In any of the problems below, you may need not explain any of the standard algorithms or data structures discussed in class. For example, if you wish to use a 2-3 tree for some problem, you may simply say "Use a 2-3 tree with the standard ADD and DELETE operations." You do not have to describe the details of the data structure or operation unless the problem specifically asks for it.

**Problem 1:** (5 points)
If you are sorting a million items, roughly how much faster is heapsort than insertion sort? (Note: \( \log(1,000,000) = 20. \))

**Answer:**
\[
\frac{n^2}{n \log_2(n)} = \frac{n}{\log_2(n)} = 1,000,000/20 = 50,000 \text{ times faster.}
\]

**Problem 2:** (10 points)
Given a list of integers, you wish to find the mode; that is, the value that appears most often in the list. Let \( n \) be the length of the list, and let \( k \) be the number of different values in the list. For instance, in the list \( \langle 1, 5, 2, 5, 2, 5, 2 \rangle, \) \( n = 9, \) \( k = 3, \) and the mode is 5, which appears four times.

Assume that the values are all between 1 and \( M, \) and that you have enough memory to construct an array of size \( M. \) Given an algorithm to find the mode in time \( O(n). \)

**Answer:**
This assumes that array Count[1 .. M] is initialized to 0.

```plaintext
Mode = 0;
ModeCount=0;
for (I in L) {
    Count[I] = Count[I]+1;
    if (Count[I] > ModeCount) {
        Mode=I;
        ModeCount = Count[I];
    }
}
```

**Problem 3:** (20 points)

A. Give a trace of Dijkstra’s single-source shortest path algorithm running on the graph below from source vertex A. You should show the successive states of the array that holds the current estimate of the shortest path. Do not worry about path recovery.

**Answer:**

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Array</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C D E F G H I J</td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>0 15 I I I 30 2 I I I</td>
<td>{A}</td>
</tr>
<tr>
<td>2.</td>
<td>0 12 9 I I 22 2 10 I I</td>
<td>{A,G}</td>
</tr>
<tr>
<td>3.</td>
<td>0 12 9 19 18 22 2 10 I I</td>
<td>{A,G,C}</td>
</tr>
<tr>
<td>4.</td>
<td>0 12 9 16 18 22 2 10 18 I</td>
<td>{A,G,C,H}</td>
</tr>
<tr>
<td>5.</td>
<td>0 12 9 16 18 22 2 10 18 I</td>
<td>{A,G,C,H,B}</td>
</tr>
<tr>
<td>6.</td>
<td>0 12 9 16 18 22 2 10 18 18</td>
<td>{A,G,C,H,B,D}</td>
</tr>
</tbody>
</table>

E,I,J,F are added in further iterations, but there is no further change.

B. Show the sequence in which Kruskal’s algorithm adds edges to the minimum spanning tree in the graph below. You need not show the union-find sets.

Problem 4: (10 points)
Show how the disjoint tree implementation of Union-Find sets with merge by rank and path-compression, can be modified to support the following three operations with specified running times. Be sure to explain how the UNION operation is affected by the need to maintain this additional information. The UNION operation should still run in time $\Theta(1)$. It should not be necessary to modify the FIND operation.

- $\text{MIN}(r)$ — Find the smallest element in the set with root $r$. Time = $\Theta(1)$.
- $\text{MAX}(r)$ — Find the largest element in the set with root $r$. Time = $\Theta(1)$.
- $\text{ENUMERATE}(r)$ — List the elements in the set with root $r$. Time = $\Theta(|S|)$.

Answer: At the root of each tree, keep a record of the min, max, and a linked list of the elements with pointers to front and end. When two trees are merged, set the min to be the lesser of the two mins, the max to be the greater of the two maxes, and splice the two lists.

Problem 5: (20 points) Describe an implementation of an ADT for sets of ordered elements that supports all of the following operations in worst-case time $\Theta(\log n)$. (The same data structure should support all three operations simultaneously.)

- $\text{ADD}(x,S)$ — Add element $x$ to set $S$.
- $\text{DELETE}(x,S)$ — Delete element $x$ from set $S$.
- $\text{SUM-BETWEEN}(x,y,S)$ — Find the sum of the values in $S$ that lie between $x$ and $y$ inclusive. For example, if $S$ currently has the value \{2, 3, 5, 7, 11, 13, 17, 19\}, then $\text{SUM-BETWEEN}(5,14,S)$ should return 36 (= 5 + 7 + 11 + 13).

