This exam has 10 questions (including question 0) worth a total of 55 points. You have 80 minutes. This exam is preprocessed by a computer, so please write darkly and write your answers inside the designated spaces.

Policies. The exam is closed book, except that you are allowed to use a one page cheatsheet (8.5-by-11 paper, one side, in your own handwriting). No electronic devices are permitted.

Discussing this exam. Discussing the contents of this exam before solutions have been posted is a violation of the Honor Code.

This exam. Do not remove this exam from this room. Write your name, NetID, and the room in which you are taking the exam in the space below. Mark your precept number. Also, write and sign the Honor Code pledge. You may fill in this information now.

Name:

NetID:

Exam room:

Precept: P01 P02 P03 P03A P04 P05 P06

○ ○ ○ ○ ○ ○ ○

“I pledge my honor that I will not violate the Honor Code during this examination.”

______________________________
Signature
0. Initialization. (1 point)

In the space provided on the front of the exam, write your name, NetID, and the room in which you are taking the exam; mark your precept number; and write and sign the Honor Code pledge.

1. Memory and data structures. (4 points)

Suppose that you implement a weighted quick-union data type using the following parent-link representation:

```java
public class WeightedQuickUnion {
    private final int n; // number of elements
    private final Node[] nodes; // array of n nodes

    private static class Node {
        private int size = 1; // subtree size
        private Node parent = null; // parent link
    }

    // construct a weighted quick-union data structure with n elements
    public WeightedQuickUnion(int n) {
        this.n = n;
        this.nodes = new Node[n];
    }

    ...
}
```

Using the 64-bit memory cost model from lecture and the textbook, how much memory does a `WeightedQuickUnion` object use as a function of the number of elements \( n \)? Use tilde notation to simplify your answer.

\[ \sim \boxed{\text{bytes}} \]
2. Five sorting algorithms. (5 points)

The column on the left is the original input of strings to be sorted; the column on the right are the strings in sorted order; the other columns are the contents at some intermediate step during one of the five sorting algorithms listed below. Match each algorithm by writing its number in the box under the corresponding column. Use each number exactly once.

<table>
<thead>
<tr>
<th></th>
<th>prim</th>
<th>heap</th>
<th>flip</th>
<th>edge</th>
<th>flip</th>
<th>miss</th>
<th>edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>left</td>
<td>left</td>
<td>heap</td>
<td>find</td>
<td>left</td>
<td>load</td>
<td>find</td>
</tr>
<tr>
<td>1</td>
<td>load</td>
<td>java</td>
<td>flip</td>
<td>load</td>
<td>loop</td>
<td>flip</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>push</td>
<td>lazy</td>
<td>left</td>
<td>hash</td>
<td>loop</td>
<td>list</td>
<td>hash</td>
</tr>
<tr>
<td>3</td>
<td>sink</td>
<td>node</td>
<td>load</td>
<td>heap</td>
<td>miss</td>
<td>left</td>
<td>heap</td>
</tr>
<tr>
<td>4</td>
<td>time</td>
<td>loop</td>
<td>java</td>
<td>null</td>
<td>lazy</td>
<td>java</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>loop</td>
<td>loop</td>
<td>miss</td>
<td>lazy</td>
<td>prim</td>
<td>heap</td>
<td>lazy</td>
</tr>
<tr>
<td>6</td>
<td>flip</td>
<td>flip</td>
<td>null</td>
<td>left</td>
<td>push</td>
<td>java</td>
<td>left</td>
</tr>
<tr>
<td>7</td>
<td>null</td>
<td>null</td>
<td>prim</td>
<td>list</td>
<td>sink</td>
<td>flip</td>
<td>list</td>
</tr>
<tr>
<td>8</td>
<td>miss</td>
<td>miss</td>
<td>push</td>
<td>load</td>
<td>swim</td>
<td>hash</td>
<td>load</td>
</tr>
<tr>
<td>9</td>
<td>trie</td>
<td>edge</td>
<td>rank</td>
<td>loop</td>
<td>time</td>
<td>edge</td>
<td>loop</td>
</tr>
<tr>
<td>10</td>
<td>swim</td>
<td>find</td>
<td>sink</td>
<td>miss</td>
<td>trie</td>
<td>find</td>
<td>miss</td>
</tr>
<tr>
<td>11</td>
<td>java</td>
<td>sort</td>
<td>time</td>
<td>find</td>
<td>node</td>
<td>node</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>sort</td>
<td>list</td>
<td>swim</td>
<td>sort</td>
<td>heap</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>13</td>
<td>rank</td>
<td>prim</td>
<td>time</td>
<td>rank</td>
<td>java</td>
<td>prim</td>
<td>prim</td>
</tr>
<tr>
<td>14</td>
<td>heap</td>
<td>rank</td>
<td>trie</td>
<td>sink</td>
<td>list</td>
<td>push</td>
<td>push</td>
</tr>
<tr>
<td>15</td>
<td>list</td>
<td>sort</td>
<td>list</td>
<td>null</td>
<td>rank</td>
<td>rank</td>
<td>rank</td>
</tr>
<tr>
<td>16</td>
<td>find</td>
<td>swim</td>
<td>find</td>
<td>swim</td>
<td>sort</td>
<td>sink</td>
<td>sink</td>
</tr>
<tr>
<td>17</td>
<td>tree</td>
<td>tree</td>
<td>tree</td>
<td>tree</td>
<td>tree</td>
<td>sort</td>
<td>sort</td>
</tr>
<tr>
<td>18</td>
<td>edge</td>
<td>trie</td>
<td>edge</td>
<td>prim</td>
<td>edge</td>
<td>swim</td>
<td>swim</td>
</tr>
<tr>
<td>19</td>
<td>hash</td>
<td>time</td>
<td>hash</td>
<td>push</td>
<td>hash</td>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td>20</td>
<td>node</td>
<td>sink</td>
<td>node</td>
<td>node</td>
<td>node</td>
<td>tree</td>
<td>tree</td>
</tr>
<tr>
<td>21</td>
<td>lazy</td>
<td>push</td>
<td>lazy</td>
<td>trie</td>
<td>lazy</td>
<td>trie</td>
<td>trie</td>
</tr>
<tr>
<td>22</td>
<td>type</td>
<td>type</td>
<td>type</td>
<td>type</td>
<td>type</td>
<td>type</td>
<td>type</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

