ANSWER KEY

1. **E**
2. **D**
3. **C**
4. **B**
5. **C**
6. **E**
7. **C**
8. **A**
9. **A**
10. **C**
11. **B**
12. **D**
13. **A**
14. **E**
15. **A**
16. **B**
17. **B**
18. **A**
19. **D**
20. **A**
21. **C**
22. **D**
23. **C**
24. **E**
25. **D**
26. **D**
27. **B**
28. **B**
29. **B**
30. **A**
31. **D**
32. **B**
33. **C**

ANSWERS EXPLAINED

1. **(E)** The time and space requirements of sorting algorithms are affected by all three of the given factors, so all must be considered when choosing a particular sorting algorithm.

2. **(D)** Choice B doesn't make sense: The loop will be exited as soon as a value is found that does *not* equal `a[i]`. Eliminate choice A because, if `value` is not in the array, `a[i]` will eventually go out of bounds. You need the `i < n` part of the boolean expression to avoid this. The test `i < n`, however, must precede `value != a[i]` so that if `i < n` fails, the expression will be evaluated as false, the test will be short-circuited, and an out-of-range error will be avoided. Choice C does not avoid this error. Choice E is wrong because both parts of the expression must be true in order to continue the search.

3. **(C)** The binary search algorithm depends on the array being sorted. Sequential search has no ordering requirement. Both depend on choice A, the length of the list, while the other choices are irrelevant to both algorithms.

4. **(B)** Inserting a new element is quick and easy in an unsorted array—just add it to the end of the list. Computing the mean involves finding the sum of the elements and dividing by n, the number of elements. The execution time is the same whether the list is sorted or not. Operation II, searching, is inefficient for an unsorted list, since a sequential search must be used. In `sortedArr`, the efficient binary search algorithm, which involves fewer comparisons, could be used. In fact, in a sorted list, even a sequential search would be more efficient than for an unsorted list: If the search item were not in the list, the search could stop as soon as the list elements were greater than the search item.

5. **(C)** Suppose the array has 1000 elements and x is somewhere in the first 8 slots. The algorithm described will find x using no more than five comparisons. A binary search, by contrast, will chop the array in half and do a comparison six times before examining elements in the first 15 slots of the array (array size after each chop: 500, 250, 125, 62, 31, 15).