NAME:_______________________
Student ID:__________

This exam is closed book, notes, computer and cell phone.
Show partial solutions to get partial credit.
Make sure you have answered all parts of a question.
If your solutions are not written legibly, you won’t get full credit.
Clarity and succinctness will be rewarded.

Question 1:___________(out of 10)
Question 2:___________(out of 10)
Question 3:___________(out of 10)
Question 4:___________(out of 10)
Question 5:___________(out of 10)
Question 6:___________(out of 10)
Question 7:___________(out of 10)
Question 8:___________(out of 10)
Question 9:___________(out of 10)
Question 10:___________(out of 10)

Total:________________(out of 100)
1. Short questions:

(a) What is the running time of RadixSort as a function of
- the number of keys \( n \),
- the number of digits \( d \) in each key and
- the number of possible values \( k \) of a digit?

\[ \Theta(d(n + k)) \]

(b) What is recurrence for the worst-case behavior of QuickSort
and what is the worst-case running time?

\[ T(n) = T(n-1) + cn, \]

where \( n \) is the number of keys and \( c > 0 \) is a constant. This case occurs when the pivot is
the largest or the smallest. Expanding out the recurrence we get that

\[ T(n) = c \sum_{i=1}^{n} i = O(n^2). \]

(c) Somebody gives you \( n \) keys and you are to insert them into a Binary Search Tree.
How should you preprocess the keys and why?

The worst-case time for all \( n \) inserts into a BST is \( O(n^2) \). As a preprocessing step, ran-
domly permute the keys. Now the expected running time is alwa ys \( \Theta(n \log n) \).

(d) What is the definition of \( O \)-notation?

\[ f(n) = O(g(n)) \text{ if there are constants } c \text{ and } n_0 \text{ s.t. } \]

\[ \exists n_0, c \geq 0 : f(n) \leq cg(n), \forall n \geq n_0. \]
2. Somebody proposes a sorting algorithm using Red Black Trees:

   Insert the $n$ keys and then print out the keys in inorder.

   Is this a good sorting algorithm?
   Discuss its advantages and disadvantages.
   In particular, discuss
   - its worst-case running time,
   - average case running time, and
   - space requirement
   in relation to HeapSort and QuickSort.

   There are better sorting algorithms in all regards but the proposed sorting algorithm has good worst-case complexity albeit with a high constant.

   Inserting the $n$ keys is iteratively costs worst-case time $\sum_i c \log(i) = \Theta(n \log(n))$ and printing in inorder is $O(n)$.

   So the overall time for the proposed sorting algorithm is $\Theta(n \log n)$ in the worst-case.

   Quicksort is $O(n^2)$ in the worst-case, but $O(n \log n)$ time in the average case with a small constant.

   HeapSort has $O(n \log n)$ worst-case running time (as the proposed algorithm), but with a much much lower constant.

   Also HeapSort uses $O(1)$ additional space and Quicksort $O(\log n)$ additional space. The proposed algorithm requires $O(n)$ additional space.

   Overall the proposed algorithm is not a practical sorting algorithm.
3. Show by induction (on the height of the tree) that the number of nodes in a complete ternary tree of height $h$ is exactly

$$
\frac{3^{h+1} - 1}{2}.
$$

(In such a tree all internal nodes have exactly 3 children.)

Hint: Begin by drawing the complete ternary trees of height 0, 1, 2.

**Base case $h = 0$:** A complete ternary tree of height 0 consists of a single node and $\frac{3^{0+1} - 1}{2} = 1$.

Assume for $h \geq 0$, the complete ternary tree of height $h$ has $\frac{3^{h+1} - 1}{2}$ nodes and let $T$ be the complete ternary tree of height $h + 1$:

**Proof 1:** The root of $T$ has 3 complete ternary subtrees of height $h$ which by the inductive assumption have $\frac{3^{h+1} - 1}{2}$ nodes each. Together with the root, $T$ has

$$
3 \cdot \frac{3^{h+1} - 1}{2} + 1 = \frac{3^{h+2} - 3}{2} + \frac{2}{2} = \frac{3^{h+2} - 1}{2}
$$

nodes.

