

EXERCISES

1. Mark the following statements as true or false.
 - a. A sequential search of a list assumes that the list elements are sorted in ascending order.
 - b. A binary search of a list assumes that the list is sorted.
 - c. A binary search is faster on ordered lists and slower on unordered lists.
 - d. A binary search is faster on large lists, but a sequential search is faster on small lists.
 - e. When you declare a `vector` object and specify its size as 10, then only 10 elements can be stored in the object.

2. Consider the following list:

63 45 32 98 46 57 28 100

Using a sequential search, how many comparisons are required to determine whether the following items are in the list or not? (Recall that comparisons mean item comparisons, not index comparisons.)

a. 90 b. 57 c. 63 d. 120

3. a. Write a version of the sequential search algorithm that can be used to search a sorted list.
- b. Consider the following list:

5 12 17 35 46 65 78 85 93 110 115

Using a sequential search on ordered lists, which you designed in (a), how many comparisons are required to determine whether the following items are in the list or not? (Recall that comparisons mean item comparisons, not index comparisons.)

i. 35 ii. 60 iii. 78 iv. 120

4. Consider the following list:

2 10 17 45 49 55 68 85 92 98 110

Using the binary search, how many comparisons are required to determine whether the following items are in the list or not? Show the values of `first`, `last`, and `middle` and the number of comparisons after each iteration of the loop.

a. 15 b. 49 c. 98 d. 99

5. Sort the following list using the bubble sort algorithm as discussed in this chapter. Show the list after each iteration of the outer `for` loop.

26, 45, 17, 65, 33, 55, 12, 18

6. Sort the following list using the selection sort algorithm as discussed in this chapter. Show the list after each iteration of the outer `for` loop.

36, 55, 17, 35, 63, 85, 12, 48, 3, 66

7. Consider the following list:

5, 18, 21, 10, 55, 20

The first three keys are in order. To move 10 to its proper position using the insertion sort as described in this chapter, exactly how many key comparisons are executed?

8. Consider the following list:

7, 28, 31, 40, 5, 20

The first four keys are in order. To move 5 to its proper position using the insertion sort as described in this chapter, exactly how many key comparisons are executed?

9. Consider the following list:

28, 18, 21, 10, 25, 30, 12, 71, 32, 58, 15

This list is to be sorted using the insertion sort algorithm as described in this chapter. Show the resulting list after six passes of the sorting phase—that is, after six iterations of the `for` loop.

10. Recall the insertion sort algorithm as discussed in this chapter. Assume the following list of keys:

18, 8, 11, 9, 15, 20, 32, 61, 22, 48, 75, 83, 35, 3

Exactly how many key comparisons are executed to sort this list using insertion sort?

11. Suppose that `L` is a list of 10,000 elements. Find the average number of comparisons made by bubble sort, selection sort, and insertion sort to sort `L`.
12. a. It was remarked in this chapter that the performance of bubble sort can be improved if we stop the sorting process as soon as we find that in an iteration, no swapping of elements takes place. Write a function that implements bubble sort algorithm using this fact.
- b. Using the algorithm that you designed in part (a), find the number of iterations that are needed to sort the list: 65, 14, 52, 43, 75, 25, 80, 90, 95.
13. Suppose that `L` is a sorted list of 4096 elements. What is the maximum number of comparisons made by binary search to determine whether an item is in `L`?
14. Suppose that the elements of a list are in descending order, and they need to be put in ascending order. Write a C++ function that takes as input an array of items in descending order and the number of elements in the array. The function rearranges the element of the array in ascending order. Your function must not incorporate any sorting algorithms, that is, no item comparisons should take place.

15. To use a `vector` object in a program, which header file must be included in the program?
16. What do the following statements do?
 - a. `vector<int> list(50);`
 - b. `vector<string> nameList;`
17. What is the output of the following C++ code?


```
vector<int> intList(5);
int i;

for (i = 0; i < 5; i++)
    intList[i] = 2 * i + 1;

for (i = 0; i < 5; i++)
    cout << intList.at(i) << " ";

cout << endl;
```
18. What is the output of the following C++ code?


```
vector<string> classList;

classList.push_back("Nisha");
classList.push_back("Tony");
classList.push_back("Bobby");
classList.push_back("Peter");

for (unsigned int i = 0; i < classList.size(); i++)
    cout << classList[i] << " ";

cout << endl;
```
19.
 - a. Write a C++ statement that declares `secretList` to be a `vector` object to store integers. (Do not specify the size of `secretList`.)
 - b. Write C++ statements to store the following values, in the order given, into `secretList`:
56, 28, 32, 96, 75
 - c. Write a `for` loop that outputs the contents of `secretList`. (Use the expression `secretList.size()` to determine the size of `secretList`.)
20. What is the output of the following C++ code?


```
vector<int> intList(10);

for (int i = 0; i < 10; i++)
    intList[i] = 2 * i + 5;

cout << intList.front() << " " << intList.back() << endl;
```