(0) Original input  (3) Mergesort  (5) Quicksort
(1) Selection sort   (top-down)           (standard, no shuffle)
(2) Insertion sort  (4) Heapsort          (6) Sorted
3. **Analysis of algorithms. (6 points)**

Consider an array that contains two successive copies of the integers 1 through \(n\), in ascending order. For example, here is the array when \(n = 8\):

\[
1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8
\]

Note that the length of the array is \(2n\), not \(n\).

(a) How many compares does *selection sort* make to sort the array as a function of \(n\)?
Use tilde notation to simplify your answer.

\[
\sim \ \text{compares}
\]

(b) How many compares does *insertion sort* make to sort the array as a function of \(n\)?
Use tilde notation to simplify your answer.

\[
\sim \ \text{compares}
\]

(c) How many compares does *mergesort* make to sort the array as a function of \(n\)?
Assume \(n\) is a power of 2. Use tilde notation to simplify your answer.

\[
\sim \ \text{compares}
\]
4. Red–black BSTs. (6 points)

Suppose that you insert the key 31 into the following left-leaning red–black BST:

Which of the following color flips and rotations result? Mark all that apply.

**Color flips:**

- [ ] 18
- [ ] 22
- [ ] 26
- [ ] 28
- [ ] 30
- [ ] 31
- [ ] 32

**Rotations:**

- [ ] 18 left
- [ ] 18 right
- [ ] 28 left
- [ ] 28 right
- [ ] 30 left
- [ ] 30 right
- [ ] 32 left
- [ ] 32 right
- [ ] 33 left
- [ ] 33 right

**Examples of color flips and rotations (for reference):**

- [ ] Color flip 3
- [ ] Rotate 8 right
- [ ] Rotate 3 left
5. **Hash tables. (5 points)**

Suppose that the following keys are inserted into an initially empty linear-probing hash table, but not necessarily in the order given,

<table>
<thead>
<tr>
<th>key</th>
<th>hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
</tr>
<tr>
<td>I</td>
<td>3</td>
</tr>
<tr>
<td>N</td>
<td>5</td>
</tr>
<tr>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>5</td>
</tr>
<tr>
<td>R</td>
<td>4</td>
</tr>
</tbody>
</table>

and it results in the following hash table:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>R</td>
<td>O</td>
<td>B</td>
<td>I</td>
<td>N</td>
<td>G</td>
</tr>
</tbody>
</table>

For each description at left, mark all keys that apply. Assume that the initial size of the hash table is 7 and that it neither grows nor shrinks.

- **could have been first key inserted**
  
<table>
<thead>
<tr>
<th>B</th>
<th>G</th>
<th>N</th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **could have been last key inserted**
  
<table>
<thead>
<tr>
<th>B</th>
<th>G</th>
<th>N</th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **must have been inserted before I**
  
<table>
<thead>
<tr>
<th>B</th>
<th>G</th>
<th>N</th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **must have been inserted after I**
  
<table>
<thead>
<tr>
<th>B</th>
<th>G</th>
<th>N</th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. Programming assignments. (6 points)

Answer the following questions about the COS 226 programming assignments.

