Name: Key

Instructions:

1. There are test questions on the front and the back of each sheet.

2. This is a closed book exam. Do not use any notes, books, or neighbors except your one page, two side, handwritten, cheat sheet which MUST be turned in with your exam.

3. Show your work. Partial credit will be given. Grading will be based on correctness and clarity.

4. You have 75 minutes to complete the exam. Watch your time appropriately. You should take about 15-20 minutes per question section.

Integrity: The University of Richmond’s Honor Code is “We, the students of the University of Richmond, shall promote and uphold a community of integrity and trust.” Upon accepting admission to University of Richmond, a student immediately assumes a commitment to uphold the Honor Code, to accept responsibility for learning, and to follow the philosophy and rules of the Honor System. Students will be required to state their commitment on examinations, research papers, and other academic work. Ignorance of the rules does not exclude any member of the Richmond community from the requirements or the processes of the Honor System.

I agree to uphold this commitment and produce original work in this exam, i.e., I will not cheat nor will I consciously let anyone cheat.

Signature: ________________________________

DO NOT BEGIN THE EXAM UNTIL INSTRUCTED TO DO SO. GOOD LUCK!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>received</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Bonus</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
1. (20 points, 2 points each) Answer the following questions.

(a) True or **False**: Sorting with a Priority Queue (PQ-sort) consists of performing an alternating sequence of a single \texttt{insert}(e)s and a single \texttt{removeMin}() operation.

(b) When a PQ is implemented with a sorted sequence, the PQ-sort algorithm is referred to as Insertion Sort (a type of sorting algorithm), and when a PQ is implemented with an unsorted sequence, the PQ-sort algorithm is referred to as Selection Sort (a type of sorting algorithm). Both sorts runs in time $O(n^2)$.

(c) When a PQ is implemented with a(n) array-based heap, the \texttt{insert}(k,v) operation and the \texttt{removeMin}() operations both take $O(\log n)$ time. In this case, the PQ-sort algorithm is referred to as Heap Sort (a type of sorting algorithm) and runs in time $O(n \log n)$.

(d) **True** or False: A map/dictionary implemented as a linked list (e.g., log file) has very good space usage, but might not have very fast operation times.

(e) **True** or False: A map/dictionary implemented as a direct-address table has very fast operation times, but might not have very good space usage.

(f) Assume there are $N$ slots in your hash table, and that there are $n$ data items stored in your hash table. In hashing with chaining, the space usage will be $O(\frac{n+N}{N})$. In open-addressing hashing, the space usage will be $O(\frac{N}{N})$.

(g) Consider a binary search tree $T$ storing $n$ (key,element) pairs. The time for a \texttt{find}(k) operation is $O(n)$ in the worst case and the time for a \texttt{put}(k,v) operation is $O(\log n)$ in the best case when $k$ is unique to $T$.

(h) Consider the various types of balanced binary search trees we learned in class. In all implementations, when storing $n$ (key,element) pairs the height is $O(\log n)$.

(i) Consider a Red-Black tree $T$ storing $n$ (key, value) pairs, the \texttt{put}(k,v) and \texttt{remove}(k) operations require $O(\log n)$ recolors and $O(1)$ restructures in the worst case.

(j) When implementing the Sorted Multi-Map ADT, a red-black tree is the preferred implementation, and when implementing the Set ADT, a Hash map with chaining is the preferred implementation.
2. (10 points) **Hashing.** Consider inserting the following keys, in this order, into a hash table of size \( N = 9 \).

**keys to insert (in this order): 5, 6, 7, 15, 16, 17**

(a) (3 points) Suppose you use chaining with the hash function \( h(k) = 2k \mod N \). Illustrate the result of inserting the keys above using chaining.

(b) (2 points) What is the (i) expected time and the (ii) worst-case running time for a find operation on a hash table of size \( N \) that contains \( n \) items, where collisions are resolved by chaining without rehashing? Clearly state any assumptions.

\[
\begin{align*}
\text{(i)} & \quad O\left(\frac{n}{N}\right) \\
\text{(ii)} & \quad O(n)
\end{align*}
\]

(c) (3 points) Suppose you use open addressing with hash functions \( h_1(k) = 2k \mod N \) and \( h_2(k) = 1 + (k \mod (N - 1)) \). Illustrate the result of inserting the keys above using double hashing, i.e., \( h(k, i) = (h_1(k) + i \cdot h_2(k)) \mod N \).

(d) (2 points) What is the (i) expected time and the (ii) worst-case running time for inserting \( n \) items into an initially empty hash table of size \( N \) when using open addressing with rehashing using a maximum load factor of \( \frac{2}{3} \)? Clearly state any assumptions.

\[
\begin{align*}
\text{(i)} & \quad O(n) \\
\text{(ii)} & \quad O(n^2)
\end{align*}
\]
3. (30 points) **Binary Search Trees (BSTs)**.

(a) (5 points) Draw the BST that would result from inserting the keys (6, 17, 23, 46, 3, 11) (in this order) into an initially empty BST. Show explicitly any intermediate restructuring that is required.

```
    6
   / \
  3   17
   \   /
    11 23
     \  /
      46
```

(b) (5 points) Draw the resulting BST when the item with key 4 is removed from the BST shown below.

```
  2
 / \
4   3
  \ /\n   7 6
     \ /\n      9 5
          \ /
           8
```

(c) (5 points) Draw the AVL tree that would result from inserting the keys (6, 17, 23, 46, 3, 11) (in this order) into an initially empty AVL tree. Show explicitly any intermediate restructuring that is required.

```
    6
   / \
  17 23
   \   /
    3   46
```

(d) (5 points) Draw the resulting AVL when the item with key 3 is removed from the AVL shown below.

```
  4
 / \
3   8
  \ /\n   6 9
      \ /
       5
```

5
(e) (5 points) Draw the Red-Black tree that would result from inserting the keys (6, 17, 23, 46, 3, 11) (in this order) into an initially empty Red-Black tree. Show explicitly any intermediate restructuring or recoloring that is required. Use circles for black nodes, triangles for red nodes, and little squares for external nodes.