**Proof 2:** After stripping the bottom layer from $T$, we are left with a tree of height $h$ which (by the inductive assumption) has $\frac{3^{h+1} - 1}{2}$ nodes. The bottom layer has $3^{h+1}$ nodes, for a total of

$$
\frac{3^{h+1} - 1}{2} + 3^{h+1} = \frac{3^{h+1} - 1}{2} + 2 \cdot \frac{3^{h+1}}{2} = \frac{3^{h+1} - 1}{2} = \frac{3^{h+2} - 1}{2}
$$

nodes.

□
4. A simple hash function for collision resolution by chaining is

\[ h(k) = k \mod m, \]

where \( k \) is the key and \( m \) the number of slots/chains.

Give some properties of a good hash function and present an alternate hash function that could be used.

What are convenient choices for \( m \) for your alternate hash function?

A good hash function spreads the keys evenly in the slots and is easy to compute.

Use the “multiplication method” as the alternate hash function:

Multiply the key \( k \) by a number \( A \) in \((0, 1)\). Extract the fractional part of \( kA \). Then multiply by \( m \) and floor the result.

\[ h(k) = \lfloor m(kA \mod 1) \rfloor. \]

Knuth suggests \( A \sim (\sqrt{5} - 1)/2 = .61803\ldots \). The choice of \( m \) is not critical, but powers of 2 are convenient. In particular, this makes it easy to enlarge and decrease the table size.
5. What operation are efficiently implementable on a Binomial Heap? Give a list of operations and the time bound in big O notation.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make-Heap</td>
<td>O(1)</td>
</tr>
<tr>
<td>Insert</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Minimum</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Extract-Min</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Union</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Decrease-Key</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Delete</td>
<td>O(log n)</td>
</tr>
</tbody>
</table>

Decrease-Key and Delete require a pointer to the node.
Minimum can be made $O(1)$, by keeping the minimum value in the header and updating this value after the other $O(log n)$ operations.

Show how to merge the following two BH’s A and B.
Hint: What binary number do A and B correspond to?

![Figure 1: Two binomial heaps, A and B.](image)

<table>
<thead>
<tr>
<th>name</th>
<th>binary</th>
<th>decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1011</td>
<td>11</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>A+B</td>
<td>1110</td>
<td>14</td>
</tr>
</tbody>
</table>

![A union B](image)
6. Show how to sort $n$ integers in the range from 0 to $n^2 - 1$

in $O(n)$ time using RadixSort.

Use 2 digit numbers and base $n$, i.e. $d = 2$ and $k = n$.

The largest number you can express this way is

$$(n - 1, n - 1) = (n - 1)n^1 + (n - 1)n^0 = n^2 - n + n - 1 = n^2 - 1.$$ 

The smallest number is $(0, 0) = 0$.

Converting an integer key in the range $[0..n^2 - 1]$ into its two digit representation is $O(1)$ per key:

- the lowest significant digit is $k \mod n$
- the first digit is $(k - (k \mod n))/n$.

RadixSort costs $O(d(n + k)) = O(n)$ time.
7. What is the binary-search-tree property?
   For any node,
   the keys in the left subtree are $\leq$ node.key $\leq$ the keys in the right subtree.

What is the min-heap property?
As you walk from a node to the root of the tree, the key values are non-increasing.

What is the time to build a BST with $n$ arbitrary keys (worst-case and average case)?

**Worst case $O(n^2)$:** occurs when nodes appear in sorted order.
**Average case $O(n \log n)$ (under assumption that each permutation equally likely).**

What is the time to build a min-heap with $n$ arbitrary keys?

**$O(n)$** time. From the last internal node, according to increasing height, fix the heap in the sub-tree below.

Can the min-heap property be used to print the keys in the tree in sorted order in $O(n)$ time?
Show how or explain the contradiction to a known theorem about sorting.

