Programming, Algorithms and Data Structures coursework

GitHub page: <https://github.com/davisj23/Programming-Algorithms-and-Data-Structures-course-work>

Question 3

Part 1

**Pseudocode**

A🡨array of numbers

B🡨 array of numbers

C🡨empty array

Counter🡨-1

For I in A

Num🡨 A[counter] + B[counter]

C[insert Num at the start of the array]

Counter🡨 -1

If range of B > A

If range of B + Counter = 0

C[insert B[0] at the start of the array]

Else

C[insert B[0,counter + 1]at start of the array]

Part 4

Part1: the algorithm that would be used for part one is an 0(n) type operation. It scales linearly with the size of the input. so, the larger the input the longer the run time will be.

Part2: This algorithm is 0(n^2). This means it scales polynomial. The reason for this is that the algorithm has a nested loop. this means that if we double the number of elements we want to send though the algorithm the amount of calculations that need to be made quadruple. Because of this it scales poorly compared to 0(n) and 0(1).

Question 4

This algorithm is an 0(n) type operation. The reason for this is that the programme has to reverse the first list to make the second list. it must then compare all elements of the first and the second list to see if they are all identical. Because it takes a constant amount of time to compare each individual element of a list as the lists increase in size so to will the time it takes to compare the two lists. So, the run time will increase linearly with the size of the list

**Pseudocode**

Reversed\_function(A)

B🡨 a[::-1]

If B = A

Print yes

Else

Print no

Question 5

My programme in this question is an 0(n) operation as the time it takes increases linearly with the input. This means that the more words the algorithm must cycle though to reverse them the longer it will take to do so. This is because the algorithm must go though each element in the list one by one.

Question 6

Linear search is a 0(n) operation. This means the amount of time it takes for the algorithm to complete a search will on average increase linearly with the size of the input. the algorithm can complete a search much fast than it should if the element it is looking for is at the start of the list. But if the element does not exist within the list then it will search though all elements in the list one by one until it reaches the end of said list.

*For insertion sort I used this code to help influence me one how I should go about implementing it. Due to this my code for insertion sort might have been heavily influenced by it and it would feel wrong of me to accidently plagiarise someone else’s work.*

Link to selection sort code: <https://stackoverflow.com/questions/15235264/selection-sort-python>

Question 7

Insertion Sort

Element moves = 23

Comparisons = 9

Bubble Sort

Element moves = 23

Comparisons = 81

Selection Sort

Element moves = 10

Comparisons = 10

Testing has show that selection sort is the fastest method for sorting this array of integers as it moved the elements of the array less than half of what the other two sorts did. Bubble sort was sorting method as it had made more than 8 times the comparisons of the other two sorting systems