \(\epsilon > 0\) are constants. Your answer should be in the form of the table with “yes” or “no” in each box. [Correct answer: 1 point, wrong answer = -.5]

<table>
<thead>
<tr>
<th>(A)</th>
<th>(B)</th>
<th>(A \in O(B))?</th>
<th>(A \in \Omega(B))?</th>
<th>(A \in \Theta(B))?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n^2)</td>
<td>(2^n)</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>(n)</td>
<td>(n^{\ln n})</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>(n^{\log_2 m})</td>
<td>(m^{\log_2 n})</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>(\log n!)</td>
<td>(\log n^n)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>(\log n)</td>
<td>(n^\epsilon)</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
3  Recurrence relations

(5 marks) (a) Consider the following pseudo-code for naive recursive matrix multiplication. Give a recurrence relation for its running time. You may use \( c_1, c_2, \ldots \) to denote constant times.

```plaintext
matrixMult(A, B, n) {
    // A and B are nxn matrices
    if n = 1 return A*B;  // scalar multiplication
    else {
        parity := n mod 2
        if parity = 1 then {
            add a row and a column of 0's to A and B
            n := n + 1
        }
        let A_11 A_12 := A and B_11 B_12 := B
        A_21 A_22 B_21 B_22
        C_11 := matrixMult(A11, B11, n/2) + matrixMult(A12, B21, n/2)
        C_12 := matrixMult(A11, B12, n/2) + matrixMult(A12, B22, n/2)
        C_21 := matrixMult(A21, B11, n/2) + matrixMult(A22, B21, n/2)
        C_22 := matrixMult(A21, B12, n/2) + matrixMult(A22, B22, n/2)
        let C_11 C_12 := C
        C_21 C_22
        if parity = 1 then {
            remove the last row and the last column from C
            n := n - 1
        }
    return C
}
```

Answer:
There are 8 recursive calls on \((n/2)\times(n/2)\) matrices. Addition takes time proportional to \(n^2\).

\[
T(n) = \begin{cases} 
  c_1 & \text{if } n = 1 \\
  8T(n/2) + c_2n^2 & \text{otherwise}
\end{cases}
\]
(10 marks) (b) Use recurrence trees (or the substitution method, or the iteration method) to determine an upper bound on the recurrence relation for matrixMult. NB: You do not have to give a proof of the upper bound you have found.

**Answer:**

With the substitution method:

\[
T(n) = 8T(n/2) + c_2n^2 \\
= 8(8T(n/4) + c_2(n/2)^2) + c_2n^2 \\
= 8^kT(n/2^k) + 8^{k-1}c_2(n/2^{k-1})^2 + \ldots + 8c_2(n/2)^2 + c_2n^2 \\
= c_18^k + c_2n^2 \sum_{i=0}^{k-1} \frac{8^i}{2^{2i}} \\
= c_18^k + c_2n^2 \sum_{i=0}^{k-1} \frac{2^{3i}}{2^{2i}} \\
= c_18^k + c_2n^2 \sum_{i=0}^{k-1} 2^i
\]

where \( k = \log_2 n \). Therefore:

\[
T(n) = c_18^{\log_2 n} + c_2n^2 \sum_{i=0}^{\log_2 n-1} 2^i \\
= c_1n^{\log_2 8} + c_2n^2 \sum_{i=0}^{\log_2 n-1} 2^i \\
= c_1n^3 + c_2n^2 \sum_{i=0}^{\log_2 n-1} 2^i \\
= c_1n^3 + c_2n^2 \frac{2^{\log_2 n} - 1}{2 - 1} \quad \text{geometric series, as in Question 1} \\
\leq c_1n^3 + c_2n^2 \frac{n^{\log_2 2}}{1} \\
= c_1n^3 + c_2n^3 \\
\in \mathcal{O}(n^3)
\]
(10 marks) (c) Use the Master Theorem to determine the complexity of the recurrence relation that was found for naive recursive matrix multiplication.

