1. (15 points) Stable sorting and in-place sorting.
Indicate whether the following sorting algorithms are stable and/or in-place. Also, provide their time complexity (with justification if needed).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Stable?</th>
<th>In-place?</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quick sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insertion sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Counting sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radix sort</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. (15 points) Counting sort.
Suppose that the for loop header in line 7 of the Counting Sort algorithm (lecture13.ppt Slide #12) is rewritten to

\[
\text{for } j \leftarrow 1 \text{ to } n
\]

Will the algorithm still sort correctly? Is the modified algorithm a stable sorting algorithm? If yes, prove it. Otherwise provide a counterexample.

3. (15 points) Sorting in place in linear time.
Suppose that we have an array of \( n \) data records to sort, and that the key of each record has the value 0 or 1. An algorithm for sorting such a set of records might possess some subset of the following three desirable characteristics:

1. The algorithm runs in \( \Theta(n) \) time.
2. The algorithm is stable.
3. The algorithm sorts in place, using no more than a constant amount of storage space in addition to the original array.

Answer the following questions and briefly justify your answer.

a. Give an algorithm that satisfies criteria 1 and 2 above.
b. Give an algorithm that satisfies criteria 1 and 3 above.

c. Give an algorithm that satisfies criteria 2 and 3 above.

4. **(10 points) Algorithm design.**
   Describe an algorithm that can sort $n$ integers in the range 0 to $n^2 - 1$ in $O(n)$ time.

5. **(15 points) Order Statistics.**
   Study the pseudocode and example of Rand-Select on Slides #5-8 in Lecture14.ppt. Use Slide #8 as a model, illustrate the operation of selecting the 4th smallest element on array $A = [6 \ 3 \ 16 \ 11 \ 7 \ 17 \ 14 \ 8]$. Similar to Slide #8, you can use ordinary Partition rather than Rand-Partition, i.e., you always select the first element as the pivot.
6. **(15 points) Largest $i$ numbers in sorted order.**

Given a set of $n$ numbers, we wish to find the $i$ largest in sorted order using a comparison-based algorithm. Analyze the running time of the following three methods in terms of $n$ and $i$, and compare their efficiency.

a. Sort the numbers and list the $i$ largest.

b. Build a max-priority queue from the numbers, and call Extract-Max $i$ times.

c. Use an order-statistic algorithm to find the $i$th largest number, partition around that number, and sort the $i$ largest numbers.

7. **(15 points) Extra credit.**

In the worst-case linear-time SELECT algorithm, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? Argue that the algorithm does not run in linear time if groups of 3 are used.