210CT Assignment

Block 3

Pseudo Code

The Linear search algorithm works by going through the list/array that it is handed 1 item at a time checking if it is what it is looking for, if it finds the item them it returns true, if it does not then it returns false. Linear search is not particularly efficient, especially when it comes to long lists where something with O(log n) like binary search would be better, although linear can faster when searching short arrays, this is because it has very little overhead.

The 2nd functions is the recursive version of the linear search (first on being iterative) there isn't too much difference except it call its self instead of using a loop. Recursion can be useful in some situations but when iteration can be used instead then it is better because it has less overhead, this is because recursion uses more memory because the code creates a new instance of its self each time the function is called.

The list duplicate finder works in a similar way to the just instead of the function being handed the item to look for it make "j" (j being and item in the array) what it is looking for. It then compares j to the rest of the array and if it finds a match it return true if not it increments j to the next item in the array, when j reaches the last item in the array and if no matches have been found it returns false because there are no duplicates.

```
1.  #This function is a liniar search that goes through the array it is handed until it finds the item
     it is looking for
2.  LINEARSEARCHITERATIVE(listToSearch,item)
3.      FOR i IN RANGE (0 TO (Length of listToSearch))
4.          IF i > (length of listToSearch) #when it reachs the end of the array it retruns false beca
    use the item is not in the array
5.              RETURN FALSE
6.          ELSE IF listToSearch[i] = item #when the item is found by this line it retruns true
7.              RETRUN True
8.          ELSE
9.              i ← i + 1 #if the item is not found and it is not the end of the array it increments
    i and does the for loop again
10.
11. LINEARSEACHRECURSIVE(ListToSearch,item,n)  #This is the recursive version of the linear search
12.     IF n > (length of listToSearch)
13.         RETURN FALSE
14.     ELSE IF listToSeatch[n] = item
15.         RETRURN TRUE
16.     ELSE
17.         LINEARSEACHRECURSIVE(ListToSearch,item,n+1)  #this is the main difference, the function
    calls its self and increments n instead of using a loop
18.
19. #This function tries to find duplicates in the array it is handed
20. LISTDUP(listToSearch)
21.     i←0
22.     j←0
23.     FOR i IN RANGE (0 TO (length of listToSearch)) Increment i
24.         IF i ← (length of listToSearch) #if i reaches the end of the list it is reset to the begi
    ning and j moves to the next item
25.             j ← j + 1
26.             i ← 0
27.         ELSE IF j > lenth of listToSearch) #when j reaches the end of the list all the items have
    been compared so if nothing has been found it will return false
28.             RETURN FALSE
29.         ELSE IF listToSearch[i]=listToSearch[j] AND i != j #if position i and position j are the s
    ame then there is a duplicate so it returns true
30.             RETURN True
```

Philip Bell
5690404

Block 4

Big O

The big O notation for the Linear search is O(n) because as the list it is handed increases in size the time it take to process it is the same amount. In a best case scenario it can me O(1) if the first item in the array is what the search is looking for but in a worst case it can look though the entire array and the item not even be there.

The big O notation for the duplicate finder is O(n^2), this is because for every item in the array the for loop has to loop through the array.

Philip Bell
5690404

Block 5

Pseudo Code 1

The objective of this algorithm is to find a value in the list handed to the function that is more than l and less than u, is it finds an item in the array that makes morethenL and lessthenU both true then it will return true. If not then it will return false.

```
1.  arrayA=[]
2.  int l
3.  int u
4.  FUNCTION1(arrayA,l,u) #this functions tries to find and item in the array "arrayA" that is more th
    e "L" and less than "U"
5.      FOR i IN RANGE OF END OF arrayA #loops though for loop until
6.          IF arrayA[i] > l #makes the bool "moreThenL" true if the item in the array is more then l
    if note makes it false
7.              morethenL  ← true
8.          ELSE
9.              morethenL ← false
10.         IF arrayA[i] < u  #make the bool "lessThenL" true of the item in the array is less then u
    if note makes it false
11.             lessthenU ← true
12.         ELSE
13.             lessthenU ← false
14.     IF moreThenL =true AND lessThen = true #if both lessThenL and moreThenL is both true then the
    item in the array is more the L and less the U it returns true if not return false
15.         RETURN true
16.     ELSE
17.         RETURN false
18. FUNCTION1(arrayA,l,u)
```

Big O 1

The Big O notation of algorithm (above) O(n) this is because will only have to at a worst case have to check every item in the array twice, once when comparing l and once when comparing u. In a best case it will only need to

Python code

This is the python code for the pseudo code above; because it is in python, there is not actually that much difference in syntax.

```
1.  #this functions tries to find and item in the array "arrayA" that is more the "L" and less than "U
    "
2.  def moreLessChecker(arrayA,l,u):
3.      for i in range (0,len(arrayA)): #loops though for loop until
4.          if arrayA[i] > l: #makes the bool "moreThenL" true if the item in the array is more then l
     if note makes it false
5.              moreThenL = True
6.          else:
7.              moreThenL = False
8.          if arrayA < u: #make the bool "lessThenL" true of the item in the array is less then u if
    note makes it false
9.              lessThanU = True
10.         else:
11.             lessThanU = False
12.     if moreThenL = True and lessThanU = True: #if both lessThenL and moreThenL is both true then t
    he item in the array is more the L and less the U it retruns true if not return false
13.         return True
14.     else:
15.         return False
```

