1. Priority queue basics
   5 points

Suppose that the sequence J A V A * I S * * * F * U * N * * * R I G H T * is applied to an initially empty priority queue (where a letter means insert and an asterisk means remove the maximum). Give the sequence of letters returned by the remove the maximum operators. Show intermediate steps as well as the final string that is output.

2. Priority queues generalize other collections
   10 points

Implement a stack, a vanilla (first-in-first-out) queue, and a random queue (where you remove a random element) using a priority queue. Use the Priority Queue API for a MaxPQ given in the lecture and the textbook and show how you would construct keys that would give the behavior of a stack, a FIFO queue and a random queue. You should explain your solution in English without giving complete Java code.

3. Representing binary trees with arrays
   5 points

For the array given below:

1. Draw a binary tree representation of the array.

2. Say whether or not is in heap order. If it is not in heap order, what changes would you make to put it in heap order?

   0 1 2 3 4 5 6 7 8 9 10 11
   S R Q P D P E B A C H

4. Counting keys in heaps
   20 points

Consider a max-heap $T$ for sorting $n$ keys. Give an efficient algorithm for reporting all the keys in $T$ that are greater than or equal to a given query key $x$ (which is not necessarily in $T$). Your algorithm should run in $O(k)$ time, where $k$ is the number of keys reported.
5 Heapsort trace  
15 points
Give a trace of the behavior of heapsort. Fill out a table giving the values for \( N \), \( k \) and the array using the specified input. Gray out the elements that are not touched during each step. **Bold** the item that is being moved in a sink operation. In your answer, say which implementation of heapsort you are using, either by reproducing the code or pseudocode for the algorithm or referencing the algorithm number in the textbook.

<table>
<thead>
<tr>
<th>( N )</th>
<th>( k )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial values</td>
<td></td>
<td>D</td>
<td>A</td>
<td>V</td>
<td>I</td>
<td>N</td>
<td>C</td>
<td>I</td>
<td>C</td>
<td>O</td>
<td>D</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sorted result</td>
<td></td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>E</td>
<td>I</td>
<td>I</td>
<td>N</td>
<td>O</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

Answer these questions about your Heapsort trace:

- Mark the line where the array is heap ordered or write out the array values in order.
- How many steps did it take in your trace to heap order the array?
- How many steps does it take in general to heap order an array? Why?

6 Constructing a heap  
20 points
Prove or disprove: Constructing a heap takes linear time.

7 Binary search tree (BST)  
10 points
Draw the BST that results when you insert the keys \( L \ E \ X \ I \ C \ O \ G \ R \ A \ P \ H \ Y \), in that order (associating the value \( i \) with the \( i \)th key, using the convention from the textbook) into an initially empty tree. How many compares are needed to build the tree?

8 Heaps versus binary search trees  
15 points
1. Describe the difference between a heap and a BST.
2. When might you use one over the other?
3. Consider the binary tree representation of heaps. Will all sequences of insertions produce the same heap? What sequence of insertions could you insert to incur the worst case runtime? Best case?

9 Extra credit: Block the box  
10 points
You are given a collection of \( n \) full and empty boxes \( B[1 \ldots n] \) which are arranged along a line. The entry \( B[i] \) is a 0 if the \( i \)th box is empty and is 1 otherwise, for \( 1 \leq i \leq n \). Your goal is to block all the full boxes
while minimizing the number of empty boxes that are blocked in the process. The way to block a box is to cover it with a board. Moreover, a single board can be used to block two (or more) adjacent boxes. However, you have a limited budget and hence can only afford to buy \( k \) boards (but their lengths can be arbitrarily long). Design an \( O(n + k \lg n) \) time algorithm that given the boxes \( B \) and a budget \( k \), outputs the minimum number of the empty boxes that have to be blocked in order to block all the full boxes. For simplicity, you may assume the first and last boxes are both full. You should not assume that \( k \) is a constant.

Example: The following example illustrates the desired input-output behavior.

Input: \( B = \{1, 0, 0, 1, 0, 1, 0, 0, 1\} \), \( k = 3 \)

Output: 1, that is, you can block all full boxes using 3 boards while blocking only one empty box. This can be achieved by covering the sequences of blocks \( B[1] \), \( B[4, ... , 7] \), and \( B[10] \) each with a single board; only the empty box \( B[6] \) is covered in this manner.

10 Extra credit: Merging sorted lists
10 points

Using data structures from this part of the class, give a \( O(n \lg k) \)-time algorithm to merge \( k \) sorted lists into one sorted lists, where \( n \) is the total number of elements in all the input lists.

11 Extra credit: Keys and Values
10 points

Max, an avid 121 student, is in a fierce competition with his best friend Lewis. Max claims that if he has a set \( S \) of \( n \) items, where each item has a key and a value, he can make a binary tree where each node corresponds to an item in \( S \), and is sorted with respect to the keys. Lewis argues that he can do this too, but additionally he can have the same tree also maintain the min-heap property with respect to the values \( V \). If we assume that all keys and values are distinct, is Lewis right? That is, can we make the tree that is a BST with respect to the keys and yet is a min-heap with respect to the values?