21. Suppose that you have the following C++ code:

```
vector<int> myList(5);

unsigned int length;

myList[0] = 3;
for (int i = 1; i < 4; i++)
    myList[i] = 2 * myList[i - 1] - 5;

myList.push_back(46);
myList.push_back(57);
myList.push_back(35);
```

- a. Write a C++ statement that outputs the first and the last elements of `myList`. (Do not use the array subscripting operator or the index of the elements.)
 - b. Write a C++ statement that stores the size of `myList` into `length`.
 - c. Write a `for` loop that outputs the elements of `myList`.
22. What is the difference between the size and capacity of a vector?

PROGRAMMING EXERCISES

1. Write a program to test the function `seqOrderedSearch`. Use either the function `bubbleSort` or `selectionSort` to sort the list before the search.
2. Write a program to test the function `binarySearch`. Use either the function `bubbleSort` or `selectionSort` to sort the list before the search.
3. Write a function, `remove`, that takes three parameters: an array of integers, the number of elements in the array, and an integer (say, `removeItem`). The function should find and delete the first occurrence of `removeItem` in the array. If the value does not exist or the array is empty, output an appropriate message. (Note that after deleting the element, the number of elements in the array is reduced by 1.) Assume that the array is unsorted.
4. Write a function, `removeAt`, that takes three parameters: an array of integers, the number of elements in the array, and an integer (say, `index`). The function should delete the array element indicated by `index`. If `index` is out of range or the array is empty, output an appropriate message. (Note that after deleting the element, the number of elements in the array is reduced by 1.) Assume that the array is unsorted.
5. Write a function, `removeAll`, that takes three parameters: an array of integers, the number of elements in the array, and an integer (say, `removeItem`). The function should find and delete all of the occurrences of `removeItem` in the array. If the value does not exist or the array is empty, output an appropriate message. (Note that after deleting the element, the number of elements in the array is reduced.) Assume that the array is unsorted.

6. Redo Exercises 3, 4, and 5 for a sorted array.
7. Write a function, `insertAt`, that takes four parameters: an array of integers, the number of elements in the array, an integer (say, `insertItem`), and an integer (say, `index`). The function should insert `insertItem` in the array at the position specified by `index`. If `index` is out of range, output an appropriate message. (Note that `index` must be between 0 and the number of elements in the array.) Assume that the array is unsorted.
8. Write a version of the selection sort algorithm that can be used to sort a list of strings.
9. Write a version of the binary search algorithm that can be used to search a list of strings. (Use the selection sort that you designed in Exercise 8 to sort the list.)
10. Write a version of the sequential search algorithm that can be used to search a string `vector` object. Also, write a program to test your algorithm.
11. Write a version of the bubble sort algorithm that can be used to sort a string `vector` object. Also, write a program to test your algorithm.
12. Write a version of the selection sort algorithm that can be used to sort a string `vector` object. Also, write a program to test your algorithm.
13. Write a program to test the insertion sort algorithm as given in this chapter.
14. Write a version of the insertion sort algorithm that can be used to sort a string `vector` object. Also, write a program to test your algorithm.
15. Write a version of the binary search algorithm that can be used to search a string `vector` object. Also, write a program to test your algorithm. (Use the selection sort algorithm you developed in Programming Exercise 12 to sort the vector.)
16. Write a program that creates three identical arrays, `list1`, `list2`, and `list3` of 5000 elements. The program then sorts `list1` using bubble sort, `list2` using selection sort, and `list3` using insertion sort and outputs the number of comparisons and item assignments made by each sorting algorithm.
17. Write a program to test the function you designed in Exercise 12 to improve the performance of bubble sort.
18. Write a program to test the function you designed in Exercise 14.
19. Write a program that reads in a set of positive integers, representing test scores for a class, and outputs how many times a particular number appears in the list. You may assume that the data set has, at most, 100 numbers and that `-999` marks the end of the input data. The numbers must be output in increasing order. For example, for the data:

55 80 78 92 95 55 78 53 92 65 78 95 85 92 85 95 95