Linked Lists

Big O:

- space O(n)
- time
  - search worst O(n), average O(n)
  - insert worst O(1), average O(1)
  - delete worst O(1), average O(1)

Advantages:

- linked lists are a dynamic data structure, allocating the needed memory while the program is running
- insertion and deletion node operations are easily implemented in a linked list
- linear data structures such as stacks and queues are easily executed with a linked list
- they can reduce access time and may expand in real time without memory overhead

Disadvantages:

- they have a tendency to use more memory due to pointers requiring extra storage space
- nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access
- nodes are stored incontiguously, greatly increasing the time required to access individual elements within the list
- difficulties arise in linked lists when it comes to reverse traversing
  - singly linked lists are cumbersome to navigate backwards[1] and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back pointer

Uses:

- stack

- queue
- memory allocation

Creating a Linked List:

```
class Node {
   Node next = null;
   int data;
   public Node(int d) { data = d; }
   void appendToTail(int d) {
      Node end = new Node(d);
      Node n = this;
      while (n.next != null) { n = n.next; }
      n.next = end;
   }
}
```

Deleting a node:

```
Node deleteNode(Node head, int d) {
   Node n = head;
   if (n.data == d) {
      return head.next; /* moved head */
   }
   while (n.next != null) {
      if (n.next.data == d) {
         n.next = n.next.next;
         /* head didnt change */
         return head;
      }
      n = n.next;
   }
}
```

Notes:

- alternative to array to implement stack and queue
- allows any length