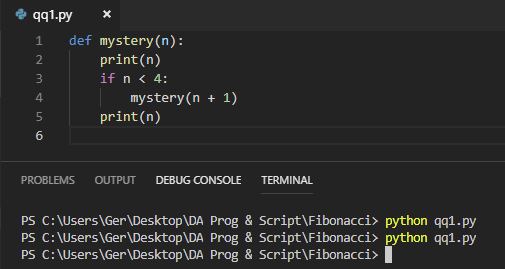
COMP08033 Computational Thinking with Algorithms

Algorithms Problem Sheet (Python)

**Q1.**

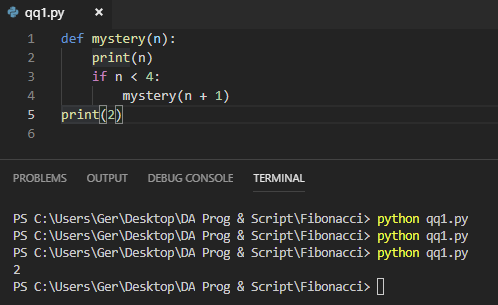
There are several issues with this code.

Firstly, running this python code will not return any result, as the recursive algorithm is not calling itself (mystery). Also it will not produce an error, the program just terminates with no result.

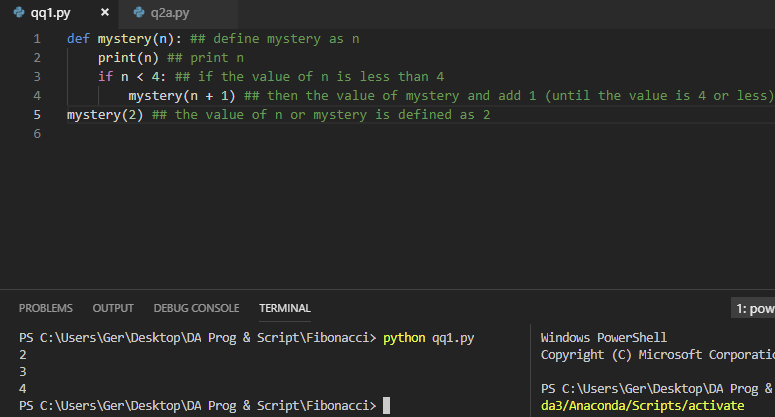


Secondly, the print statement has remained inside the if condition. The code states to print(n) and it should have called mystery(n).

If the code was formatted and the print statement was outside the if statement, the result is 2, as we are telling python to print 2, see below



Finally, n has not been defined so the algorithm does not know where to begin the sequence, if the rest of the code had been written and formatted correctly. Below is an example of the output if n had been defined as 2.



The recursion trace for the call mystery (2)

Call

Return

Mystery (2)

Call

Return

Mystery (3)

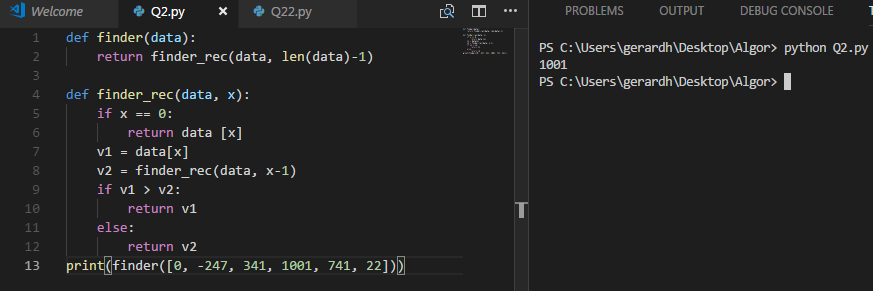
Call

Return

Mystery (4)

**Q2(A)**

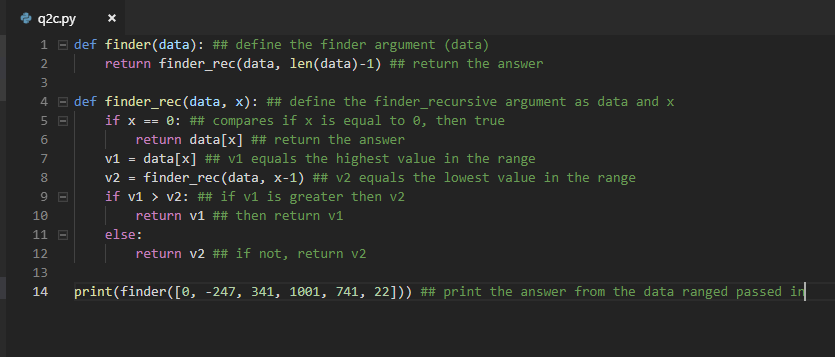
The result of the call to finder will be 1001 for the range stated.



