Search Algorithms

Lab Assignment 1: Binary search

Binary search can be implemented as a recursive algorithm. Each call makes a recursive call on one-half of the list the call received as an argument.

Complete the recursive function BinarySearch() with the following specifications:

1. Parameters:
   - a target integer
   - a vector of integers
   - lower and upper bounds within which the recursive call will search
2. Return value:
   - the index within the vector where the target is located
   - -1 if target is not found

The template provides the main program and a helper function that reads a vector from input.

The algorithm begins by choosing an index midway between the lower and upper bounds.

1. If **target == integers.at(index)** return index
2. If **lower == upper**, return -1 to indicate not found
3. Otherwise call the function recursively on half the vector parameter:
   - If **integers.at(index) < target**, search the vector from index + 1 to upper
   - If **integers.at(index) > target**, search the vector from lower to index - 1

The vector must be ordered, but duplicates are allowed.

Once the search algorithm works correctly, add the following to BinarySearch():

1. Count the number of calls to BinarySearch().
2. Count the number of times when the target is compared to an element of the vector. Note: lower == upper should not be counted.

Hint: Use a global variable to count calls and comparisons.

The input of the program consists of:

1. the number of integers in the vector
2. the integers in the vector
3. the target to be located

Ex: If the input is:

In [ ]:
```
9
1 2 3 4 5 6 7 8 9
2
```

the output of the program is:

In [ ]:
```
index: 1, recursions: 2, comparisons: 3
```

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Read integers from input and store them in a vector.
// Return the vector.
vector<int> ReadIntegers() {
  int size;
  cin >> size;
  vector<int> integers(size);
  for (int i = 0; i < size; ++i) {             // Read the numbers
     cin >> integers.at(i);
     }
sort(integers.begin(), integers.end());
  return integers;
}

int BinarySearch(int target, vector<int> integers, int lower, int upper) {
   /* Type your code here. */
}

int main() {
   int target;
   int index;

   vector<int> integers = ReadIntegers();

   cin >> target;

   index = BinarySearch(target, integers, 0, integers.size() - 1);
   printf("index: %d, recursions: %d, comparisons: %d\n",
          index, recursions, comparisons);

   return 0;
}
```

Assignment 1 Tests:

Apply the following 5 tests for 10 points

1. Compare output (2 points)

When input is

```
9
1 2 3 4 5 6 7 8 9
2
```

Standard output exactly matches

```
index: 1, recursions: 2, comparisons: 3
```

2. Compare output (2 points)

When input is

```
9
11 22 33 44 55 66 77 88 99
11
```

Standard output exactly matches

```
index: 0, recursions: 3, comparisons: 5
```

3. Compare output (2 points)

When input is

```
8
10 15 20 25 30 35 40 45
50
```

Standard output exactly matches

```
index: -1, recursions: 4, comparisons: 7
```

4. Compare output (2 points)

When input is

```
13
10 20 20 20 20 20 25 30 35 40 45 50 60
20
```

Standard output exactly matches

```
index: 2, recursions: 2, comparisons: 3
```

5. Unit test (2 points)

Test BinarySearch(99, {11 22 33 44 55 66 77 88 99}, 0, 8). Should return 8.

\

Sort Algorithms

Lab Assignment 2: Insertion sort

The program has four steps:

1. Read the size of an integer array, followed by the elements of the array (no duplicates).
2. Output the array.
3. Perform an insertion sort on the array.
4. Output the number of comparisons and swaps performed.

main() performs steps 1 and 2.

Implement step 3 based on the insertion sort algorithm in the book. Modify InsertionSort() to:

- Count the number of comparisons performed.
- Count the number of swaps performed.
- Output the array during each iteration of the outside loop.

Complete main() to perform step 4, according to the format shown in the example below.

Hints: In order to count comparisons and swaps, modify the while loop in InsertionSort(). Use global variables for comparisons and swaps.

The program provides three helper functions:

In [ ]:
```
// Read size numbers from cin into a new array and return the array.
int* ReadNums(int size)

// Print the numbers in the array, separated by spaces
// (No space or newline before the first number or after the last.)
void PrintNums(int nums[], int size)

// Exchange nums[j] and nums[k].
void Swap(int nums[], int j, int k)
```

Ex: When the input is:

In [ ]:
```
6 3 2 1 5 9 8
```

the output is:

In [ ]:
```
3 2 1 5 9 8

