1. Write a program that takes as input a sequence of distinct integers, until the character x is read. It then iteratively inserts them into the "middle" of a *doubly linked list*, as detailed below.

   Let, $j = (i+1)/2$ if i is odd and $j = i/2$, otherwise. Then, after the i^{th} insertion, the i^{th} integer should be present in the j^{th} node of the list (assume that the *head* pointer points to the 1^{st} node of the list). The relative ordering of the other nodes should not change.

2. Modify your program for the preceding question by adding a new function that would delete the $\lceil n/2 \rceil^{th}$ node, where n is the number of elements in the queue at the time of invoking the function. Your implementation should run in constant time.

3. Write a program to perform *Insertion Sort* on a doubly linked list of integers.

   (*Hint:* Try to simulate the comparisons performed during Insertion Sort on an array.)

4. Write a recursive function for searching an element in a singly linked list.

5. Assume that you are given the *head* pointer of a linked list l, that contains n nodes, for some unknown integer n. Note that n is **not** part of the input. Specified below is the function $kLast$.

$kLast$
Input: The *head* pointer of a linked list and an integer k, such that $k \leq n$, where n is the length of the list.
Question: The $(n-k+1)^{th}$ node in the list.

   Implement this function by traversing the linked list *only once*. That is, you *may not* compute the value of n first.

   (*Hint:* Think recursively!)

6. Write a program to implement a stack S using an array A of size $n = 8$ as the underlying data structure. The stack must support the standard *isEmpty*, *PUSH*, *POP* and *isFull* operations.

7. In the preceding question, modify the *isFull* and/or *PUSH* functions to support the following functionality.

   If the stack is full when the *PUSH* function is called, allocate a new array B of size $2 \times n$, copy all the elements of A into B, make A point to array B, deallocate the old array A, and finally perform the *PUSH* operation on the new array A.

8. A priority queue supports the following operations.

   ISEMPTY(Q): Return *TRUE* if the queue is empty.

   ENQUEUE(x, Q, p): Enqueue the integer x into the queue Q, with priority $p \geq 0$. Note that the priorities of different elements need not be distinct.

   DEQUEUE(Q): Dequeue the highest priority node that was first inserted into the queue Q, and print its value.

   Write a program to implement a priority queue that uses a *singly* linked list as the underlying data structure. Your implementations for ENQUEUE and DEQUEUE should, respectively, take $\mathcal{O}(n)$ and $\mathcal{O}(1)$ time in the worst case, where n is the number of elements in the queue at the time of performing the operation.

9. Suppose that, in the preceding question, the priorities are integers in the range $[0, 10]$. Modify your program so that each of the priority queue operations take $\theta(1)$ time only.