If this was true then we would have an $O(n)$ comparison sorting algorithm: building the heap requires $O(n)$ comparisons and printing another $O(n)$. This contradicts the $\Omega(n \log n)$ comparison lower bound for sorting.
8. Show that
\[ \sum_{i=1}^{n} \log_2(4i) = \Theta(n \log(n)). \]

We suggest to reason the upper and lower bound separately.

\[
\sum_{i=1}^{n} \log_2(4i) \leq \sum_{i=1}^{n} \log_2(4n) = \sum_{i=1}^{n} (\log_2(n) + 2) = n \log_2(n) + 2n \leq 3n \log_2(n) \quad \text{for } n \geq 4 = O(n \log(n))
\]

\[
\sum_{i=1}^{n} \log_2(4i) > \sum_{i=1}^{n} \log_2(i).
\]

When \( n \) is even, then \( n/2 \) summands are at least \( \log_2(n/2) \).

When \( n \) is odd, then \( \lceil n/2 \rceil \) summands are at least \( \log_2(\lceil n/2 \rceil) \).

Thus in both cases we have a lower bound of
\[
(n/2) \log_2(n/2) = (n/2)(\log(n)) - n/2.
\]

For \( n \geq 4 \), the above is lower bounded by
\[
(n/4) \log_2(n) = \Omega(n \log n).
\]
9. Show the red-black trees that result after successively inserting the keys 41 38 31 12 19 8 into an initially empty red-black tree.

10. Solve the following recurrences using the Master’s Theorem. In each case, reason which case of the Master theorem (given below) applies and why and determine the solution in Θ notation.

\[ T(n) = 2T\left(\frac{n}{2}\right) + n \]  
\[ T(n) = 3T\left(\frac{n}{4}\right) + n^2 \]  
\[ T(n) = 2T\left(\frac{n}{4}\right) + n \]

**Master’s Theorem**
Let \( a \geq 1 \) and \( b > 1 \) be constants, let \( f(n) \) be a function, and let \( T(n) \) be defined on the nonnegative integers by the recurrence

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n). \]

Then \( T(n) \) has the following asymptotic bounds:

(a) If \( f(n) = O(n^{\log_b(a)-\varepsilon}) \) for some constant \( \varepsilon > 0 \), then \( T(n) = \Theta(n^{\log_b(a)}) \).

(b) If \( f(n) = \Theta(n^{\log_b(a)}) \), then \( T(n) = \Theta(n^{\log_b(a)} \log(n)) \).

(c) If \( f(n) = \Omega(n^{\log_b(a)+\varepsilon}) \) for some constant \( \varepsilon > 0 \),
    and if \( af\left(\frac{n}{b}\right) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).

(1) Case (b) with \( a = b = 2 \), \( \log_2(2) = 1 \), \( f(n) = n^1 \).
    Thus \( T(n) = \Theta(n \log n) \).
    BTW, this is the recurrence of MergeSort.

(2) Case (c) with \( a = 3 \) and \( b = 4 \), \( \log_4(3) < 1 \).
    Thus \( f(n) = n^2 = n^{1+1} = n^{\log_4(3)+1} = \Omega(n^{\log_4(3)+\varepsilon}) \), for \( \varepsilon = 1 \).
    Also \( 3(n/4)^2 = (3/16)n^2 \) and \( c = 3/16 < 1 \) fulfills the last condition of case (c).
    We conclude that \( T(n) = \Theta(n^2) \).

(3) Case (c) with \( a = 2 \) and \( b = 4 \), \( \log_4(2) = 1/2 \).
    Thus \( f(n) = n = n^{1/2+1/2} = n^{\log_4(2)+1/2} = \Omega(n^{\log_4(2)+\varepsilon}) \), for \( \varepsilon = 1/2 \).
    Also \( 2(n/4) = (1/2)n \) and \( c = 1/2 < 1 \) fulfills the last condition of case (c).
    We conclude that \( T(n) = \Theta(n) \).