2 weeks to midterm

Take a sequence of data and sort it, come up with algorithm to sort in a certain way

is there a fast way to go through it? Not just one by one

list data type function – tool allowing us to work with a sequence of data

\*\*\*

Def liststuff():

# create an empty list

A = list()

#or

B = []

# create a populated list

C = [1,2,3,4]

#add elements to a list

D = c+5 # doesn’t work, cant add integer this way to a list

D = c+[5]

#we can access individual elements or slices like with strings

E = d[3]

F = d[1:3]

NEW STUFF

# we can mix types within a list

G = [1,’hello’,4.5.True,None, [1,2,3,4]] #heterogeneous list

# lists are mutable!!

D[1] = 100 #changes 1st index to be 100

#have functions associated with them, append, etc.

Print(a,b,c,d,e,f,g)

# tuples cannot be changed, lists that cant be changed, immutable

\*\*\*\*

Memory models:

A = 6

B = a

A = 4

#has b changed, no b retains 6

A = [1,2,3]

B = a

# a and b reference same list, if the **contents** of that list change, then all references to that list adopt that change of the indeces in the list

A[1] = 24

Has b changed? Yes

A = [2,4,6]

B = a

A = [3,6,9]

# entire new list created to have a reference, b stays referencing the first list

\*code for above model\*

Def memorymodel():

A = 6

B = a

A = 4

Print”what are a and b?” + str(a) + ‘,’ + str(b))

#changing a component of the list

A = [1,2,3]

B = a

A[1] = 33

Print”what are a and b?” + str(a) + ‘,’ + str(b))

#creating a new reference for a does not change b

(((((Code….)))))

\*\*\*\*\*\*

A = 6

Foo(a)

Print(a)

Def foo(b):

B = 4

#a is still 6a = [1,2,3]

Foo2(a)

Print(a)

Def f002(b):

B[1] = 9

# a maintains its reference but the contents may change

A = [1,9,30 result

A = [1,2,3]

Foo3(a)

Print(a)

Def foo3(b)

B = [2,4,6]

#a refernces same list, still [1,2,3]

\*\*\*

#Splitting is a string function that splits a long string into elements and returns a list

Def splitrange():

Mystr = ‘Here is a sentence with a few words’

Mystrlist = mystr.split()

Print(mystrlist)

Mystr2 = ‘apples,bananas,plums,carrots,lettuce.

Mystr2 = mystr2.split()

Mystr2list2 = mystr2.split(‘,’)

Print(mystr2list)

Print(mystr2list2)

Mystrlist()

\*\*

Range in for loops

#we know this

For el in mystrlist:

Print(el)

#using range:

For idx in range(len(mystrlist)):

Print(mystrlist[idx])

#quivalently

For idc in range(0,len(mystrlist),l):

Print(mystr[idx])

#every other element:

For idx in rrange (0,len(mystr),2):\#equivalent;ly:

For idx in range(len(mystrlist)):

If(idx % 2 == 0):

Print(mystrlist[idx])

\*\*

Break here

\*\*

[5,2,3,6,7,1,4]

Selection sort, find the lowest value and put it in front, look at those that haven’t been sorted and move it forward

Skip 0 indez, do second, skip 1st index, do second and beyond, etc.

Insertions sort:

At each step k, we want to make sure the first k elements are sorted.

Step one, 5 is already sorted

Step one, sort first element, leave list alone

Step 2, sort elements 1 through 2 knowing that element 1 is alredy sorted.

At end of step 2 [5,2,3,6,7,1,4]

Step 3;

Sort element 3 knowing elements 1 trough 2 are already sorted

[2,3,4,6,7,1,4]

After steps, the first few are sorted.

Each iteration

Identify current element

Compare to previous sorted elements

2 stopping conditions:

1. if element to left is lessthan/equal than current element, stop

When there is nothing to the left, empty string, then you sotp sorting

Def insertionsourt(lst):

For idx in range(1,len(lst)):

#each iteration is responsible to make sure the first idx+1 elements are sorted

Temp = lst[idx]

Mark = idx-1

While(mark>=0 and lst[mark]>temp):

lst[mark+1] = lst [mark]

Mark = mark-1

NON INDENTED lst[mark] = temp

Def test()

Lst = [2,3,4,5,1]

Insertionsort(lst)

Print(lst)

Lst = []

Insertionsort(lst)

Print(lst == [])

Lst = [1,2,3]

Insertionsort(lst)

Print(lst == [1,2,3])

Test()

Big o notation

O(1) = conastant time

O(n) = linear time

Insertion sort of n elements

Outer loop going from 1 up to n

Inner loop checks where element k is locatef

BEST case , inner loop takes one comparison \* 1 = linear function (data comes in already sorted)

WORST case, the mark has to march through the whole list to be sorted (numbers are in reverse

Summation n(n+1)/n

O(n squared)

Binary search:

List = [10,20,30,40,50,60,70,80]

Goal: return index of value 50 or None if it doesn’t in the list.

Def binary\_search(lst,target,left,right):

#returns index of match or else none.

Mididy = (left + right)//2 #integer division

Val = lst[mididdy]

#stopping conditions,

#1. We cfound target

#o

#2. Right end point<left end point

1 it = n

2 it = n/2

3 it = n/4

.

.

.

2^k = n

At ^ poin, k = log2(n)

Binary search is faster than a function that goes character by character.