

## CSCI 204: Data Structures & Algorithms

### Singly Linked lists

Revised based on textbook author's notes.

### Consider this problem

- Given a sorted list of numbers, insert a new number into this sorted list
  - [3, 5, 7, 10, 12, 20]
  - insert 6 into this list to become [3, 5, 6, 7, 10, 12, 20]
- How do YOU accomplish this seemingly simple task? Take 3 min to work it out. Consider what may happen with your solution.

### Here is a possible solution

```
def insert_sorted(k, my_list):
    """ Insert k into a sorted list my_list"""
    new_list = [k for x in my_list + 0] # have k to occupy a spot!
    i = find_pos(k, my_list)
    # shift elements from i to len(my_list)-1, i-1, -1): # shift each element right
    new_list[i:] = new_list[i-1:]
    # new list k where it belongs to
    new_list[i] = k
    return new_list

def find_pos(k, my_list):
    """ Find where k should be in the sorted list my_list"""
    i = 0
    while i < len(my_list) and k > my_list[i]:
        i += 1
    return i # i could be len(my_list)!
```

How many steps to complete this work?  
It takes  $4*n$  steps to do this.  
Can we do better? How?

### Let's look at one issue at a time

- $new\_list = [x \text{ for } x \text{ in } my\_list] + [k]$ 
  - We need to increase the capacity of the list to hold the new element. Whether the list is implemented as a Python list or an array, this would take  $n$  steps.
- $i = find\_pos(k, my\_list)$ 
  - We need to find the right spot for the new number, which takes  $n$  steps
- for  $j$  in  $range(len(new\_list)-1, i-1, -1)$ :
  - Shifting elements to the right takes  $2*n$  steps

### Linked lists

- Use linked list we can make two of the three operations in constant time!

### Linked Structure

- Constructed using a collection of objects called **nodes**.
- Each node contains data and at least one reference or **link** to another node.
- Linked list** – a linked structure in which the nodes are linked together in linear order.



## Linked List

- Terms:
  - node** – each element in the list.
  - head** – first node in the list.
  - tail** – last node in the list; link field has a null reference.



## How does it look like in Python?

- The nodes are constructed from a simple storage class:

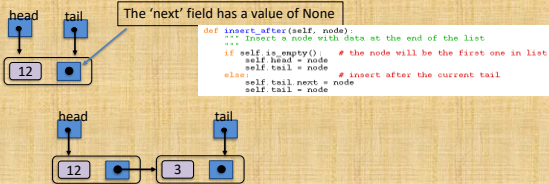
```
class ListNode:
    def __init__( self, data ):
        self.data = data
        self.next = None
```

- List contains two nodes, a head and a tail

```
class UserList:
    def __init__( self ):
        self.head = None
        self.tail = None
```

## How to build a list?

```
my_list = UserList() # initial list head == None
a_node = ListNode(12) # create a node
my_list.insert_after(node) # insert the node to list
my_list.insert_after(ListNode(3)) # another node
```



## Try out an example

```
from random import *
from userlist import *

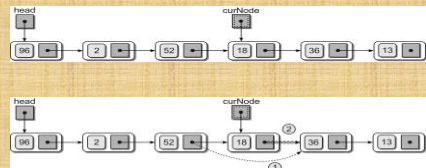
def test_list():
    """ A test program for singly linked list """
    my_list = UserList()
    nums = [randint(1, 100) for i in range(10)]
    for x in nums:
        my_list.insert_after(ListNode(x))
    print(my_list)

test_list()
```

Exercise: write the function `insert_before()` that inserts the node before the head.

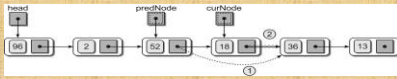
## Removing nodes

- An item can be removed from a linked list by removing or unlinking the node containing the item.
  - Find the node containing the item.



## Removing nodes

- Removing a node from the middle of the list requires a second external reference.



## Removing Nodes

- Removing the first node is a special case.
- The head reference must be repositioned to reference the next node in the list.



## Removing nodes

- Given the head reference, we can remove a target from a linked list.

```
def remove_node(self, target):
    """ Remove the node containing the target """
    prev = None
    cur = self.head
    while cur != None and cur.data != target:
        prev = cur
        cur = cur.next
    if cur != None: # found it and remove it
        if cur == self.head:
            self.head = cur.next # head is removed, reset head
        else:
            prev.next = cur.next # remove a middle node
```