**Answer:**
In the Master Theorem, $a = 8$, $b = 2$, so $n^{\log_b a} = n^3$, and $f(n) = c_2n^2$. We can find a positive $\epsilon = 1$ such that:

$$f(n) = c_2n^2 \in \mathcal{O}(n^2) = \mathcal{O}(n^{3-\epsilon})$$

therefore, $T(n)$ is of the first case of the Master Theorem. Thus:

$$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^3)$$
4  Stacks and Queues

(25 marks) Using a stack ADT, write a pseudocode algorithm that reads in a sequence of characters in one pass from left to right from an input stream and returns true if and only if the {} and the () parentheses are balanced. For example, the following strings are balanced: “()”, “{(())}”, “{()())}”, but “(({{ ” and “{}})” are not. Follow the template given below.

Answer:

func isBalanced(instream IS): returns boolean
  S = new Stack();
  c = IS.getNextCharacter()
  while (c != END_OF_FILE) do
    if (c == '{') then
      S.push('{');
    else if (c == '(') then
      S.push('(');
    else if (c == '}') then
      if (S.isempty() or S.top() != '{') then
        return false;
      end;
      S.pop();
    else if (c == ')') then
      if (S.isempty() or S.top() != '(') then
        return false;
      end;
      S.pop();
    end;
    c = IS.getNextCharacter();
  end;
  return (S.isempty());
end;
5 Bonus

(5 marks) (a) Show for positive integer constants \(a, b\), that \((n + a)^b \in \Theta(n^b)\).

**Answer:**
First show that \((n + a)^b \in \mathcal{O}(n^b)\). We have
\[
(n + a)^b = n^b + c_1 n^{b-1} a + \ldots + a^b,
\]
for some constants \(c_1, \ldots\) that correspond to the binomial coefficients. Let \(c = 1 + c_1 a + \ldots + a^b\).
Therefore, for all \(n \geq 1\):
\[
(n + a)^b \leq n^b(1 + c) = cn^b \in \mathcal{O}(n^b).
\]
Finally, show that \((n + a)^b \in \Omega(n^b)\).
\[
(n + a)^b \geq n^b \in \Omega(n^b).
\]

(5 marks) (b) For each statement, say whether it is true or false. Denote the (worst case) running time of an algorithm \(A_i\) on an input of length \(n\) by \(T_{A_i}(n)\). (1 point each)

(i) If algorithms \(A_1\) and \(A_2\) produce solutions to the same problem, then \(T_{A_1}(n)\) is in \(\Theta(T_{A_2}(n))\).

**Answer:**
FALSE: For instance, merge sort and selection sort have different running times.

(ii) If \(T_{A_1}(n)\) is in \(\mathcal{O}(T_{A_2}(n))\), then \(T_{A_2}(n)\) is in \(\Omega(T_{A_1}(n))\).

**Answer:**
TRUE: By definition.

(iii) For \(n\) large enough, if \(T_{A_1}(n)\) is in \(\Theta(n \log n)\), then all such problems of size \(n\) require a running time of at least \(n\) or \(\log n\).

**Answer:**
FALSE: Some, but not necessarily all, instances of size \(n\) do.

(iv) For \(n\) large enough, if \(T_{A_1}(n)\) is in \(\Theta(n \log n)\), then all instances of size \(n\) can be solved within time at most \(n^2\).

**Answer:**
TRUE: For \(n\) large enough, \(n^2 > cn \log n\), for any positive constant \(c\).
(v) For \( n \) large enough, if \( T_{\ell_1}(n) \) is in \( \Theta(n) \), it is still possible that some instances of size \( n \) are solved within time at least \( n^2 \).

**Answer:**
FALSE: For \( n \) large enough, \( n^2 > cn \), for any positive constant \( c \). That \( T_{\ell_1}(n) \) is in \( \Theta(n) \) is true in general, i.e. for all instances of the problem.