CptS 223 Homework #3 - Heaps, Hashing, Sorting

Please complete the homework problems on the following page using a separate piece of paper. Note that this is an individual assignment and all work must be your own. Be sure to show your work when appropriate. Please scan the assignment and upload the PDF to Git, but also bring a printed out copy for the grading TAs. We've found that many of the scans are difficult to read and notate.

1. [6] Starting with an empty hash table with a fixed size of 11, insert the following keys in order into four distinct hash tables (one for each collision mechanism): {12, 9, 1, 0, 42, 98, 70, 3}. You are only required to show the final result of each hash table. In the <u>very likely</u> event that a collision resolution mechanism is unable to successfully resolve, simply record the state of the last successful insert and note that collision resolution failed. For each hashtable type, compute the hash as follows:

hashkey(key) = (key * key + 3) % 11

Separate Chaining (buckets)

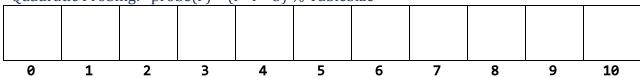


To probe, start at i = hashkey and do i++ if collisions continue

Linear Probing: probe(i') = (i + 1) % TableSize



Quadratic Probing: probe(i') = (i * i + 5) % TableSize



2. [3] For implementing a hash table. Which of these would probably be the best initial table size to pick?

Table Sizes:

1 100 101 15 500

Why?

	you'll need to decide if you need to nto the table, bringing your data count is currently sized at 106963 buckets.
$ullet$ Calculate the load factor (λ):	
Given a linear probing collision	function should we rehash? Why?
 Given a separate chaining collisi 	ion function should we rehash? Why?
4. [4] What is the Big-O of these act	tions for a well designed and properly
loaded hash table with N elements?	
Function	Big-O complexity
Insert(x)	
Rehash()	
Remove(x)	
Contains(x)	

7. [3] I grabbed some code from the Internet for my linear probing based hash table because the Internet's always right. The hash table works, but once I put more than a few thousand entries, the whole thing starts to slow down. Searches, inserts, and contains calls start taking *way* longer than O(1) time and my boss is pissed because it's slowing down the whole application services backend I'm in charge of. I think the bug is in my rehash code, but I'm not sure where. Any ideas why my hash table starts to suck as it grows bigger?

8. [4] Time for some heaping fun! What's the time complexity for these functions in a binary heap of size N?

Function	Big-O complexity
<pre>insert(x)</pre>	
findMin()	
deleteMin()	
<pre>buildHeap(vector<int>{1N})</int></pre>	

9. [4] What would a good application be for a priority queue (a binary heap) Describe it in at least a paragraph of why it's a good choice for you example situation.	
10. [4] For an entry in our heap (root @ index 1) located at position i where are it's parent and children?	ر .
Parent:	
Children:	
What if it's a d-heap?	
Parent:	
Children:	

at a ti	me, int	to an i	nitiall	y empty				5, 15, 1-based			
book do	es. Aft	er inse	rt(10):								
After i	nsert (12):								<u>I</u>	
etc:							l	I		l	
		<u> </u>		Г	Г	Г	T	Т	Г	ı	
							I			I	
12. [4] on the									ing bui	1dHea	ap()
		<u>, </u>				-			.		

4] Now heap:	show	the	result	of thr	ee succ	essive	deleteM:	in oper	ations	from	the

14. [4] What are the average complexities and the stability of these sorting algorithms:

Algorithm	Average complexity	Stable (yes/no)?
Bubble Sort		
Insertion Sort		
Heap sort		
Merge Sort		
Radix sort		
Quicksort		

15. [2] What are the key differences between Mergesort and Quicksort? How does this influence why languages choose one over the other?

16. [4] Draw out how Mergesort would sort this list:

2	24	16	9	10	8	7	20

17. [4] Draw how Quicksort would sort this list:

24	16	9	10	8	7	20