CptS 223 Homework #3 - Heaps, Hashing, Sorting

Due Date: Nov 20th 2020 Jocelyn Strmec

Please complete the homework problems and upload a pdf of the solutions to blackboard assignment and upload the PDF to Git.

1. [6] Starting with an empty hash table with a fixed size of 11, insert the following keys in order into three 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 **very likely** 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)

	3		0	12	1	98	9	42	70	
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

To probe on a collision, start at hashkey(key) and add the current probe(i') offset. If that bucket is full, increment i until you find an empty bucket.

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

	3		0	12	1	98	9			
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

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

	42		0	12	3		9	70	1	98
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

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 did you choose that one?

101 is the best initial size for the table because it is a prime number and therefore the best choice to start the table.

[4] For our running hash table, you'll need to decide if you need to rehash. You just inserted a new item into the table, bringing your data count up to 53491 entries. The table's vector is currently sized at 106963 buckets.

- Calculate the load factor (λ):
  53491entries/106963 buckets=0.5enteries per bucket.

- Given a linear probing collision function should we rehash? Why?
  Yes, a hashtable should rehash when the total number of entries is 75% or less of your total size.

Given a separate chaining collision function should we rehash? Why?
No we shouldn't rehash, it is better to rehash the when load factor = 0.75

[4] What is the Big-O of these actions for a well designed and properly loaded hash table with N elements?

Function	Big-O complexity
Insert(x)	O(N)
Rehash()	O(1)
Remove(x)	O(N)
Contains(x)	O(N)

7. [3] I grabbed some code from the Internet for my linear probing based hash table at work because the Internet's always right (totally!). 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 *much* 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?

```
/**
* Rehashing for linear probing hash table.
*/
void rehash( )
{
    ArrayList<HashItem<T>> oldArray = array;

    array = new ArrayList<HashItem<T>>( 2 * oldArray.size() );

    for( int i = 0; i < array.size(); i++ )
        array.get(i).info = EMPTY;
// Copy old table over to new larger array
    for( int i = 0; i < oldArray.size(); i++ ) {
        if( oldArray.get(i).info == FULL )
        {
            addElement(oldArray.get(i).getKey(),
                    oldArray.get(i).getValue());
        }
    }
}
```

This code slows down because a new array is initialized which takes up more space and memory which will cause a lag in time.

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

Function	Big-O complexity
push(x)	O(log N)
top()	O(1)
pop()	O(log N)
PriorityQueue(Collection<? extends E> c) // BuildHeap	O(N)

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 your example situation.

A good application for a priority queue would be incoming patient priority at a hospital. As patient's come in some injuries, pains, or conditions are more or less severe, as well as the order in which they come. A priority queue is an ideal way to efficiently know which patients to see in which order. If a patient comes in with a possible break, another with a severed limb, and the third with abdominal pain the most critical patient would be the severed limb patient even though the possibly broken limb came in first because it is a more time sensitive need to either reattach the limb or cauterize the wound. Also depending on how many staff are available maybe in this scenario two more ER doctors became available and can take care of the broken leg and the patient with abdominal pain. Because each patient can be assigned a level of critical-ness it is effective to use a priority queue to know when to get to patients.

10. [4] For an entry in our heap (root @ index 1) located at position i, where are it's parent and children?

Parent: $\frac{i+1}{2}$

Children: $2i + 1\ left\ child\ and\ 2i + 2\ right\ child$

11. [6] Show the result of inserting 10, 12, 1, 14, 6, 5, 15, 3, and 11, one at a time, into an initially empty binary heap. Use a 1-based array like the book does. After insert(10):

10										

After insert (12):

12	10									

etc:

12	10	1								

14	12	1	10							

14	12	1	10	6						

14	12	1	10	6	5					

15	12	14	10	6	1	5				

15	12	14	10	6	1	5	3			

15	12	14	11	6	1	5	3	10		

12. [4] Show the same result (only the final result) of calling buildHeap() on the same vector of values: {10, 12, 1, 14, 6, 5, 15, 3, 11}

15	14	10	12	6	5	1	3	11		

13. [4] Now show the result of three successive deleteMin / pop operations from the prior heap:

15	14	10	12	6	5	1	3			11

15	14	10	3	6	5	1				12

15	6	10	3	2	5					14

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

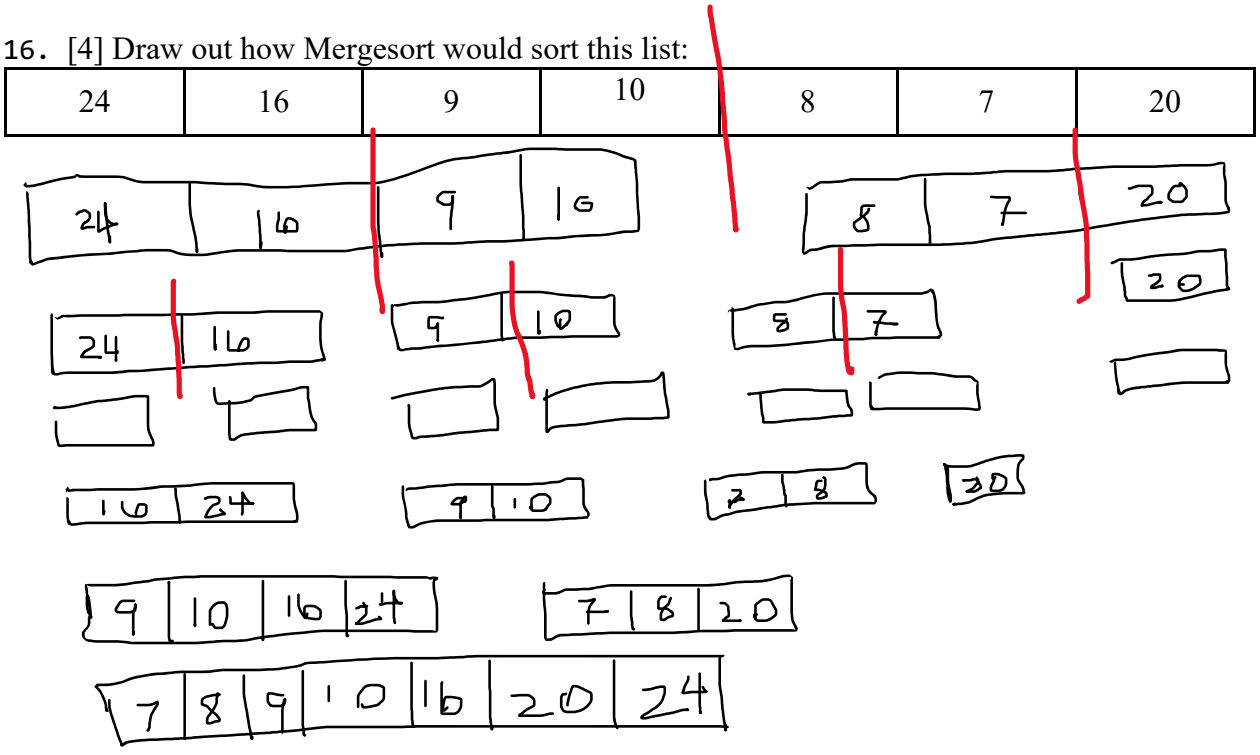
Algorithm	Average complexity	Stable (yes/no)?
Bubble Sort	$O(n^2)$	Yes
Insertion Sort	$O(n^2)$	Yes
Heap Sort	O(n log n)	No
Merge Sort	O(n log n)	Yes
Radix Sort	Θ(nk)	Yes
Quick Sort	O(n log n)	No

15. [3] What are the key differences between Mergesort and Quicksort? How does this influence why languages choose one over the other?

In the merge sort, the array/linkedlist/data type is parted into just 2 halves (i.e. n/2). In quick sort, the array is parted into the length of the array. The worst case complexity of quick sort is O(n^2) as there is need of lot of comparisons in the worst condition. In merge sort, worst case and average case has same complexities O(n log n). Merge sort can work well on any type of data sets irrespective of its, quick sort is slow with large datasets. Quicksort is not stable but uses not extra memory which is why languages chose quicksort over merge sort.