**Q2(B)**

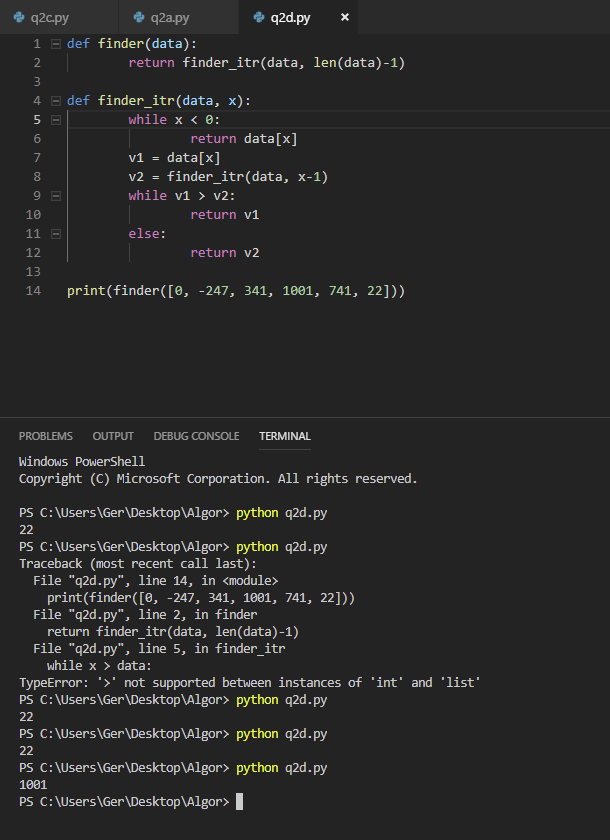
The Finder method is attempting to determine the highest value in the data set.

**Q2(C)**



**Q2(D)**

An iterative version of this function uses a while loop. It involves executing the same block of code over and over. In this function the while loop is executing until it returns the highest value in the sequence of numbers passed in.



**Q3(A)**

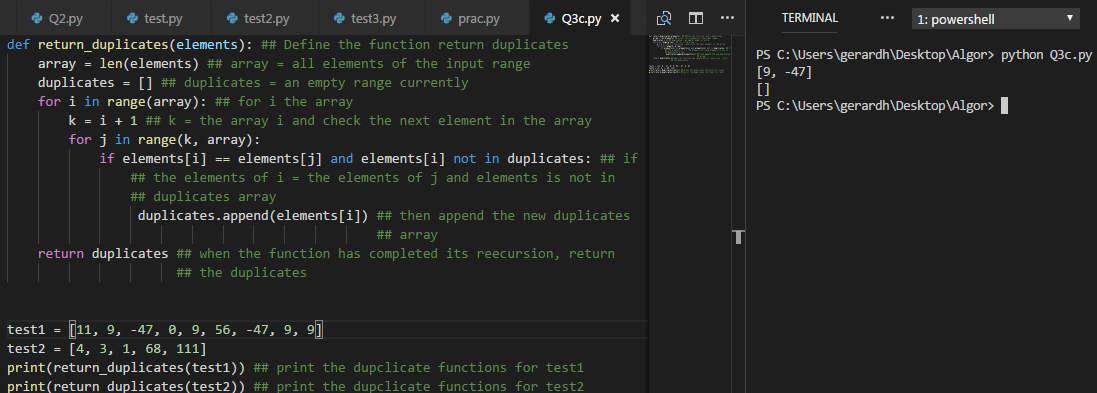
When considering complexity in terms of algorithms the focus is on internal factors, in this case the time required to run the algorithm. We need to consider what is the result on performance as the size of the problem or the input of the dataset gets larger. The characteristics of the dataset will also have an impact on the length of time an algorithm takes to execute. For example, does the dataset contain all positive integers, or a mix of positive and negative, are we comparing words etc, all these factors will need to be considered.

