Algorithms

Assignment: Sorting

Name: ...........................................................................................................

Id: .............................................................................................................

<table>
<thead>
<tr>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Good Luck!
1. Let $A[1], A[2], \ldots, A[n]$ be an array containing $n$ very large positive integers.

Describe an efficient algorithm to find the minimum positive difference between any two integers in the array.

What is the complexity of your algorithm? Explain.
2. Let $A$ be an array containing $n$ very large positive integers not necessarily distinct.

**Problem $\mathcal{P}$:**

Find the integer that appears the most times in the array $A$ and the number of times it appears in the array. If there are several such integers it is enough to find one such integer.

Describe an $O(n \log n)$-time algorithm that solves problem $\mathcal{P}$ with a constant-size ($O(1)$) additional memory. Explain why your algorithm has time complexity $O(n \log n)$. 
3. Design an efficient algorithm to sort 5 distinct keys.
4. Let \( A = A[1], \ldots, A[n] \) be an array of \( n \) distinct positive integers (the value of these integers could be very very large). An inversion is a pair of indices \( i \) and \( j \) such that \( i < j \) but \( A[i] > A[j] \).

For example in the array \([30000, 80000, 20000, 40000, 10000]\), the pair \( i = 1 \) and \( j = 3 \) is an inversion because \( A[1] = 30000 \) is greater than \( A[3] = 20000 \). On the other hand, the pair \( i = 1 \) and \( j = 2 \) is not an inversion because \( A[1] = 30000 \) is smaller than \( A[2] = 80000 \). In this array there are 7 inversions and 3 non-inversions.

Describe an efficient algorithm that counts the number of inversions in any array. What is the running time of your algorithm?
An array $A$ of $n \geq 1$ distinct positive integers is bitonic if there exists an index $1 \leq i \leq n$ such that:


Describe an algorithm that uses a linear number ($O(n)$) of comparisons to sort bitonic arrays with very large integers (you cannot use radix-sort like algorithms). Note that if the array is bitonic the index $i$ is not known. Justify your answer.