**MID-TERM**

**Remarks:** Solve 4 of the following 5 questions. In all algorithm, always explain how and why they work. ALWAYS, analyze the complexity of your algorithms. In all algorithms, always try to get the fastest possible. A correct algorithm with slow running time may not get full credit.

1. **Question 1:** Let $A$ be an array of pairwise different numbers. The array is unsorted. Consider $A[i]$. We say that $A[i]$ has distance $\text{dist}(A[i])$ in the array if its sorted place (the place that $A[i]$ will be in if we sort the array) is $j$ and $k = |i - j|$. In other words, the distance of $A[i]$ is the number of entries $A[i]$ is located from its “correct” place. Define the distance of $A$ to be

$$\text{dist}(A) = \sum_{i=1}^{n} \text{dist}(A[i]).$$

- Show that $\text{dist}(A) = O(n^2)$.
- Show that there are arrays $A$ so that $\text{dist}(A) = \Omega(n^2)$, namely, $\text{dist}(A) \geq c \cdot n^2$ for some (maybe smaller than 1) constant $c$.
- A sorting algorithm is called *local* if it only exchanges adjacent elements; That is, every swapping of the algorithm is such that for some $i$, it exchanges $A[i]$ and $A[i + 1]$ (so that $A[i + 1] < A[i]$).
  
  Show that any local sorting algorithm has $\Omega(n^2)$ running time.

2. **Question 2:** We are given an array $A[1..n]$ of numbers, and say that the number of different values that appear in $A$ is at most 10. Give an algorithm to sort $A$.

3. **Question 3:** We are given an array $A$. A contiguous increasing sequence is a subarray $A[i], A[i + 1], \ldots, A[j]$ so that $A[i] \leq A[i + 1] \leq \ldots \leq A[j - 1] \leq A[j]$. The size of the above contiguous sequence is $j - i + 1$.

  Give an algorithm that finds the largest contiguous subarray.

4. **Question 4:**

  A non contiguous subarray of $A$ is an sequence $A[i], A[j], A[k], \ldots$ so that $j > i$, $k > j$, etc. You take only part of the elements, but keep the order. Give an algorithm that computes the largest increasing not necessarily contiguous sequence.

5. **Question 5:** Solve the Knapsack problem under the assumption that there are only two different price values in the input (but the volumes of items are arbitrary).