CS2223
HW#3

DUE: Friday, March 31

1. (8 points) We saw that INSERTIONSORT takes $\Omega(n^2)$ time in the worst-case when all n elements of A are distinct. Prove that INSERTIONSORT takes $\Omega(n^2)$ time in the worst-case even when A only has two distinct elements. That is, if $A[i] \in \{0,1\}, 1 \le i \le n$, then in the worst-case $\Omega(n^2)$ *swap*s are still necessary. Hint: Every *swap* "corrects" an *inversion*, that is, a pair (i, j) such that $1 \le i < j \le n$ and $A[i] > A[j]$. For every $n \ge 1$, find an instance of $A[1..n]$ which has $\Omega(n^2)$ inversions whose order must be corrected. With fewer distinct elements in A, one might suspect that there will be fewer inversions

2. (12 points) There are many possible implementations for the abstract data type (MAX)-PRIORITY QUEUE. For each of the following data structures to implement a (MAX)-PRIORITY QUEUE A, give a brief description of the data structure, and describe the asymptotic worst-case time (using Θ-notation) to implement INSERT and DELETE-MAX.

- Implement A as an ordered linked list.
- Implement A as an ordered array.
- Implement A as an unordered array.

3. (10 points) It is known that:

- a (MIN)-HEAP can be built in $O(n)$, and
- a (MIN)-HEAP is somewhat ordered, and,
- INSERTIONSORT works fairly well on a somewhat ordered array.

Test this conjecture by writing a program which first calls BUILD-MIN-HEAP and then INSERTIONSORT. Test and then time this program for several values of n, and derive a function describing how much time your program takes, as a function of n, on your system.

CS2223
HW#3 SOLUTIONS

1. Consider $A[i] = \begin{cases} 1, \text{if } 1 \le i \le \lceil n/2 \rceil \\ 0, \text{if } \lceil n/2 \rceil + 1 \le i \le n \end{cases}$. It has $\lceil n/2 \rceil \left(n - \lceil n/2 \rceil \right) \in \Omega\left(n^2\right)$ inversions. Each inversion must be "corrected" by a *swap* in INSERTIONSORT, so that for this class of input instances $\Omega\left(n^2\right)$ *swap*s are necessary.

2. For an ordered linked list, the elements are stored in decreasing order. To INSERT, the list is traversed (worst case $\Theta(n)$ time), until the proper place is found. To DELETE-MAX, the first element is removed from the list and returned.

For an ordered array, the elements are stored in increasing order. To INSERT, the place for a new element can be found in $O(\lg n)$ time using binary search. But moving all elements below it down in the table takes $\Theta(n)$ time, even in the average case. To DELETE-MAX, the last element is removed and returned.

For an unordered array, the elements are stored in any order. To INSERT, the new element is stored in the end, in constant time. To DELETE-MAX, the elements are all examined until the largest is found. After replacing it with the last element, it is returned.

	INSERT	DELETE-MAX
ordered linked list	$\Theta(n)$	$\Theta(1)$
ordered array	$\Theta(n)$	$\Theta(1)$
unordered array	$\Theta(1)$	$\Theta(n)$