Laboratory 13: Cover Sheet

Name: Ernest Landrito Date: October 2, 2013

Section: 1

Place a check mark in the *Assigned* column next to the exercises your instructor has assigned to you. Attach this cover sheet to the front of the packet of materials you submit following the laboratory.

Activities	**Assigned:** Check or list exercise numbers	**Completed**
Implementation Testing	✓	
Programming Exercise 1	✓	
Programming Exercise 2	✓	
Programming Exercise 3	✓	
Analysis Exercise 1		
Analysis Exercise 2		
	Total	

Laboratory 13: Implementation Testing

Name: Ernest Landrito Date: October 2, 2013

Section: 1

Check with your instructor whether you are to complete this exercise prior to your lab period or during lab.

Question 1: What is the resolution of your implementation—that is, what is the shortest time interval it can accurately measure?

The shortest time that the timer can measure is 0.00001 seconds.

Test Plan 13-1 (Timer ADT operations)			
Test case	**Actual time period (in seconds)**	**Measured time period (in seconds)**	**Checked**
0.005 sec	0.005	0.00567	
0.004 sec	0.004	0.004527	
0.003 sec	0.003	0.003419	
0.002 sec	0.002	0.00212	
0.001 sec	0.001	0.001144	
0.0009 sec	0.0009	0.001035	
0.0005 sec	0.0005	0.000592	
0.0001 sec	0.0001	0.000131	
0.00009 sec	0.00009	0.00012	
0.00005 sec	0.00005	0.000076	
0.00001 sec	0.00001	0.000032	

Laboratory 13: Measurement and Analysis Exercise 1

Name: Ernest Landrito Date: October 2, 2013

Section: 1

In the table below, fill in values of *N*, 2*N*, and 4*N*: try 1000, 2000, 4000. If you do not obtain meaningful timing data—especially for `binarySearch`—change the value of *N* and try again.

Timings Table 13-2 (Search routines execution times)			
Routine	**Number of keys in the list (numKeys)**		
	N = 4000	*2N = 8000*	*4N = 16000*
`linearSearch` O(N)	4.13344	*8.38374*	16.5995
`binarySearch` O(Log N)	0.02233	*0.02327*	0.02608
`STLSearch` O(N)	3.24400	*6.30039*	12.5956

Please list times in seconds

Question 1: How well do your measured times conform to the order-of-magnitude estimates given for the `linearSearch` and `binarySearch` routines?

The measured times conform to the order-of-magnitude estimates well. The linear search seems to follow a linear progression and the binary search also seems to follow a logarithmic progression.

Question 2: Using the code in the file *search.cpp* and your measured execution times as a basis, develop an order-of-magnitude estimate of the execution time of the `STLSearch` routine. Briefly explain your reasoning behind this estimate.

Based on the values found in the test executions, I find that the STLSearch is of magnitude N because it traverses the list as well as because the data follows a linear progression. It is not at slow as a plain linear search because it was optimized in stl.

Laboratory 13: Measurement and Analysis Exercise 2

Name: Ernest Landrito Date: October 2, 2013

Section: 1

In the table below, fill in values of *N*, 2*N*, and 4*N*: try 1000, 2000, 4000. If you do not obtain meaningful timing data—especially for `quickSort`—change the value of *N* and try again.

Timings Table 13-3 (Execution times of a set of sorting routines)			
Routine	**Number of keys in the list (numKeys)**		
	N = 4000	*2N = 8000*	*4N = 16000*
`selectionSort` O(N^2)	17.2236	72.2919	272.003
`quickSort` O(*N* Log *N*)	0.19008	0.40632	0.92194
STL `sort` O(*N Log N*)	0.12700	0.25950	0.58389

Please list times in seconds

Question 1: How well do your measured times conform with the order-of-magnitude estimates given for the `selectionSort` and `quickSort` routines?

For both the selection sort and quick sort routines, the found execution times seem to follow the order-of-magnitude given.

Question 2: Using the code in the file *sort.cpp* and your measured execution times as a basis, develop an order-of-magnitude estimate of the execution time of the STL `sort` routine. Briefly explain your reasoning behind this estimate.

Based on the execution time of the stl sort, I found the order-of-magnitude to be of N Log N. The time was similar to the quicksort as well as following the N Log N progression.s

Laboratory 13: Measurement and Analysis Exercise 3

Name: Ernest Landrito Date: October 2, 2013

Section: 1

In the table below, fill in values for the various constructor tests. Try an initial value of N=1000. If you do not obtain meaningful timing data, change the value of N and try again.

Timings Table 13-4 (Timing constructor/initialization just before vs. inside loop)		
	Constructor/initialization location	
Your value of N: 50000	Outside loop	Inside loop
`int`	0.005098	0.005304
`double`	0.004646	0.004904
`vector`	14.3357	29.5368
`TestVector`	14.2896	29.6105

Please list times in seconds

Question 1: For each data type, how do your measured times for the constructor just before the loop compare to the times for the constructor inside the loop? What might explain any observed differences?

For integers, the inside loop constructors turned out to be slower than those declared before the loop. Likewise for doubles vectors and TestVectors. A new action frame being added to the stack when the constructor is called could cause this delay, as opposed to calling the constructor once and overwriting the data when in the loop.

In the table below, fill in values for the various increment tests. Try an initial value of *N*=1000. If you do not obtain meaningful timing data, change the value of *N* and try again.

Timings Table 13-5 (Timing pre-/post-increment operators)		
Your value of *N*: 1000	pre-increment	post-increment
`int`	`0.03327`	`0.002943`
`double`	`0.00417`	`0.003613`
`TestVector`	`34.2601`	`35.7456`

Please list times in seconds

Question 2: For each data type, how do your measured times for the pre-increment operator compare to the times for the post-increment operator? What might explain any observed differences?

For int and double, the post increment times were faster than the pre-increment times. This could be caused by the fact that the post increment operator would already have the value passed and would then just simply increment the number whereas the pre increment operator would only know to add one to the next value then have to get the value passed to it.

For the TestVector, the post increment time was slower. I suspect that this is caused by the fact that the implementation of the TestVector operator overload has the post increment operator call the pre increment operator.