Problem Set III

1. Does there exist a function $f(n)$ such that $f(n)$ is $O(n)$ and $\Omega(n^2)$?
2. Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of θ notation. (Cormen, 2nd edition 1.2-5)
3. Show how to implement a queue using two stacks. Analyze the running time of the queue operations. (You have to implement the functionalities of a queue using the functionalities of a stack.)
4. Show how to implement a stack using two queues. Analyze the running time of the stack operations.
5. Implement a stack using a singly linked list L. The operations PUSH and POP should still take $O(1)$ time. (Cormen, 2nd edition 11.2-2).
6. Implement a queue using a singly linked list L. The operations ENQUEUE and DEQUEUE should still take $O(1)$ time. (Cormen, 2nd edition 11.2-3).
7. Write the *Insert, Find* (both versions of Find) and *Delete* operations using a doubly linked list. Compare the run time complexities with that of a singly linked list.
8. Where in a heap might the smallest element reside? (Whether it is in the leaf node or at an internal node. Should it be at any level or at the highest or lowest level)
9. Is an array with elements sorted in non-increasing order, a heap?
10. Write an efficient *Heapify* that uses an iterative control construct (a while or a for loop) instead of recursion. Note that the runtime complexity should not change.
11. Show that there are at most $\lceil n/2^{h+1} \rceil$ nodes of height h in any n-element heap. (Cormen, 2nd edition 7.3-3).
12. Consider the implementation of the heap tree, where each node of the tree is a structure, named $Node$, having three attributes: **Priority, Left-Child** *(Pointer to Left Node),* **Right-Child** *(Pointer to Right Node).* Answer the following:
    (a) Rewrite $Heapify$, where the input parameter would be a pointer to the root of the subtree, you want to Heapify.
    (b) Analyze how the *Build-Heap* algorithm can be implemented, when (i) the input parameter is an Array A (ii) the input parameter is a binary tree with the heap structure. Analyze the runtime complexities in each of the cases.