![Red-Black tree diagram](image1)

(f) (5 points) Draw the resulting Red-Black tree when the item with key 3 is removed followed by the item with key 2 is removed form the Red-Black tree shown below. Show explicitly any intermediate restructuring or recoloring that is required. Use circles for black nodes, triangles for red nodes, and little squares for external nodes.

![Red-Black tree diagram](image2)
4. (20 points) **Priority Queues.** Show how to implement the Queue ADT using only the Priority Queue ADT and at most one additional data member. Your approach must work for any combination of $k$ \texttt{enqueue}(e), \texttt{dequeue}(), and \texttt{first}() operations. You do not need to justify complexity bounds, simply state the tightest bound.

(a) (5 points) **General approach.** Describe the general approach and define the variables and extra data needed. Loosely explain why your approach will work.

My approach to ensure the FIFO ordering that a queue produces is to store an extra integer variable $t$ that increases each time an element is added to the queue. Upon insertion to the queue, $t$ will be the key for the enqueued element. I assume that integer overflow will not be an issue (however can be addressed in practice). By doing this, this ensures the first item into the queue will be the smallest in the priority queue, and so forth. Let $PQ$ be the underlying priority queue.

(b) (5 points) **Enqueue.** Describe your algorithm for \texttt{enqueue}(e) in pseudocode. What is the time complexity of this operation if the priority queue is implemented using an array-based heap?

**Algorithm** \texttt{enqueue}(e)

\begin{itemize}
  \item [1:] \texttt{PQ.insert}($t,e$)
  \item [2:] $t \leftarrow t + 1$
\end{itemize}

(c) (5 points) **Dequeue.** Describe your algorithm for \texttt{dequeue}() in pseudocode. What is the time complexity of this operation if the priority queue is implemented with a sorted list?

**Algorithm** \texttt{dequeue}()

\begin{itemize}
  \item [1:] \texttt{return PQ.removeMin().getValue()}
\end{itemize}

(d) (5 points) **First.** Describe your algorithm for \texttt{first}() in pseudocode. What is the time complexity of this operation if the priority queue is implemented with an unsorted list?

**Algorithm** \texttt{first}()

\begin{itemize}
  \item [1:] \texttt{return PQ.min().getValue()}
\end{itemize}
5. (20 points) Maps. Maps are used at various stages of compression and decompression algorithms. Often, these algorithms use symbol frequencies in their more complicated routines. Design and analyze an algorithm using the Map ADT to determine symbol frequencies in a large string $S$.

(a) (10 points) Pseudocode. Provide pseudocode and a description of your algorithm.

**Algorithm FREQUENCIES($s$)**

**Input:** String $S$  
**Output:** Frequency map of characters to integers

1. Map of characters to integers $M \leftarrow \emptyset$
2. for all Symbols $s \in S$ do
3. \hspace{1em} if $M.$get($s$) = $\emptyset$ then
4. \hspace{2em} $M.$put($s$, 1)
5. \hspace{1em} else
6. \hspace{2em} $M.$put($s$, $M.$get($s$) + 1)
7. return $M$

Essentially, my map contains characters as the key and counts as their values. For each character of the string, I add it to the map. If it already exists, the count is incremented by 1.

(b) (10 points) Complexity. Determine the complexity of your algorithm with the following implementations. Let $n$ be the number of symbols in the string $S$. You do not need to justify complexity bounds, simply state the tightest bound.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>i. Unsorted list</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>ii. Sorted table</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>iii. Direct address table</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>iv. Hash table with rehashing at a maximum load factor of $\frac{5}{8}$ assuming uniform hashing</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>v. AVL tree</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>
6. **(Bonus 10 points)** You are given two lists of points \( P, Q \) and a constant time function \( \delta \) that takes two points and returns their distance in some metric space. Give an efficient algorithm for finding the \( k \)-closest pairs of points, where one point is in \( P \) and the other in \( Q \). Determine its complexity assuming the best implementation.

**Algorithm** \textsc{closest-pairs}(\( P, Q, k \))

**Input:** List of points \( P, Q \) and integer \( k \)

**Output:** \( k \)-closest pairs of points between \( P \) and \( Q \)

1. Maximum priority queue of distance, pair entries \( PQ \leftarrow \emptyset \)
2. for all \( p \in P \) do
3. \hspace{1em} for all \( q \in Q \) do
4. \hspace{2em} Distance \( d \leftarrow \delta(p, q) \)
5. \hspace{2em} if \( PQ \text{.size()} < k \lor d < PQ \text{.max()}.getValue() \) then
6. \hspace{3em} \( PQ \text{.insert}(d, \{p, q\}) \)
7. \hspace{2em} if \( PQ \text{.size()} > k \) then
8. \hspace{3em} \( PQ \text{.removeMin()} \)
9. List of point pairs \( L \leftarrow \emptyset \)
10. while \( \neg PQ \text{.isEmpty()} \) do
11. \hspace{1em} \( L \text{.addFirst}(PQ \text{.removeMin()}.getValue()) \)
12. return \( L \)

The complexity using an array-based heap is \( O(|P||Q| \log k + k \log k) \). Here, there are \(|P||Q|\) pairs total, and each pair could cause a remove or insert of the priority queue in the worst case. The last component is caused by \( k \) removals from the priority queue.