1. **Searching.** Given a sorted list $L$ containing exactly 16 distinct elements. We want to locate an element $x$ in the list.

(a) Suppose that $x$ is in $L$, at position 2. What is the number of comparisons made by Linear Search when searching for $x$? What is the number of comparisons made by Binary Search when searching for $x$?

(b) Suppose that $x$ is in $L$, at position 12. What is the number of comparisons made by Linear Search when searching for $x$? What is the number of comparisons made by Binary Search when searching for $x$?

(c) For what positions of $x$ in $L$ does Linear Search make fewer comparisons than Binary Search?

2. **Sorting and Searching.** Give the number of comparisons that will be performed by each sorting algorithm below if the input array of length $n$ is of the form $[1, 2, ..., n - 3, n - 2, n, n - 1]$ (i.e., sorted except for the last two elements). [Note: On a real exam, you would be given pseudocode for algorithms that are invoked like this. This said, you should be comfortable with algorithms from class and homework to potentially save time on the exam. For now, you can refer to the textbook or lecture slides for pseudocode.]

(a) MinSort (SelectionSort)

(b) BubbleSort

(c) InsertionSort

3. **Analyzing an Iterative Algorithm for Exponentiation.** Given a real number $x$ and a positive integer $n$, we want to compute the $n$th power of $x$, i.e., $x^n$. Here is an iterative algorithm to compute $x^n$:

$$\text{IterExponentiate}(x, n)$$

1. result $\leftarrow 1$
2. $i! \leftarrow 1$
3. WHILE ($i \leq n$)
   4. result $\leftarrow$ result * $x$
   5. $i \leftarrow i + 1$
4. Return result

(a) State a loop invariant that can be used to show that the algorithm $\text{IterExponentiate}$ is correct.

(b) Prove your loop invariant from part (a).

(c) Explain why the loop invariant implies that the algorithm $\text{IterExponentiate}$ is correct.
(d) Describe the running time of this algorithm in big-O notation, assuming that arithmetic operations take constant time. Justify your answer.

4. Iterative Algorithms and Loop Invariants

Given an array \( A[1, \ldots, n] \) of positive integers, we want to compute the maximum value in \( A \), i.e., the maximum among \( \{A[1], \ldots, A[n]\} \). Here is an iterative algorithm to compute the maximum:

\[
\text{IterFindMax}(A[1, \ldots, n])
\]

1. \( M \leftarrow 0 \)
2. FOR \( i \leftarrow 1 \) TO \( n \) DO:
3. \quad IF \( A[i] > M \), THEN \( M \leftarrow A[i] \)
4. Return \( M \)

(a) State a loop invariant that can be used to show that the algorithm IterFindMax is correct.
(b) Prove your loop invariant from part (a).
(c) Explain why the loop invariant that the algorithm IterFindMax is correct.
(d) Describe the running time of this algorithm in Big-O notation. Justify your answer.

5. Iterative Algorithms and Loop Invariants

Consider the pseudocode given below:

\[
\text{sequenceSum}(n: \text{integer})
\]

1. \( addSub := 0 \)
2. for \( i := 1 \) to \( n \)
3. \quad if \( i \%2 = 1 \) then
4. \quad \quad \quad \quad addSub := addSub + \( i \)
5. \quad \quad \quad \quad else
6. \quad \quad \quad \quad addSub := addSub - \( i \)
7. return \( addSub \)

Prove or disprove the following loop invariant: After the \( i^{th} \) iteration of the loop, the value of \( addSub \) will be \( addSub = \sum_{j=1}^{i} (-1)^j j \).

6. Iterative Algorithms and Loop Invariants

In the following problem, we are given a list \( A = a_1, \ldots, a_n \) of salaries of employees at our firm and two integers \( L \) and \( H \) with \( 0 \leq L \leq H \). We wish to compute the average salary of employees who earn between \( L \) and \( H \) (inclusive), and the number of such employees. If there are no employees in the range, we say that 0 is the average salary. Below is an iterative algorithm which takes as input \( A \), \( L \) and \( H \) and returns an ordered pair \((avg, N)\) where \( avg \) is the average salary of employees in the range, and \( N \) is the number of employees in the range.

\[
\text{IterAverageInRange}(A: \text{list of n integers}, L, H: \text{integers with } 0 \leq L \leq H)
\]

1. \( sum := 0 \)
2. \( N := 0 \)
3. for \( i := 1 \) to \( n \)
4. \quad if \( L \leq a_i \leq H \) then
5. \quad \quad \quad \quad \( sum := sum + a_i \)
6. \quad \quad \quad \quad \( N ++ \)
7. if \( N = 0 \) then
8. \quad return \( (0, 0) \)
9. return \( (sum/N, N) \)
(a) State a loop invariant that can be used to show the algorithm IterAverageInRange is correct.
(b) Prove your loop invariant from part (a).
(c) Conclude from the loop invariant that the algorithm AverageInRange is correct.
(d) Describe the running time of this algorithm in \( \Theta \) notation, assuming that comparisons and arithmetic operations take constant time. Justify your answer.

7. Best and Worst Case Suppose that we are adding two \( n \)-digit integers, using the usual algorithm learned in grade school. The operation that we count as unit-cost is a single-digit additions. For example, to add 48 plus 34, we would do three single-digit additions:

1. In the ones place, add \( 8 + 4 = 12 \).
2. In the tens place, add \( 4 + 3 = 7 \).
3. In the tens place, add \( 7 + 1 = 8 \).