2 3 1 5 9 8
1 2 3 5 9 8
1 2 3 5 9 8
1 2 3 5 9 8
1 2 3 5 8 9

comparisons: 7
swaps: 4
```

```
#include <iostream>
using namespace std;

// Read size numbers from cin into a new array and return the array.
int* ReadNums(int size) {
  int *nums = new int[size];
  for (int i = 0; i < size; ++i) {          // Create array
     cin >> nums[i];                        // Read the numbers
     }
  return nums;
}

// Print the numbers in the array, separated by spaces
// (No space or newline before the first number or after the last.)
void PrintNums(int nums[], int size) {
  for (int i = 0; i < size; ++i) {
     cout << nums[i];
     if (i < size - 1) {
        cout << " ";
     }
  }
  cout << endl;
}

// Exchange nums[j] and nums[k].
void Swap(int nums[], int j, int k) {
  int temp;
  temp = nums[j];
  nums[j] = nums[k];
  nums[k] = temp;
}

// Sort numbers
/* TODO: Count comparisons and swaps.
         Output the array at the end of each iteration. */
void InsertionSort(int numbers[], int size) {
  int i;
  int j;

  for (i = 1; i < size; ++i) {
     j = i;
     while (j > 0 && numbers[j] < numbers[j - 1]) {
        Swap(numbers, j, j - 1);
        --j;
     }
  }
}

int main() {
  // Step 1: Read numbers into an array
  int size;
  cin >> size;                              // Read array size
  int* numbers = ReadNums(size);            // Read numbers

  // Step 2: Output the numbers array
  PrintNums(numbers, size);
  cout << endl;

  // Step 3: Sort the numbers array
  InsertionSort(numbers, size);
  cout << endl;

  // Step 4: Output the number of comparisons and swaps
  /* TODO: Output the number of comparisons and swaps performed */


  return 0;
}
```

Assignment 2 Tests:

Apply the following 5 tests for 10 points

1. Compare output (2 points)

When input is

```
6 3 2 1 5 9 8
```

Standard output exactly matches

```
3 2 1 5 9 8

2 3 1 5 9 8
1 2 3 5 9 8
1 2 3 5 9 8
1 2 3 5 9 8
1 2 3 5 8 9

comparisons: 7
swaps: 4
```

2. Compare output (2 points)

When input is

```
8 1 2 3 4 5 6 8 9
```

Standard output exactly matches

```
1 2 3 4 5 6 8 9

1 2 3 4 5 6 8 9
1 2 3 4 5 6 8 9
1 2 3 4 5 6 8 9
1 2 3 4 5 6 8 9
1 2 3 4 5 6 8 9
1 2 3 4 5 6 8 9

comparisons: 7
swaps: 0
```

3. Compare output (2 points)

When input is

```
8 9 7 6 5 4 3 2 1
```

Standard output exactly matches

```
9 7 6 5 4 3 2 1

7 9 6 5 4 3 2 1
6 7 9 5 4 3 2 1
5 6 7 9 4 3 2 1
4 5 6 7 9 3 2 1
3 4 5 6 7 9 2 1
2 3 4 5 6 7 9 1
1 2 3 4 5 6 7 9

comparisons: 28
swaps: 28
```

4. Compare output (2 points)

When input is

```
8 2 3 4 5 6 7 9 1
```

Standard output exactly matches

```
2 3 4 5 6 7 9 1

2 3 4 5 6 7 9 1
2 3 4 5 6 7 9 1
2 3 4 5 6 7 9 1
2 3 4 5 6 7 9 1
2 3 4 5 6 7 9 1
1 2 3 4 5 6 7 9

comparisons: 13
swaps: 7
```

5. Compare output (2 points)

When input is

```
8 2 1 4 3 6 5 9 7
```

Standard output exactly matches

```
2 1 4 3 6 5 9 7

1 2 4 3 6 5 9 7
1 2 4 3 6 5 9 7
1 2 3 4 6 5 9 7
1 2 3 4 6 5 9 7
1 2 3 4 5 6 9 7
1 2 3 4 5 6 9 7
1 2 3 4 5 6 7 9

comparisons: 10
swaps: 4
```

Submissions

Note: Do not forget to submit the TWO assignments and their corresponding test outputs to receive full credit.

1. Name your C++ files FirstName_Lastname_Binary_Search.cpp and FirstName_Lastname_Insertion_Sort.cpp.
2. Prepare your report in docx or pdf format and name it Firstname_Lastname.docx or Firstname_Lastname.pdf.
3. Add the screenshot of your codes to the report and provide a description for them. All tests should be performed and the result screenshot be included in the report.