MATH 243E Test #1 Solutions

(1) (a) Convert the decimal number 157 into octal (base 8). (5 points)

Solution: Repeatedly divide 157 by 8, until the quotient is zero, then take the remainders in reverse order:

\[
\begin{align*}
0 & \text{ R 2} \\
8 & \mid \underline{157} \\
2 & \text{ R } 3 \\
8 & \mid \underline{19} \\
1 & \text{ R 5} \\
8 & \mid \underline{2} \\
1 & \text{ R 7}
\end{align*}
\]

The answer is thus 235. Grading: +4 points for 532.

(b) Convert the octal number 157 into decimal. (5 points)

Solution: Recall that 157 (base 8) is an abbreviation for

\[1 \cdot 8^2 + 5 \cdot 8^1 + 7 \cdot 8^0 = 64 + 40 + 7 = 111,\]

in base 10.

(2) Prove that \( f(x) = 2x^5 + 3x^4 + 2x^2 - 3x + 1 \) is \( O(x^5) \). Use the definition of big-O notation. (10 points)

Solution: The definition of the big-O notation says that we need to find constants \( C \) and \( k \) such that if \( x \geq k \), then \( f(x) \leq Cx^5 \). One such pair that works is \( C = 8 \) and \( k = 1 \). If \( x \geq 1 \), then

\[ f(x) = 2x^5 + 3x^4 + 2x^2 - 3x + 1 \leq 2x^5 + 3x^5 + 2x^5 + 0 + x^5 = 8x^5. \]

(Note that \(-3x \leq -3x^5 \) if \( x \geq 1 \); instead of this incorrect statement, the correct \(-3x \leq 0 \) was used.) Many other answers are possible. Grading: +5 points for stating the definition of big-O notation, +5 points for the resulting computation; −1 point for using the statement “\(-3x \leq -3x^5 \).”
(3) Let \( A_i = \{i, i+1, \ldots, n+i-1\} \), for \( i = 1, \ldots, n \). What are \( \bigcup_{i=1}^{k} A_i \) and \( \bigcap_{i=1}^{k} A_i \), where \( 1 \leq k \leq n \)? (2 \times 10 \text{ points})

Solution: It helps to write out a few \( A_i \)'s to see what they look like:

\[
A_1 = \{1, 2, 3, 4, \ldots, n\}
\]
\[
A_2 = \{2, 3, 4, \ldots, n, n+1\}
\]
\[
A_3 = \{3, 4, \ldots, n, n+1, n+2\}
\]
\[
A_4 = \{4, \ldots, n, n+1, n+2, n+3\}
\]

Also, \( A_k = \{k, k+1, \ldots, n+k-1\} \). The union will then be a contiguous set of integers, from 1 to the last element of \( A_k \), which is \( n+k-1 \); hence

\[
\bigcup_{i=1}^{k} A_i = \{1, \ldots, n+k-1\}.
\]

For the intersection, note that the condition that \( k \leq n \) implies that the intersection will be the integers from the first element of \( A_k \) to the last one of \( A_1 \). Hence

\[
\bigcap_{i=1}^{k} A_i = \{k, k+1, \ldots, n\}.
\]

(If \( k > n \), then the intersection is actually empty, since no integer less than \( n \) is in \( A_k \).) Grading: +10 points for writing out \( A_i \); +5 for evaluating the union and intersection. +4 total points were awarded for any work on the problem, and +16 points were awarded for only considering the case \( k = n \).

(4) Give a big-\( O \) estimate for the function \( f(x) \) below. For the function \( g \) in your estimate that \( f(x) \) is \( O(g(x)) \), use a simple function of the smallest order. (10 points)

\[
f(x) = (n^2 \log n + 7^5 n^2 + 5 \log n)(n + \log n) + (n^2 + \log n + 1)(n \log n + \sqrt{n} + 3 \log n)
\]

Solution: We work from the inside out, discarding any terms which grow slower. Note that we do not remove the \( \log n \) factor from products!

\[
(n^2 \log n + 7^5 n^2 + 5 \log n) \quad \cdot \quad (n + \log n) + (n^2 + \log n + 1) \quad \cdot \quad (n \log n + \sqrt{n} + 3 \log n)
\]

\[
\downarrow \quad \downarrow \quad \downarrow
\]

\[
O(n^2 \log n) \quad \cdot \quad O(n) + O(n^2) \quad \cdot \quad O(n \log n)
\]

\[
\downarrow \quad \downarrow \quad \downarrow
\]

\[
O(n^3 \log n) \quad + \quad O(n^3 \log n) \quad + \quad O(n^3 \log n)
\]

\[
O(n^3 \log n)
\]

+8 points were awarded for any function which grows faster than this function; -5 points if a factor was removed, and -3 points for removing \( n \) instead of \( \log n \).
(5) Evaluate the following expressions. (2 × 10 points)

(a) \( \sum_{k>0 \text{ and } k|14} k^2 \) \hspace{1cm} (b) \( \prod_{i=1}^{n} 3^i \)

Solution: Part (a): The “\( k > 0 \) and \( k|14 \)” underneath the \( \sum \) sign indicates that we should sum over all positive integers which divide evenly into 14; i.e., \( k \) is 1, 2, 7, and 14, in term. The sum is then

\[
\sum_{k>0 \text{ and } k|14} k^2 = 1^2 + 2^2 + 7^2 + 14^2 = 250.
\]

−5 points if “\( k|14 \)” was understood to mean “\( k \) is a multiple of 14”; –2 points if one of 1, 2, 7, 14 was omitted.