Philip Bell
5690404

Pseudo Code 2

This is the pseudo code is sorted ranger search, this works by ordering the algorithm before it searches through. This allows the algorithm when it finds an incorrect value to delete that value and all the values before it or after it because of it being ordered meaning they all also will be incorrect. This is called divide and conquer and this means the algorithm will find what it is looking much faster because it removed a bunch of incorrect values with every pass.

```
1.  arrayA=[]
2.  int l
3.  int u
4.  FUNCTION2(arrayA,l,u)
5.      ORDER arrayA
6.          FOR i IN RANGE OF END OF arrayA
7.          IF arrayA[i] > l #makes the bool "moreThenL" true if the item in the array is more then l
    if note makes it false and removes it from the array
8.              morethenL ← true
9.          ELSE
10.             REMOVE arrayA[i] TO arrayA[0]
11.             morethenL ← false
12.         IF arrayA[i] < u
13.             lessthenU ← true
14.         ELSE
15.             REMOVE arrayA[i] TO arrayA[1] #make the bool "lessThenL" true of the item in the array
     is less then u if note makes it false and removes it from the array
16.             lessthenU ← false
17.     IF moreThenL =true AND lessThen = true #if both lessThenL and moreThenL is both true then the
    item in the array is more the L and less the U it returns true if not return false
18.         RETURN true
19.     ELSE
20.         RETURN false
21. FUNCTION2(arrayA,l,u)
```

Big O 2

The Big O notation of search with removal of already used items is O(log n), this is because with the removal of items from the search the search will get gradually faster. An algorithm that uses divide and conquer like the algorithm above does will have

Philip Bell
5690404

Block 6

Pseudo Code

This code is a harmonic series, it works adding half of the previous value to be added. Equation being:

$$1+\frac{1}{2}+\left(\frac{1}{3}+\frac{1}{4}\right)+\left(\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\frac{1}{8}\right)+\ldots$$

Wolfram(2015)

```
1.  HARMONIC(n) #recives what n which is what number in the harmonic sequence is wanted, it then retru
    ns that value
2.          harmTot ← 0
3.          harmStart ← 1
4.          FOR i IN RANGE(0,n) #loops through the for loop until the desired harmonic value is output
    ted
5.              harmTot ← harmStart+harmStart/2
6.              harmStart ← harmStart/2
7.          RETURN harmtot
8.  n ← 10
9.  HARMONIC(n)
```

Python Code

Once again the python code is not that different, you do have to make sure to divide “2.0” not “2” because python will round if use “2” and the algorithm will not work.

```
1.  def harmonic(n): #returns harmTot which will be the nth value in the harmonic series
2.      harmTot = 1
3.      harmStart = 1
4.      for i in range(0,n): #loops through n amount of times to genereate the nth value in the harmon
    ic sereis
5.          harmTot=harmTot+harmStart/2.0 #makes the current total by added the previous total to half
     the previous harmonic value
6.          harmStart=harmStart/2.0 #halfs the value that is added
7.      return harmTot
8.
9.  n=10
10. print(harmonic(n)) #calls the functions and hands it the parameters
```

Philip Bell
5690404

Block 8

This is a node delete algorithm, I have implemented the delete function into the code provided, the code works by handing the delete function the node that need to be deleted, if it is the head then head gets changed to the next node, if it is the tail it is changed to the previous node. If it is a node in the middle, it is changed to the previous or next node depending on where it is on the list and if the previous or next node are not empty.

```
1.  class Node(object):
2.      def __init__(self, value):
3.          self.value=value
4.          self.next=None
5.          self.prev=None
6.
7.  class List(object):#inserts the given parameters to the list
8.      def __init__(self):
9.          self.head=None
10.         self.tail=None
11.     def insert(self,n,x):
12.         if n!=None:
13.             x.next=n.next
14.             n.next=x
15.             x.prev=n
16.             if x.next!=None:
17.                 x.next.prev=x
18.         if self.head==None:
19.             self.head=self.tail=x
20.             x.prev=x.next=None
21.         elif self.tail==n:
22.             self.tail=x
23.     def delete(self,n): #this will delete a Node
24.         if n.prev != None: #if next n isnt empty then the previous next n will become the next n
25.             n.prev.next = n.next
26.         else: #if not head will become the next n
27.             self.head = n.next
28.         if n.next != None: #if next n is empty then the next previous n will become the previous
29.             n.next.prev = n.prev
30.         else: #if not tail will become the previous n
31.             self.tail = n.prev
32.     def display(self): #outputs the list to the screen
33.         values=[]
34.         n=self.head
35.         while n!=None:
36.             values.append(str(n.value))
37.             n=n.next
38.         print "List: ","".join(values)
39.
40. if __name__ == '__main__':
41.     l=List()
42.     l.insert(None, Node(4))
43.     l.insert(l.head,Node(6))
44.     l.insert(l.head,Node(8))
45.     l.display()
46.     l.delete(l.head)
47.     l.display()
```

Philip Bell
5690404

References

Wolfram(2015) - (http://mathworld.wolfram.com/HarmonicSeries.html)

Philip Bell
5690404