For this problem, you should use a modification of a standard ADT. In your answer, it suffices to sketch the changes that you would make to the data structure and the operations; you do not have to give a complete account of the algorithm.

Answer: Use a 2-3 tree supplemented by labelling each node with the sum of the elements in the subtree.

ADD and DELETE are the usual 2-3 tree operations, except that for every node that is modified, you recompute the total at the node and at all its ancestors as the sum of the total of the children. Since only $\log n$ nodes are involved, this is additional work of $O(\log n)$.

To compute $\text{SUM-BETWEEN}(x,y,S)$:

Find the path from the root to $x$;
Find the path from the root to $y$;
Let R be the lowest common ancestor of x and y;
SUM = x+y;
If (R has three children and the path to x goes through the first child
and the path to y goes through the third child)
SUM = SUM plus the total on the second child;
For (each node P on the path from R to x, non-inclusive)
P1 = the child of P on the path to x (possibly x itself)
SUM = SUM + the sum of the totals on all children of P to the right of P1;
endfor
For (each node P on the path from R to y, non-inclusive)
P1 = the child of P on the path to y (possibly y itself)
SUM = SUM + the sum of the totals on all children of P to the left of P1;
endfor
return SUM

Problem 6: (20 points)
Construct an algorithm with the following properties:

A. The input is an integer k and an array A of integers of arbitrary size. The length of A is at least 2k. When the algorithm is complete, the k smallest values in A are in order in the first k places of A and the k largest values in A are in order in the last k places of A. For instance, if k = 2 and A on input is the array [69, 2, 300, 10, 91, 4, 106, 289], then an acceptable output would be [2, 4, 91, 10, 69, 289, 300].

B. The algorithm runs in place; that is, it requires only constant storage beyond A.

C. The worst case running time is Θ(N + K log N) where N is the length of A.

(Note: If you cannot find an algorithm with the desired running time, some partial credit will be given for an algorithm with longer running time. In that case, you should specify the running time of your algorithm.)

Step 1: Construct a max-heap with the root at A[1].
Step 2: Do K DELETE-MAX’s, putting the result backward in A[N] ... A[N+1-K].
Step 3: Using only the array from A[1] up to A[N-K], construct a backward
min-heap with root at A[N-K].
Step 4: Do K DELETE-MIN’s, putting the results in A[1] ... A[K].

Steps 1 and 3 take time O(N); steps 2 and 4 take time O(K log N).

Problem 7: 15 points
Let G be a directed graph where each edge U → V has a cost C[U, V]. Modify the Floyd-Warshall algorithm so that it returns two matrices: A[U, V] is the cost of the shortest path from U to V and M[U, V] is the number of different paths from U to V that have the minimal cost. For instance, in the graph below, there are seven different paths of cost 8 from A to G:

A---->B---->D---->E---->G
A---->B---->D---->F---->G
A---->C---->D---->E---->G
A---->C---->D---->F---->G
A--->D--->E--->G
A--->D--->F--->G
A--->G

Hint: When you are comparing the shortest paths from $I$ to $J$ through $K$ with the shortest paths already found from $I$ to $J$, you have to consider three cases:

A. The old paths are shorter.
B. The new paths are shorter.
C. The paths are the same length.

**Answer:** In case (A), $M[I, J]$ is unchanged. In case (B) we switch to the paths $I \rightarrow K \rightarrow J$. The number of these is $M[I, K] \cdot M[K, J]$. In case (C), we combine both sets of paths. In that case we have a total of $M[I, J] + M[I, K] \cdot M[K, J]$ shortest paths. So the algorithm is as follows

```plaintext
for (I=1 to N)
    for (J=1 to N) {
        A[I,J]=Cost[I,J];
        M[I,J]= 1;
    }
for (K=1 to N)
    for (I=1 to N)
        for (J=1 to N) {
            if (NewShort < A[I,J]) {
                A[I,J] = NewShort;
            }
            elseif (NewShort == A[I,J])
        }
```