Sheri Wong Week 5 Sort

1.The bubble sort algorithm compares adjacent items and will exchange them if they are out of order. It will make multiple passes through the list as it places the next largest value in its proper place. From the given example of a selection sort from the sorting module on canvas, it writes

def bubbleSort(alist):

for passnum in range(len(alist)-1,0,-1):

for i in range(passnum):

if alist[i]>alist[i+1]:

temp = alist[i]

alist[i] = alist[i+1]

alist[i+1] = temp

With a list that is given [1,2,3,4,5,6,7,8,9,10], a bubble sort will stop at the if statement as the numbers are not out of order. As it compares the adjacent number, the value it will return will always return false, since the previous number is never bigger than adjacent number. Therefore it will not sort through.

2. A selection sort will make one exchange for every pass through the list. It will find the max value first and compare the length of the list and will put it in the proper location. From the given example of a selection sort from the sorting module on canvas, it writes

def selectionSort(alist):

for fillslot in range(len(alist)-1,0,-1):

positionOfMax=0

for location in range(1,fillslot+1):

if alist[location]>alist[positionOfMax]:

positionOfMax = location

temp = alist[fillslot]

alist[fillslot] = alist[positionOfMax]

alist[positionOfMax] = temp

With a list of [1,2,3,4,5,6,7,8,9,10], it will find the length of the list, and use the range function to loop through the list. It will also count backwards because we are setting the step of the range to be -1. We are also counting from the second to last from the list, not the last value of the list. Then, we set a variable called positionofMax where it would then compare the last of the range value with the position of max value, as the position of max accumulates with each loop. With this example, the position of max will count upwards till 9.

Once it is done with its loop, it will move on to the next part of the code where it compares the position of max with the range of your list. Since in this particular case, the list is already in order, so the position of max is the same position as with range in the list. So there is no need to move anything. This instance of code will loop through nine times through the list until it reaches the end the list.

3. The insertion sort maintains a sorted sublist in the lower positions of the list. It will be sorted in a sublist where each new item is inserted in front of the sublist as it compares which is the higher number. From the given example of a selection sort from the sorting module on canvas, it writes

def insertionSort(alist):

for index in range(1,len(alist)):

currentvalue = alist[index]

position = index

while position>0 and alist[position-1]>currentvalue:

alist[position]=alist[position-1]

position = position-1

alist[position]=currentvalue

With a list of [1,2,3,4,5,6,7,8,9,10] we are also using the length of the list, and using the range function to loop through. We are then using the range function to go through each item in the list and compare it only if the position of that item is greater than 0, and if the value adjacent to the current value is greater then the current value. If this is true, then it will insert a new value into the list. In this instance, the previous value will never be greater than current value since it is already in order. So it will skip that part in the algorithm and will continue to loop through nine times while skipping the insertion part of the code, until the end of the list.