San José State University
Department of Computer Engineering

CMPE 180-92

Data Structures and Algorithms in C++

Fall 2016
Instructor: Ron Mak

Assignment #9B (with Extra Credit)

Assigned:	Saturday, October 22
Due:	Friday, October 28 at 11:59 PM
URL:	http://codecheck.it/files/16102405141lugqpzfsxbmrpyxnro3f89sq
Canvas:	Assignment 9.b. Vector vs. linked List
Points:	100 with possible extra credit

Vector vs. linked list

In this assignment, you will compare the performances of the Standard Template Library (STL) vector and the STL linked list. A test program causes the vector and the linked list to undergo an identical set of operations, and it will time how long it takes each data structure to perform the operations. The program prints a table that compares the timings for vectors and lists of various sizes: 100, 500, 1000, 5000, 10000, 50000, and 100000.

Classes `SortedVector` (based on the STL vector) and `SortedList` (based on the STK list) keep their data sorted in smallest to largest order. They have an identical set of member functions whose performances will be tested and timed. You need to complete their class definitions in files **SortedVector.cpp** and **SortedList.cpp**, respectively.

Member function `prepend`

For each data structure, use an iterator to append an integer value at the beginning. You may assume that the data arrives in sorted order, largest to smallest. Therefore, you do not need to sort the data into smallest to largest order.

Member function `append`

For each data structure, append an integer value to the end. You may assume that the data arrives in sorted order, smallest to largest. Therefore, you do not need to sort the data.

Member function `find`

For each data structure, use a constant iterator to search for the given value. Return true if the value is found, else return false.

Member function `get_value`

For each data structure, return the value at the given index `i`. You do not need to check the range of the index.

For the list, use a constant iterator to access the `i`th element. Since you can walk a doubly linked list in both directions starting from its beginning or the end, also use a reverse constant iterator and start from the end whenever that is advantageous.

Member function `clear`

For each data structure, use an iterator to repeatedly erase (remove) the head element until the all the elements are gone.

Sample output

You output should be similar to:

```
         |---Prepend---|   |-----Gets----|   |----Search---|   |----Remove---|   |----Append---|
  Size   Vector     List   Vector     List   Vector     List   Vector     List   Vector     List
   100     0 ms     0 ms     0 ms     0 ms     0 ms     1 ms     0 ms     0 ms     0 ms     0 ms
   500     0 ms     0 ms     0 ms     0 ms     3 ms     4 ms     0 ms     0 ms     0 ms     0 ms
  1000     0 ms     0 ms     0 ms     1 ms     6 ms     9 ms     0 ms     0 ms     0 ms     0 ms
  5000     1 ms     0 ms     0 ms     7 ms    35 ms    54 ms     1 ms     0 ms     0 ms     0 ms
 10000     7 ms     1 ms     0 ms    13 ms    75 ms    88 ms     6 ms     1 ms     0 ms     1 ms
 50000   183 ms     7 ms     0 ms    65 ms   317 ms   428 ms   180 ms     5 ms     1 ms     5 ms
100000   777 ms    16 ms     0 ms   125 ms   611 ms   889 ms   793 ms    11 ms     2 ms    12 ms

Done!
```

Since each run on each machine will produce different timings, CodeCheck will not compare the output. However, you should see similar trends – which data structure is faster for certain operations, how the timings increase as the size increases, etc.

Function parameter

Note that in file **PerformanceTests.cpp**, the overloaded function **`elapsed_time`** has a function parameter. The function to be timed is passed in.

What to submit

Submit the signed zip file into **Canvas: Assignment 9.b. Vector vs. linked List**.

You can submit as many times as necessary to get satisfactory results, and the number of submissions will not affect your score. When you're done with your program, click the

“Download” link at the very bottom of the Report screen to download the signed zip file of your solution.

Run the program one more time, outside of CodeCheck. Add 500000 (500 thousand) to the sizes. Make a text file of the output that includes this extra size, and submit it along with the signed zip file. Do not run with this extra size inside of CodeCheck since that will cause a timeout error.

Extra credit!

For extra credit, analyze the performance figures in the output and answer the following questions:

1. [5 points] Explain the difference in timings between the vector and the linked list for the prepend operation.
2. [5 points] Explain the difference in timings between the vector and the linked list for the get value operation,
3. [5 points] Explain the difference in timings between the vector and the linked list for the search operation.
4. [5 points] Explain the difference in timings between the vector and the linked list for the remove operation.
5. [5 points] Explain the difference in timings between the vector and the linked list for the append operation.

For the following questions, change file **PerformanceTests.cpp** to loop the sizes from 5000 to 200000 (200 thousand) by 5000.

Tip: Make a separate run for each question and run only the required operations. For questions 7 and 8, you will also need to run the prepend operation in order to first load data into the data structures.

6. [30 points] Identify and graph the rates of growth (linear, quadratic, exponential, logarithmic, etc.) of the prepend operation as the size increases for both the vector and the linked list, and explain the growth rates.
7. [15 points] Identify, graph, and explain the rate of growth of the get value operation for the linked list.
8. [30 points] Identify, graph, and explain the rates of growth of the search operation for both the vector and the linked list.

Rubrics

Criteria	Maximum points
Correct program output (similar to the sample output)	**30**
• `SortedVector`:	• `SortedVector`:
○ prepend	○ 3
○ gets	○ 3
○ search	○ 3
○ remove	○ 3
○ append	○ 3
• `SortedList`:	• `SortedList`:
○ prepend	○ 3
○ gets	○ 3
○ search	○ 3
○ remove	○ 3
○ append	○ 3
Class definitions	**70**
• `SortedVector`:	• `SortedVector`:
○ prepend (use iterator)	○ 5
○ append	○ 5
○ find (use constant iterator)	○ 10
○ get_value	○ 10
○ clear (use iterator)	○ 5
• `SortedList`:	• `SortedList`:
○ prepend (use iterator)	○ 5
○ append	○ 5
○ find (use constant iterator)	○ 10
○ get_value (use constant iterators)	○ 10
○ clear (use iterator)	○ 5
Extra credit questions	**100**
• Question 1	• 5
• Question 2	• 5
• Question 3	• 5
• Question 4	• 5
• Question 5	• 5
• Question 6: vector prepend: identify the rate of growth	• 5
• Question 6: vector: graph	• 5
• Question 6: vector: explain	• 5
• Question 6: list prepend: identify the rate of growth	• 5
• Question 6: list: graph	• 5
• Question 6: list: explain	• 5
• Question 7: list get_value: identify the rate of growth	• 5
• Question 7: list: graph	• 5
• Question 7: list: explain	• 5
• Question 8: vector search: identify the rate of growth	• 5
• Question 8: vector: graph	• 5
• Question 8: vector: explain	• 5
• Question 8: list search: identify the rate of growth	• 5
• Question 8: list: graph	• 5
• Question 8: list: explain	• 5