(a) If \( n = 5 \), give an example of two \( n \)-digit numbers that would be a best-case input to the addition algorithm, in the sense that they would cause the fewest single-digit additions possible.
(b) In the best case, how many single-digit additions does this algorithm make when adding two \( n \)-digit numbers?
(c) If \( n = 5 \), give an example of two \( n \)-digit numbers that would be a worst-case input to the addition algorithm, in the sense that they would cause the most single-digit additions possible.
(d) In the worst case, how many single-digit additions does this algorithm make when adding two \( n \)-digit numbers?

8. Asymptotic Notation Answer the following questions:-
(a) Is \( \log(n!) \in \Omega(n \log n) \)? Why or why not?
(b) Is it always the case that \( f(2n) \in O(f(n)) \)? If yes, give a proof; if no, give a counterexample.

9. Asymptotic Notation For each part, answer True or False, and give a short justification for your answer. All logarithms are base 2.
(a) \( \sqrt{n^3} \in O(n^2) \).
(b) \( 8^{\log(n^2)} \in \Theta(n^6) \).
(c) \( \log(n) \in \Omega(\log(\log(n))) \).
(d) \( n \in \Omega(\sqrt{n}) \).
(e) If \( f \), \( g \), and \( h \) are functions from the natural numbers to the non-negative real numbers with \( f(n) \geq g(n) \forall n \geq 1 \), \( f(n) \in \Theta(h(n)) \), and \( g(n) \in \Theta(h(n)) \), then \( (f - g)(n) \in \Theta(h(n)) \).
(f) If \( f \), \( g \), and \( h \) are functions from the natural numbers to the non-negative real numbers with \( f(n) \in \Theta(h(n)) \) and \( g(n) \in \Theta(h(n)) \), then \( (f \ast g)(n) \in \Theta((h(n))^2) \).
(g) If \( f \) and \( g \) are functions from positive integers to positive integers, then \( f(g(n)) \in O(f(n)) \).

10. Master Theorem Apply the Master Theorem to determine the asymptotic complexity of \( T(n) \) in each of the following.
(a) \( T(n) = 8T(\frac{n}{2}) + 1000n^2 \)
(b) \( T(n) = 2T(\frac{n}{2}) + 10n \)
(c) \( T(n) = 2T(\frac{n}{2}) + n^2 \)

11. Recursive Algorithms Given an array \( A[1, \ldots, n] \) of positive integers, we want to compute the maximum value in \( A \), i.e., the maximum value among \( \{A[1], \ldots, A[n]\} \). Here is a recursive algorithm to compute the maximum:

\[
RecFindMax(A[1, \ldots, n])
\]
1. IF \( n = 1 \), THEN Return \( A[1] \)
2. \( M \leftarrow \text{FindMax}(A[1, \ldots, n-1]) \)
3. IF \( A[n] > M \), THEN \( M \leftarrow A[n] \)
4. Return \( M \)

(a) Prove by induction on \( n \) that \( \text{RecFindMax} \) correctly returns the maximum of \( \{A[1], \ldots, A[n]\} \).

(b) Write down a recurrence for the time taken by the \( \text{RecFindMax} \) algorithm, assuming that arithmetic operations take constant time.

(c) Use your answer from part (b) to determine the running time of this algorithm in Big-O notation.

12. Recursive Algorithms. Given a list \( A = a_1, \ldots, a_n \) of salaries of employees at our firm and two integers \( L \) and \( H \) with \( 0 \leq L \leq H \). We wish to compute the average salary of employees who earn between \( L \) and \( H \) (inclusive), and the number of such employees. If there are no employees in the range, we say that 0 is the average salary. Below is a recursive algorithm which takes as input \( A \), \( L \) and \( H \) and returns an ordered pair \((\text{avg}, N)\) where \( \text{avg} \) is the average salary of employees in the range, and \( N \) is the number of employees in the range.

\[
\text{RecAverageInRange}(A : \text{list of } n \text{ integers}, L, H : \text{integers with } 0 \leq L \leq H)
\]

1. if \( n = 0 \) then
2. return \((0, 0)\)
3. \( B := a_1, a_2, \ldots, a_{n-1} \)
4. \((\text{avg}, N) := \text{RecAverageInRange}(B, L, H)\)
5. if \( L \leq a_n \leq H \) then
6. return \(((\text{avg} \times N + a_n)/(N + 1), N + 1)\)
7. else
8. return \((\text{avg}, N)\)

(a) Prove by induction on \( n \) that for any input, the algorithm correctly returns the average salary and number of employees in the range.

(b) Write down a recurrence for the time taken by this algorithm, assuming that comparisons and arithmetic operations take constant time.

(c) Use your answer from part (b) to determine the running time of this algorithm in \( \Theta \) notation. Justify your answer.

13. Recursive Algorithms. Given a real number \( x \) and a positive integer \( n \), we wish to compute the \( n \)th power of \( x \), i.e., \( x^n \). Here is a recursive algorithm to compute \( x^n \):

\[
\text{RecExponentiate}(x, n)
\]

1. IF \( n = 1 \), THEN Return \( x \)
2. Return \( x \times \text{RecExponentiate}(x, n - 1) \)

(a) Prove by induction on \( n \) that for any real number \( x \), the \( \text{RecExponentiate} \) algorithm correctly returns \( x^n \).

(b) Write down a recurrence for the time taken by the \( \text{RecExponentiate} \) algorithm, assuming that arithmetic operations take constant time.

(c) Use your answer from part (b) to determine the running time of this algorithm in Big-O notation.
14. **Solving Recurrences** Suppose a function $f$ is defined by the following recursive formula, where $n$ is a positive integer.

$$f(n) = f(n - 1) + 2n - 1, \quad f(1) = 6$$

Use any method we learned in this class to obtain a closed-form formula for $f(n)$. 