CSE 330 LABORATORY -- Week 5, Spring 2018

Instructor: Kerstin Voigt

In this lab you will implement the Binary Search algorithm and experience its efficient O(log n) performance for presorted vectors.

Exercise 1: Implement the binary search function from your textbook on p. 67 and add an additional reference parameter 'int& steps' which is counting up the costs associated with the iterations needed to find the target value. When the target is found in the vector, binary_search returns the index at which the target is stored; otherwise, the function returns value -1.

Notice that we are using our own Vector data structure. So make sure to include "Vector.h"

```
int binary_search(int x, const Vector<int>& vec, int& steps)
{
  int low = 0;
  int high = vec.size() - 1;
  steps = 0;

  while (low <= high)
    {
      int mid = (low + high) / 2;

      steps++;

      if (vec[mid] < x)
         low = mid + 1;
      else if (vec[mid] > x)
         high = mid - 1;
      else
         return mid;
    }
  return -1; // not found
}
```

Test your function in an 'int main()' which builds up a vector of 25 arbitrary integer values in ascending order. For binary search to work, your vector must be sorted. The manner in which the binary_search function is written calls for the vector values to be in ascending order.

Run binary_search for 5-10 target values, some included in the vector, some not included. Let the main() function report

1. whether the value was found and
2. at what "cost" (you get this information from the int& parameter).

Examine your output, and discuss whether you see the theoretical O(log n) performance confirmed. For your vector of 25 values, what is the worst cost of (not) finding a target value. How does this cost compare to our total N of 25?

Increase your vector to a larger size and (automatically) fill it up with ascending values. Repeat the process of finding 5-10 target values. How high are the costs now?

Exercise 2: Implement an analogous binary search function for linked lists. Use your own List.h data structure and add to class List a function List<T>::nth(int, int&). You may use this one

```
int nth(int k, int& steps) const
  {
    steps = 0;
    if (k < 0 || k >= theSize)
      return -1;

    const_iterator itr = begin();
    if (k == 0)
      {
     steps++;
     return *itr;
      }
    for (int i = 1; i <=k; i++)
      {
     itr++;
     steps++;
      }
    return *itr;
  }
```

Again, notice how this function is set up to keep track of its “cost”; in this case, the cost is made up of the number of iterations that are needed to reach the nth item in the list. (Think! – where did we count this cost for Vector? …)

Now implement function

```
int binary_search(int x, const List<int>& vec, int& steps){…}
```

The body of this function will naturally be similar to the body of binary_search for Vector; however, some modifications are necessary and it is your task to figure out which and to apply them.

Test our function binary_search as in Exercise 1 with a sorted list of 25 integer values and 5-10 target values. Again, have your program report

1. whether the item was found, and
2. at what cost.

Discuss the performance of binary search on the linked list. Is the cost of finding a target item still O(log n) as the Binary Search Algorithm seems to promise?

You should find the answer is “NO”. –

For Homework Assignment 2,

…**complete Exercise 1 and Exercise 2** neatly in files BinarySearch.h, BinarySearch.cpp, and BinarySearchMain.cpp. Make sure that your implementations use our own data structures in Vector.h and List.h (not STL <vector> or <list>). No need to submit Vector.h or List.h

...**test both binary_search functions** for data structures of 32 data values (in ascending order). Attempt to find 10 target values each (some contained, some not contained), and have the program report on success/failure and cost.

…**submit hardcopies of all** .h and .cpp files, and a hardcopy of a **typescript** that shows your program compilation, program testing and output.

…**submit a well-written paragraph** in which you explain the reasons why binary search on linked lists data structures (while technically possible) is not a good idea, and not worthwhile implementing. Also indicate and justify the algorithm complexity O(?) that you can determine from both the C++ code and the output of your test runs. **This paragraph will be worth a substantial portion of your hw points, so make sure you put effort int your answer.**

… submit by DUE DATE, Thursday, May 10, at the time of Lab 6.2.

Credit for this lab: make sure you have your name on the signup sheet.

Credit for this lab: (1) Make sure to sign the **signup sheet**. (2) Nothing else to turn in, but keep your code handy for an upcoming next homework assignment.