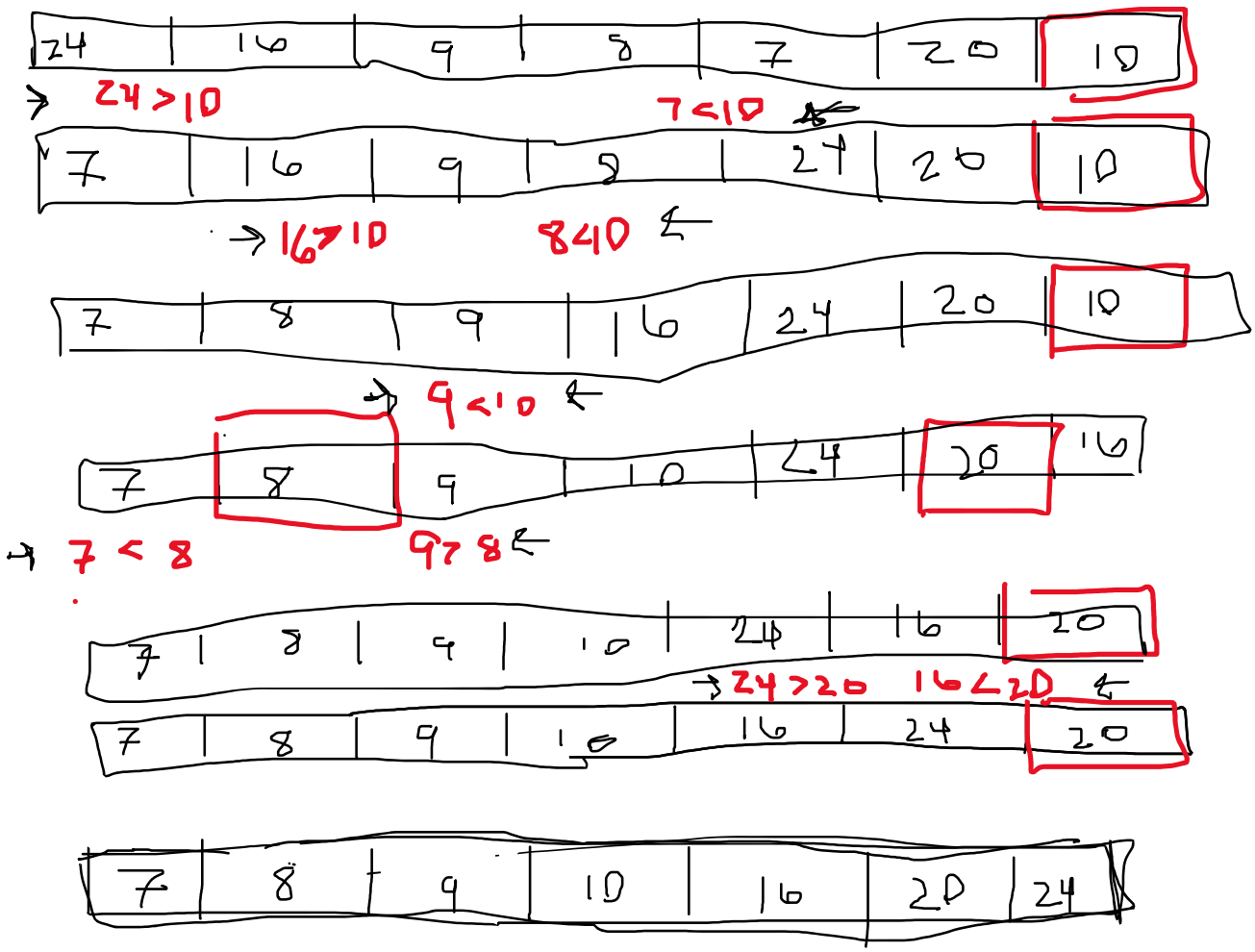
16. [4] Draw out how Mergesort would sort this list:

24	16	9	10	8	7	20



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

24	16	9	10	8	7	20



Let me know what your pivot picking algorithm is (if it's not obvious):

```
private Int PivotChoice (int averageOfArrayValues){
Int min = array[max value];
        For(int I = 0; I < array.length; i++){
        Int temp =(Array[i]-averageOfArrayValues);
        Java.lang.Math.abs(temp);
        If(temp<min){
                Min = temp;
                }
        }
        Return min;
 }
```

Basically, I am just finding the closest value to the average of the array.