The execution time for this method is constant in the best case, as the number of elements in the dataset is finite. The algorithm loops once through the array, until it finds the first duplicate and then stops, it does not continue checking for any other duplicates.

**Q3(B)**

As discussed in the previous question, the number of inputs in the range will have a direct correlation to performance of the function. Again, I would say that the worst-case complexity would be constant, as the function will terminate if no duplicates are found, but depending on where the duplicates are located in the sequence and the size of the inputs to be checked i.e. the order of magnitude of the inputs will affect the running time of the function. A few hundred or thousand inputs may not have a material impact on performance, but if its hundreds of thousands or millions, then this will result in a noticeably longer running time for the function. The order of growth for this function can be described as O(N2), the deeper the nested recursions in the function the more the order of N will increase.

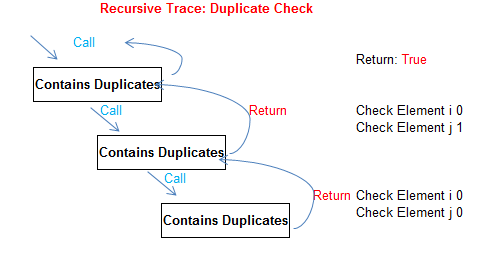
**Q3(C)**



This is a function that will return all of the duplicates found in the arrays passed in [1].

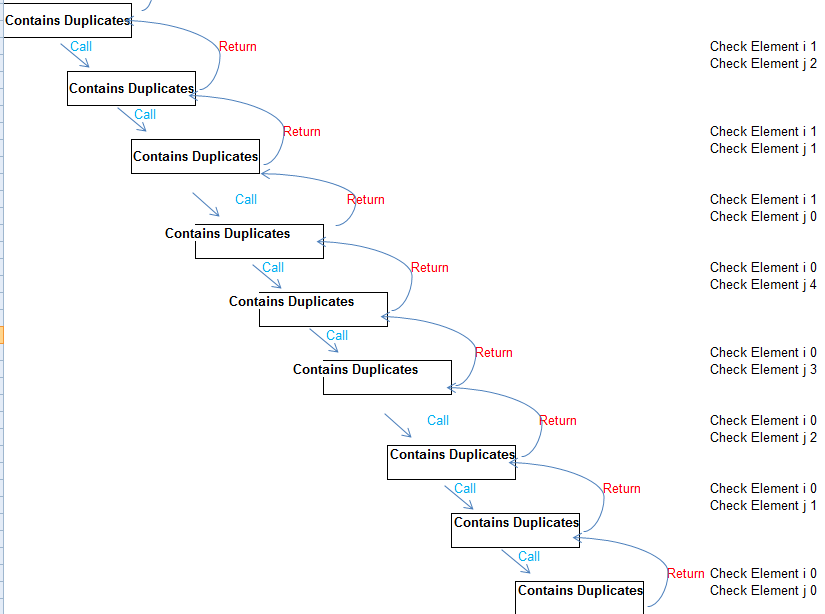
**Q3(D)**

The following instance will exhibit the base-case running time [7, 7, -13, 0, 7]. The program will return true and terminate after the first check of the dataset, it will not continue checking to determine if there are any other duplicates contained in the dataset instance.



**Q3(E)**

The following instance would result in the worst-case running time [10, 0, -13, 7, 7]. This is because the function must run through all of the elements before it finds the duplicate. Each [i] must check each [j] for duplicates, and it will not be until the element 3 in [i] is checked against element 4 in [j] that the duplicate is found. The algorithm has to perform 24 calls to the function before it finds the duplicate, compared to 2 calls to the function in the best-case in the previous question. Below is a portion of the recursion trace for this instance.



**Q3(F)**

The first instance [10, 0, 5, 3, -19, 5] will take longer to process. This is because the program ceases running as soon as it finds a duplicate, and it does not find a duplicate until it checks the last variable in the dataset. In the first instance a duplicate is returned on the third element in the sequence. Element 0 in range [i] is checked against each of the elements in range [j] of the dataset to check if it is a duplicate, self comparison is avoided.

**References**

[1] https://www.geeksforgeeks.org/python-program-print-duplicates-list-integers/