Part (b): The \( \prod \) symbol indicates that we multiply the terms, instead of adding them. Thus

\[
\prod_{i=1}^{n} 3^i = 3^1 \cdot 3^2 \cdots \cdot 3^n = 3^{n(n+1)/2}.
\]

Full credit was awarded to the answer \( 3^1 \cdot 3^2 \cdots \cdot 3^n \); –5 points if the values were added up.

(6) Let \( \mathcal{P}(A) \) denote the power set of \( A \).

(a) What is \( \mathcal{P}(\mathcal{P}([1])) \)? (5 points)

(b) What is \( \mathcal{P}(\mathcal{P}([\emptyset])) \)? (5 points)

Solution: The power set of \( A \) is the set of all subsets of \( A \). For (a) and (b), we will work the answer from the inside out. The set \{1\} only has two subsets: \( \emptyset \) and \{1\}. Hence \( \mathcal{P}([1]) = \{\emptyset, \{1\}\} \). This is a set with two elements; hence its subsets are \( \emptyset, \{\emptyset\}, \{\{1\}\}, \{\emptyset, \{1\}\} \). Thus

\[
\mathcal{P}(\mathcal{P}([1])) = \mathcal{P}\left(\{\emptyset, \{1\}\}\right) = \left\{\emptyset, \{\emptyset\}, \{\{1\}\}, \emptyset, \{\{1\}\}, \emptyset, \emptyset, \{\emptyset, \{1\}\}\right\}.
\]

Part (b) is actually easier: The empty set only has one subset (namely itself), so \( \mathcal{P}([\emptyset]) \) only contains one element; that is, \( \mathcal{P}([\emptyset]) = \{\emptyset\} \). Then

\[
\mathcal{P}(\mathcal{P}([\emptyset])) = \mathcal{P}([\emptyset]) = \{\emptyset\}.
\]

Grading: –2 points for \( \mathcal{P}([\emptyset]) = \emptyset \); no points (on either half) if something that was not a set was written down.
(7) Describe an algorithm with the following specifications: Given arrays \( f \) (with \( m \) elements) and \( S \) (with \( n \) elements), determine whether the function \( f \) is onto, where the domain of \( f \) is \( \{1, \ldots, m\} \), the codomain of \( f \) is \( \{S[1], \ldots, S[n]\} \), and \( f(x) = f[x] \). What is the running time of your algorithm? Assume that comparisons and computations take a constant amount of time. (20 points)

Solution: A function \( f : A \rightarrow B \) is onto if for every element \( b \in B \), there is an element \( a \in A \) such that \( f(a) = b \). In this case, \( A = \{1, \ldots, m\} \), \( B = \{S[1], \ldots, S[n]\} \), and \( f(k) \) was given by the \( k \)th element of the array \( f \). Hence, for \( f \) to be onto, every \( i \) between 1 and \( n \) must have a \( j \) between 1 and \( m \) such that \( f[j] \) is \( S[i] \).

The most straightforward algorithm is to loop over all values of \( i \), have a loop inside of that that runs through all values of \( j \). If there is no \( j \) such that \( f[j] \) is \( S[i] \), the algorithm returns FALSE (and terminates). If all values of \( i \) are exhausted, the algorithm returns a value of TRUE. Something like this:

\[
\text{for } i := 1 \text{ to } n \\
\quad \text{begin} \\
\quad \quad \text{Found := FALSE} \\
\quad \quad \text{for } j := 1 \text{ to } m \\
\quad \quad \quad \text{if } (f[j] = S[i]) \text{ Found := TRUE} \\
\quad \quad \quad \text{if (not Found) return FALSE} \\
\quad \text{end} \\
\text{return TRUE}
\]

As to the running time, we work our way from the inside out to the function. The statement which is buried the most is the if statement. It consists of a comparison and a (possible) assignment, and so takes a constant \( O(1) \) amount of time. This if statement is inside the for \( j \) loop, which is executed \( m \) times. Hence the running time of the for \( j \) loop is \( m \cdot O(1) = O(m) \).

The statements inside the for \( i \) loop consist of an assignment, the for \( j \) loop, and another if statement; together, these take time at most \( O(1) + O(m) + O(1) = O(m + 2) = O(m) \). The for \( i \) loop is executed at most \( n \) times, so this loop takes time at most \( n \cdot O(m) = O(mn) \). Lastly, there are an initialization statement and a return statement (as well as the for \( i \) loop); this makes up the whole of the function. Thus the running time for the function is \( O(1) + O(mn) + O(1) = O(mn + 2) = O(mn) \).

Grading: +5 points if the definition of an onto function was provided (automatic if the rest of the problem was right), +10 points for the pseudocode/description, +5 points for the analysis of the running time. Most people who had trouble with this problem didn’t know what an onto function was. The five points devoted to the analysis of the running time was applied to the function which they provided.