(a) Consider implementing Percolation using a union–find implementation that supports union and find in $\sqrt{m}$ time per operation on a set of $m$ elements. What is the order of growth of the running time to perform one percolation experiment (open random sites until the system percolates) on an $n$-by-$n$ system? Mark the best answer.

\[
\begin{array}{cccccccc}
 n^{1/2} & n & n^{3/2} & n^2 & n^{5/2} & n^3 & n^{7/2} & n^4 \\
\bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc
\end{array}
\]

(b) Consider implementing a Deque using a singly linked list, storing the first item in the deque in the first node in the linked list (and the last item in the deque in the last node). Which of the following operations could be implemented to run in constant time in the worst case? Mark all that apply.

- addFirst()
- addLast()
- removeFirst()
- removeLast()
- isEmpty()

\[
\begin{array}{cccccc}
 & & & & & \\
\bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc
\end{array}
\]

(c) Consider nearest-neighbor search in a 2d tree with $n \geq 1$ points. Which of the following are features of the algorithm and pruning rule specified in the assignment? Mark all that apply.

- Guarantees to return a nearest neighbor.
- Guarantees log $n$ time per operation in the worst case.
- Guarantees log $n$ time per operation in the worst case if the 2d tree is balanced.
- Achieves log $n$ time per operation on inputs likely to arise in practice.
7. Data structure and algorithm properties. (6 points)

Match each quantity on the left by writing the letter of the best matching term at right. You may use each letter more than once or not at all. Assume each algorithm is the standard version, presented in the textbook.

- Maximum number of key compares to binary search for a key in a sorted array.

  - A. constant
  - B. $\sim \frac{1}{2} \log_2 n$

- Maximum number of key compares to perform a DELETE-MAX operation in a binary heap containing $n$ keys.

  - C. $\sim \log_3 n$
  - D. $\sim \log_e n$
  - E. $\sim \log_2 n$
  - F. $\sim 2 \log_e n$
  - G. $\sim 2 \log_2 n$

- Minimum height of a weighted quick-union tree with $n$ elements.

  - H. $\sim 4.311 \log_e n$

- Minimum height of a binary search tree with $n$ keys.

  - I. linear

- Minimum number of black links on path from the root to a null link in a left-leaning red-black BST containing $n$ keys.

- Maximum number of times an array can be resized (doubled or halved) during a sequence of $n$ push and pop operations (starting from an empty data structure) in a resizing-array implementation of a stack.
8. Largest bandwidth. (8 points)

Given \( n \) time intervals \((l_i, r_i)\) and associated bandwidths \( b_i > 0 \), the bandwidth demand at time \( t \) is the sum of the bandwidths of all intervals that contain \( t \). Design an algorithm to find a time \( t^* \) that has the largest bandwidth demand.

In the example below, there are \( n = 7 \) time intervals. The largest bandwidth demand is 29 and occurs at time \( t^* = 14.5 \): the bandwidth of the interval \((12,15)\) is 10 and the bandwidth of the interval \((14,20)\) is 19.

Give a crisp and concise English description of your algorithm in the space below.

*Your answer will be graded for correctness, efficiency, and clarity. For full credit, the worst-case running time must be proportional to \( n \log n \). You may assume that all of the interval endpoints are distinct.*
9. **Data structure design. (8 points)**

Create a data type `UniQueue` that implements a FIFO queue of strings with *no duplicates*, according to the following API:

```java
public class UniQueue

    UniQueue() create an empty uni-queue
    void enqueue(String s) add the string to the uni-queue
        (if it is not already in the uni-queue)
    String dequeue() remove and return the string least recently added to the uni-queue
    boolean isEmpty() is the uni-queue empty?

```

That is, if a duplicate key is added to the uni-queue, it is ignored. Here is an example:

```java
UniQueue queue = new UniQueue();
queue.enqueue("ant"); // [ "ant" ]
queue.enqueue("bear"); // [ "ant", "bear" ]
queue.enqueue("cat"); // [ "ant", "bear", "cat" ]
queue.enqueue("bear"); // [ "ant", "bear", "cat" ]
queue.enqueue("dog"); // [ "ant", "bear", "cat", "dog" ]
queue.dequeue(); // [ "bear", "cat", "dog" ]
queue.dequeue(); // [ "cat", "dog" ]
queue.enqueue("bear"); // [ "cat", "dog", "bear" ]
```

*Your answer will be graded for correctness, efficiency, and clarity.*

(a) Declare the instance variables for your `UniQueue` data type. You may declare nested classes or use any of the data types that we have considered in this course.

```java
public class UniQueue {

}
```
(b) Briefly describe how to implement `enqueue()` and `dequeue()`, using either crisp and concise prose or code.

```java
public void enqueue(String s) {
    // Implementation
}

public String dequeue() {
    // Implementation
}
```

*Assume that the argument to `enqueue()` is not null.*

*Do not worry about underflow.*

(c) What is the order of growth of each operation as a function of the number of items $n$ in the data structure? Mark whether it is an amortized bound and/or makes the uniform hashing assumption.

<table>
<thead>
<tr>
<th>operation</th>
<th>order of growth</th>
<th>amortized bound</th>
<th>uniform hashing assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>enqueue()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>dequeue()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>