Homework 2: Generic Recursive Binary Search

Binary search with a generic algorithm interface and a recursive implementation

Note: This assignment is used to assess some of the required ABET outcomes for the degree program. The outcomes assessed here are:

(a) an ability to apply knowledge of computing and mathematics appropriate to the discipline (divide-and-conquer recurrences)

(c) an ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs

(i) an ability to use current techniques, skills, and tools necessary for computing practice

These will be assessed using the following specific outcomes and scoring rubric

Rubric for Specific Outcomes i.-iv.		I	E	H	Key: I = ineffective, E = effective, H = highly effective
i.	Runtime/Runspace Analysis - Result	-	-	-	
ii.	Runtime/Runspace Analysis - Process	-	-	-	
iii.	Program Implementation - Base Case	-	-	-	
iv.	Program Implementation - Recursive Call	-	-	-	

In order to earn a course grade of C- or better, the assessment must result in Effective or Highly Effective for each specific outcome in the rubric.

Educational Objectives: After completing this assignment, the student should be able to accomplish the following:

- Define the concept *generic algorithm* in terms of the algorithm interface.
- Define the concepts of *recursive implementation* and *iterative implementation* of (generic) algorithms.
- Define a generic algorithm interface for a given specific algorithm.
- Define the concept of *divide and conquer* algorithm.
- Provide a recursive implementation of a generic divide and conquer algorithm.
- State the aymptotic runtime for specific divide and conquer algorithm
- Informally argue the correctness of your statement of the aymptotic runtime for specific divide and conquer algorithm
- Define the concepts *function class*, *predicate class*, *function object*, and *predicate object*.
- Use various function objects and predicate objects in client programs
- Use various function objects and predicate objects as arguments in generic algorithms

Operational Objectives: Create generic algorithm interfaces for **g_lower_bound**, **g_upper_bound**, and **g_binary_search** that operate on ranges determined by random access iterators and an optional order predicate. Implement the first two as recursive implementations and the third with a call to the first. Put these algorithms in the **alt** namespace. Test all of your generic algorithms using fsu::Vector, fsu::Deque, and ordinary arrays, with and without an order predicate.

Deliverables: Two files `rbsearch.h` and `log.txt`.

Procedural Requirements

1. The official development/testing/assessment environment is the `linprog` machines and the gnu C++ compiler "g++".

2. Develop and thoroughly test six generic algorithms as specified below. Place the source code for these algorithms in the file `rbsearch.h`.

3. Maintain a log in the ascii (text) file `log.txt`. This log should reflext the chronological history of you work on this assignment and, in summary form at the end, it should address the ABET outcomes for the assignment (see above).

4. Turn in two files `rbsearch.h` and `log.txt` using the `hw2submit.sh` submit script.

   ***Warning:** Submit scripts do not work on the program and `linprog` servers. Use `shell.cs.fsu.edu` to submit this assignment. If you do not receive the second confirmation with the contents of your project, there has been a malfunction.*

Technical Requirements and Specifications

1. `g_lower_bound`, `g_upper_bound`, and `g_binary_search` should each have a generic algorithm interface taking three arguments: two iterators that specify a search range and a search value. A fourth optional argument for each of these is a predicate object that specifies the order in the search range.

2. All of the algorithms should be in the namespace `alt`.

3. All of the algorithms should be implemented either recursively or with a direct call to another of the algorithms in your set. No loops should be used in any of the implementations. (The 4-parameter versions of `g_lower_bound` and `g_upper_bound` can be implemented recursively, and all others can be implemented by making calls to one of these.)

4. Test your algorithms in each of these cases:
   1. Range from begin to end in a `fsu::Vector` object with default ordering.
   2. Range from begin to end in a `fsu::Vector` object with reverse ("greater than") ordering.
   3. Range from begin to end in a `fsu::Deque` object with default ordering.
   4. Range from begin to end in a `fsu::Deque` object with reverse ("greater than") ordering.
   5. Range from begin to end in an array with default ordering.
   6. Range from begin to end in an array with reverse ("greater than") ordering.

5. Your submission will be assessed using a proprietary test harness. It is your responsibility to ensure correct behavior of your generic algorithms.

6. Be sure that your testing is documented in the log. Also be sure that your informal analysis of runtime and runspace is documented in the log.

Hints

- A sample executable of a test harness is posted as `LIB/area51/fgss.x`.

- The iterative versions of the binary search generic algorithms are located in `LIB/tcpp/gbsearch.h`. These are useful models for the generic algorithm interfaces in particular.

- One sample test source code is given in `LIB/hw2/vtest.cpp`. This code calls generic heap sort, which makes data input more convenient, and also illustrates a correct call to generic lower bound. You should modify this code so that it tests on deques and arrays, as well as vectors, and also test with the predicate object GT. (When you compile `vtest` as is, you will get a warning that GT is unused.)

- Your submission will be assessed using a proprietary test harness that is an elaboration of vtest.cpp.