8.1 - Week 8 - Applied Theory

Objectives of this Applied Session

- To understand the differences between sorted lists and non-sorted lists implemented with arrays.
- To understand the implementation of linked lists, and their differences with an array implementation.
- To continue testing knowledge of complexity

SortedListADT

Let us take you through another ADT, and this one will be a complicated one.

We are going to work through a SortedListADT. It has certain benefits and also, certain drawbacks.

But first, let's go through some basics.

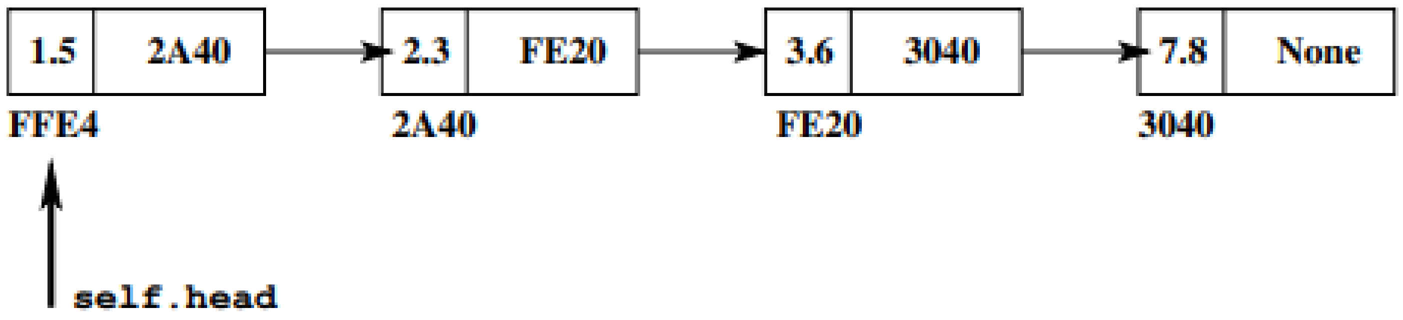
- What is the basic difference between a sorted list and an unsorted list?
- If you had a sorted list of the following items, what would be the key to sorting:
  - Names?
  - Bank Balances?
  - Grades?
- Can you think of scenarios where a sorted list would be beneficial?

Let's discuss the answers to these questions as a group, but in summary:

- A list is an ordered collection of items. That order can be randomly allocated. However, A sorted list is one that is sorted by the value in the key; there is a semantic relationship among the keys of the items in the list.
- You would probably sort it in the order of magnitude of keys. For example:
  - Names -> Probably sort by the ASCII value of the first letter (going down the string to break ties)
  - Bank Balances -> Probably sort by the amount held in each account
  - Grades -> Probably go in a pre-determined order (HD -> D -> C -> P -> N)
- Sorted lists can be extremely beneficial in various scenarios such as creating a leaderboard, storing a dictionary, calculating performance and keeping a track of the values, and several other scenarios.

Exercise 1*

Consider the **LinkList** class described in the lectures. Consider an object **my_list** of this class, to which we have appended four floats (1.5, 2.3, 3.6, and 7.8). As a result, the nodes of the list can be visualised (in high-level form) as:



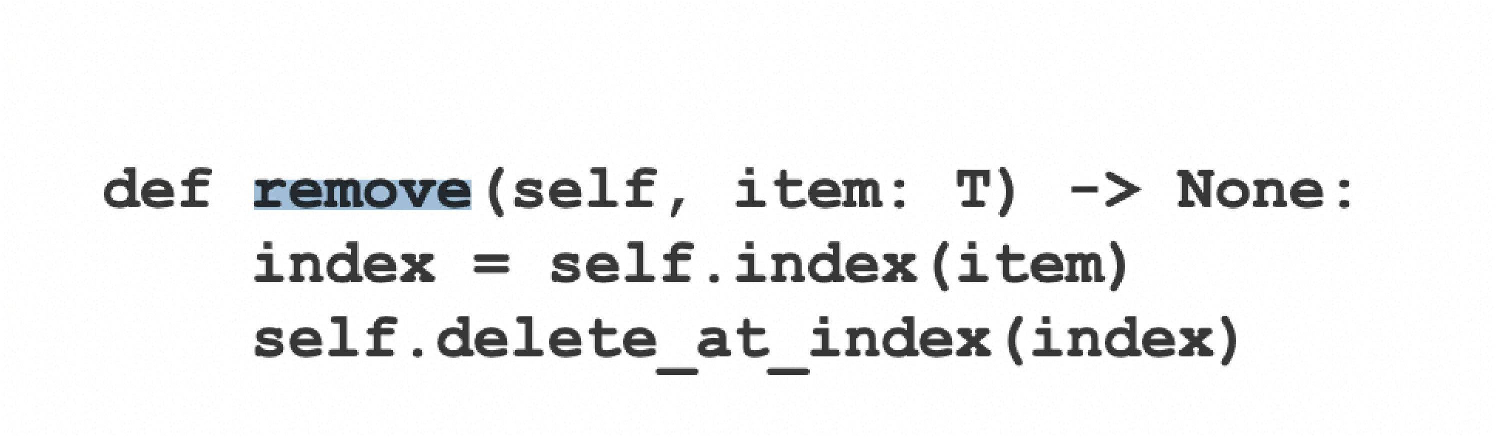
where the hexadecimal number under each node represents the address of the node object.

In `Ex1.txt` put down the values of the following variables *after* executing **item** = **my_list.remove(1.5)**:

1. my_list.head
2. my_list.length
3. item
4. my_list.head.link

Note: you may know .link as .next

For reference the remove() function:



```
def remove(self, item: T) -> None:
    index = self.index(item)
    self.delete_at_index(index)
```

```
def remove(self, item: T) -> None:
    index = self.index(item)
    self.delete_at_index(index)
```

Exercise 2*

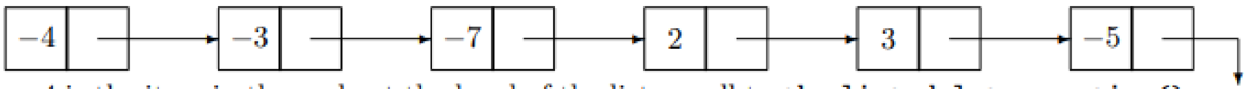
Taking the median is a type of averaging different from the usual averaging method (known as taking the mean). The median value of a set of numbers is a value x that separates the higher half from the lower half. It can be computed by sorting the values and taking the middle element for an odd-sized set or by averaging the two middle elements for an even-sized set of numbers. For example, the median of the list [3, 4, 7, 7, 9, 218] is 7.

In `Ex2.txt`, state and explain the worst-case Big O time complexity of a good algorithm that must find the median of a sorted list of N integers, in each of the following:

1. A sorted list implemented using arrays
2. A sorted list implemented using linked nodes

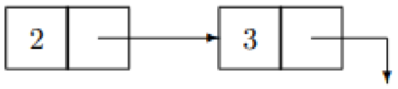
Exercise 3*

Consider the **LinkList** class from the lectures. In `Ex3.py`, add a method **delete_negative** to that class that eliminates from the list any node containing a negative number. For example, given the linked list:



where -4 is the item in the node at the head of the list, a call to `the_list.delete_negative()` would leave

the list as:



where 2 is the item in the node at the head of the list.

Exercise 4*

Consider the **LinkList** class we have seen in the lectures and consider the following function that accepts two of those lists:

```
def mystery(a_list1:LinkList[T], a_list2:LinkList[T]) -> None:
    if a_list1.head is None:
        a_list1.head = a_list2.head
    else:
        current1 = a_list1.head
        current2 = a_list2.head
        while current2 is not None:
            temp = current1.link
            current1.link = current2
            if current2.link is None:
                current2.link = temp
                current2 = None
            else:
                current1 = current2.link
                current2 = temp
    a_list2.head = None
```

Consider a call to **mystery(a_list1, a_list2)** where **a_list1** is a list with elements 1,2,5,3,8 (in that order) and **a_list2** is a list with elements 0,9,7,4,6,1,0,5 (in that order). In Paint (or alternatively this), draw the memory diagram for the resulting lists before and after the function is executed.

