Laboratory 10: Cover Sheet

Name: Saharath Kleips Date: 10/28/2014

Section: 1001

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	

Name: Saharath Kleips Date: 10/28/2014

Section: 1001

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

Test Plan 10-1 (Hash Table ADT operations)

Test case	Commands	Expected result	Checked
Insert	+1	Inserted 1	✓
Insert	+2	Inserted 2	✓
Insert	+3	Inserted 3	✓
Retrieve	?1	Retrieved 1	✓
Retrieve	?2	Retrieved 2	✓
Retrieve	?3	Retrieved 3	✓
Remove	-1	Removed 1	✓
Remove	-2	Removed 2	✓
Remove	-3	Removed 3	✓

Laboratory 10: Programming Exercise 1

Name: Saharath Kleips

Date: 10/28/2014

Section: 1001

Test Plan 10-2 (Login Authentication Program)		
Test case	**Expected result**	**Checked**
jack jill jack broken.crown jack broken.crrrrown	0: 1: jack mary 2: 3: bopeep cole jill 4: 5: 6: simon 7: Login: Password: Authentication Failure Login: Password: Authentication Successful Login: Password: Authentication Failure Login:	

Laboratory 10: Programming Exercise 2

Name: Saharath Kleips Date: 10/28/2014

Section: 1001

Test Plan 10-3 (perfect minimal hash operation)		
Hash formula	**Expected result**	**Checked**

Laboratory 10: Programming Exercise 3

Name: Saharath Kleips

Date: 10/28/2014

Section: 1001

Discuss the results with your lab instructor.

Test Results Table 10-4 (stdDeviation operation)			
Hash function used	**Expected distribution quality (good/fair/poor)**	**Standard deviation**	**Measured relative distribution quality**

Hash Algorithm #1
`return 0;`
Hash Algorithm #2
`return int(str[0]) * 10 + str.length();`
Hash Algorithm #3
`double val = 0;`
`for (int i=0;`
`     i<str.length();`
`     i++)`
`{ val += (val*1.1)*str[i];}`
`return int(val);`
 	
Hash Algorithm #4
 	
Hash Algorithm #5			

Laboratory 10: Analysis Exercise 1

Name: Saharath Kleips Date: 10/28/2014

Section: 1001

Given a hash table of size *T*, containing *N* data items, develop worst-case, order-of-magnitude estimates of the execution time of the following Hash Table ADT operations, assuming they are implemented using singly-linked lists for the chained data items and a reasonably uniform distribution of data item keys. Briefly explain your reasoning behind each estimate.

insert O(n)

Explanation: You insert based on an index location of an array, which takes constant time. The worst-case scenario is if you would need to insert it into the bottom of that specific index location, but even then, the hashes would be the same and therefor still be linear time.

retrieve O(n)

Explanation: Same logic as the insert function. The worst-case scenario is if the value is at the end of the bottom of a specific index location, but would still be in linear time.

What if the chaining is implemented using a binary search tree instead of a singly-linked list? Using the same assumptions as before, develop worst-case, order-of-magnitude estimates of the execution time of the following Hash Table ADT operations. Briefly explain your reasoning behind each estimate.

insert O(log2(n))

Explanation: One again, locating the index location is constant time, but this time the insertion per index is based on a log2(n) time of the binary search tree. Therefore insert is log2(n) time with worst-case scenario being all inserted values being hashed into the same binary search tree and indexed to the bottom of the tree.

retrieve O(log2(n))

Explanation: Similarly, constant time for index locations, and then the retrieve is based on the binary search tree implementation making it log2(n). The worst-case scenario is if retrieval was from the bottom of a tree.

Name: Saharath Kleips Date: 10/28/2014

Section: 1001

Part A

For some large number of data items—e.g., *N*=1,000,000—would you rather use a binary search tree or a hash table for performing data retrieval? Explain your reasoning.

Under perfect conditions, a hash table would always be optimal... But with such a large amount of data items, collisions are bound to occur. If it was a perfectly implemented hash, then it is hard to beat retrieval in O (1) time.

Part B

Assuming the same number of data items given in Part A, would the binary search tree or the hash table be most memory efficient? Explain your assumptions and your reasoning.

BSTs are more memory efficient. BSTs allocate space as needed, you do not need to preallocate an array of (most likely) a ton of pointers that you might never use.

Part C

If you needed to select either the binary search tree or the hash table as the general purpose best data structure, which would you choose? Under what circumstances would you choose the other data structure as preferable? Explain your reasoning.

Memory issues aren't much of a concern anymore which makes hash tables a very optimal solution as the best data structure, but binary search trees are easier to implement and easier to read. There does not need to be any complicated hash function or programming to deal with problems such as collisions. In this circumstance, I believe BSTs to be the better general purpose data structure due to their ease of implementation.