Theoretical exercise II

Remarks: 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. In all data structures, try to minimize as much as possible the running time of any operation.

1. Question 1:
   (a) The distance of an element $A[i]$ in an unsorted array is $|i - j|$ with $j$ the place of $A[i]$ in the sorted array. For example, in the array 6, 4, 2, 3, 8 the distance of 8 is 0 (8 is in its proper place), the distance of 6 is 3 (6 is now first in the order but should be forth), the distance of 4 is 1, the distance of 2 is 2 and the distance of 3 is 2.
   Show an input where the sum of distances is $\Omega(n^2)$
   (b) Suppose a sorting algorithm in each stage swaps only adjacent elements. That is, (like for example Bubble-Sort and Simple Insertion-Sort) in each stage it swaps $A[i]$ and $A[i+1]$ for some $i$. Let $d_i$ denote the the distance of $A[i]$. Show that $A[i]$ must go through $d_i$ comparisons in any algorithm of this type.
   (c) Show that any algorithm of this type performs at least $\sum_i d_i/2$ comparisons
   (d) Again suppose a sorting algorithm in each stage swaps only adjacent elements. Show that any such algorithm has running time $\Omega(n^2)$ in the worst case. The bad input is always the same. Which is it?

2. Question 2: Let $A$ and $B$ be two non-sorted arrays. Assume that all the numbers in $A$ are pairwise disjoint and the same is true for $B$. Write an efficient as you can algorithm that lists all the elements (values) of $B$ that that do not appear in $A$.

3. Question 3: Say that we want to maintain two Heaps one with the minimum at the top (and the children no smaller than the parent) and one with the maximum on the top (and the children no larger than the parent).
   We want to be able to handle the operations $Delete-Max(S), Delete-Min(S)$ and $Insert(S)$. Use two heaps with pointers between them to be able to employ these operations in time $O(\log n)$ each.
   Note: when you delete the minimum for example, then you have to delete the copy of this element in the Heap for Maximum as well. In fact the pointer between the two copies can lead to an elements in the middle of the Heap. However you have to explain how to remove a vertex from a middle of the heap.

4. Question 4: Give an efficient data structure supporting the following operations.
• *Insert*($S, x$): add $x$ to $S$.
• *Delete* − *Max*($S$): Delete the maximum value from $S$.
• *Delete* − 100 − *Max*($S$): Delete from $S$ the 100 largest element.
• *Delete* − 100 − *Min*($S$): Delete from $S$ the 100 smallest element.

5. **Question 5:** Show how to implement First in First out with a priority Queue. Show also how to implement a stack in a priority Queue.