In `Ex4.txt`, answer the following questions:

1. Explain what the function does.
2. What happens to the heads of **a_list1** and **a_list2**. What does this mean for the algorithm?
3. What is the best and worst Big O time complexity of our **mystery** function in terms of the lengths **n1** and **n2** of the lists? Give an explanation.

Exercise 5

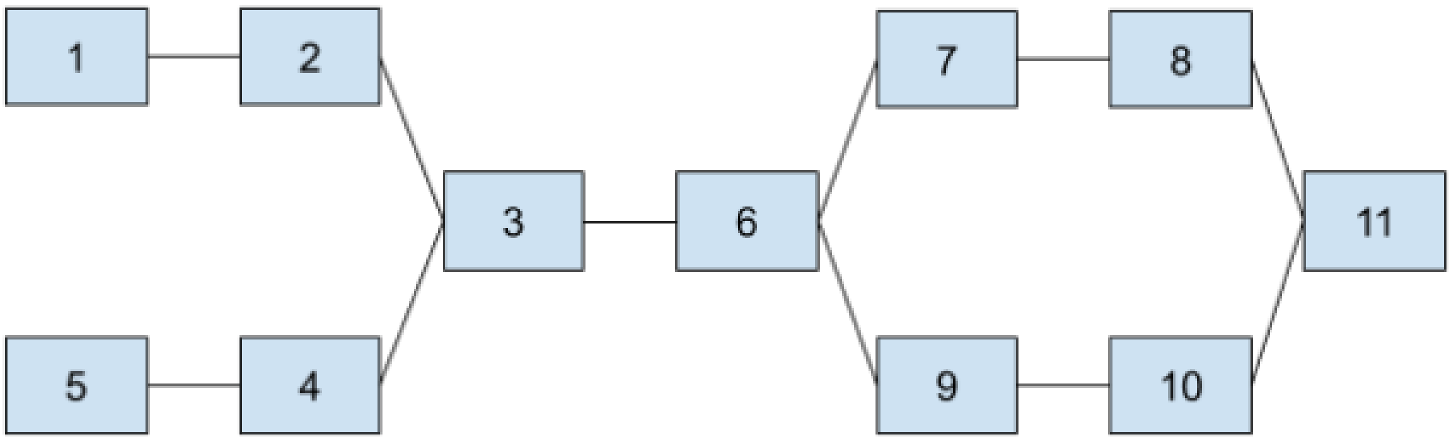
In `Ex5.py` define a function **print_repeating** which has a list of values as a parameter. Then for each item in position k in that list, the function prints the item $k + 1$ times. In other words, the number of times each item in the list is printed equals the position of that item in the list plus 1.

For example, if the input list has the elements [20,-1,2,10,7] then the method should print the items in the following sequence (though the exact layout is not critical): 20, -1, -1, 2, 2, 2, 10, 10, 10, 10, 7, 7, 7, 7, 7. That means 20, which is in position 0 is printed once, -1 which is in position 1 is printed twice, and so on.

In `Ex5.txt`, give and explain the best and worst Big O complexity of your function.

Exercise 6

Consider the following mystery linked nodes structure below, where the direction of each line pointer is not shown:



In `Ex6.txt`, answer the following questions:

1. Using the **LinkList** class we have seen in lectures, do you think it is possible to create a linked list having the same structure as the mystery linked nodes above with just one instance? Explain.
2. If you are allowed to modify the **LinkList** or **Node** implementations, state what would you change so that you can create a linked list having the same structure as mystery linked nodes above with just one instance. Explain how this modification allows you to achieve the goal.