Algorithms

An algorithm is a set of instructions or a procedure that dictates the solution to a problem.

Pseudocode notation is commonly used to represent algorithms so they can be understood by programmers of all languages.

Pseudocode = Notation resembling simplified of imitation programming code.

The standard algorithms of sorting and searching lists

Different algorithms can be used to search for items in a list.

Linear (or Sequential) Searching

Linear search algorithm: Starting at the first element, the program iterates through every item in the list comparing it to the target value until a matching item is found or the end of the list is reached and the item hasn't been found.

Code is usually built into the algorithms which counts the item which is currently being compared. When the target value is identified the index position of this item in the list can therefore be returned.

The efficiency of this algorithm?

Provide the list is sorted:

The first and last item can be compared to the target item. If the target value is at the upper or lower end of the list then the list can be searched ascending or descending to this preference. This means that overall fewer iterations have to be made.

A While loop can be used so iterations only occur up until the target item is located.

This algorithm is not very efficient since all the items in the list must be compared with the target value unnecessarily until it is found or the end of the list is reached.

Worst case scenario = item is last index of list to be checked

However, this does guarantee reliability as all items are checked sequentially and so it does not rely on the list being sorted.

Linear searching is useful for searching a list with a small number of elements or a list that only has to be searched once. It is considered less efficient than binary searching since every element in the list must be checked unit the item found. In the worst case scenario, the item may be the last element in the list so there are many unnecessary comparisons/ iterations made.

The pseudocode:

```
PROGRAM START

        List01 = [3,6,2,4,66,33,0,1]
        search_item = 4
        position = 0
        found = FALSE


        WHILE (position < Length(list01)) AND (found = FALSE)
                IF List01[position] =! Search_item THEN
                        found = TRUE
                        Print("Item is at index position: ", position)
                ELSE:
                        Print("Item not at position: " position)
                        position = position + 1

                END IF

        END WHILE

END PROGRAM
```

Binary Searching

The efficiency of this algorithm?

Provide the list is sorted:

Binary searching is a more efficient searching algorithm for finding an element within a sorted list.

This works by repeatedly comparing the middle element of the list with the target value, and if it is not equal, the list is divided in half, removing the portion of the list the element will not be present in. The portion that could contain the target is amended as the new list. The process repeats until the possible location is narrowed down to just one and this is compared to the target value. If item is found before this the iteration stops.

This is more efficient as vast numbers of elements do not require comparing so fewer iterations are needed to locate the value. Not all elements are sequentially checked.

However, the list must be ordered either ascending or descending otherwise it is very unlikely the target value will be located. It functions on conditional statements that determine the lower and upper portions of the list relative to a mid-point.

It is not effective to use a sorting algorithm and then apply a binary search. Conversely, an unordered list is best to search with linear searching only.

Unless the list is to be searched many times, sorting it once then using a binary search can be effective. Sorting such as bubble sort requires many iterations to shift elements into an ordered position.

The pseudocode:

```
PROGRAM START

        List01 = [0,5,8,12,22,45,60,70,333,888,8889,10000]
        search_item = 333
        upper = length(List01) - 1
        lower = 0
        found = FALSE


        WHILE (lower <= upper) AND (found = FALSE)
                mid =  int((lower + upper) DIV 2)

                IF List01[mid] = search_item THEN
                        found = TRUE
                ELSE List01[mid] > search_item THEN
                        upper = mid -1
                ELSE:
                        lower = mid + 1
                END IF
        END WHILE


        IF found = TRUE THEN
                Print("Item is at index position: ", position)
        ELSE:
                print("Item not in list")
        END IF

END PROGRAM
```

Built in python functions for searching & sorting

- **Mylist.sort() OR Mylist.reverse()**
  - **If item in Mylist:**
  - **Mylist.index(item)**

Bubble Sorting

Bubble sorting is a method of sorting items in list of the same data type into ascending or descending order. Every item in the list is checked with the item next to it, if the item next to it is of a lower value then the two items swap positions.

This means that after only one iteration the highest number in the list will be the last item in the list. However, the algorithm must continue to check through the list since this will not fully sort the list, there may be a higher number to the left or several places back of an element as this comparison was not made. The algorithm must iterate n times where n comparisons are made. In total there are n^2 iterations unless efficiency is improved.

n being the list length

Bubble sort derives its name from larger numbers moving up the list like bubbles floating to the top of a glass.

After just one iteration, the highest element in the list will be shifted along to the last position in the list. This means for subsequent iterations we don't have to iterate n number of times as we can do n – 1 , -2, -3, -4 so on..

The pseudocode:

```
PROGRAM START

        List01 = [77,4,8,44,22,4,0,9,7,10]

        complete = FALSE

        WHILE complete = FALSE:

                complete = TRUE

                FOR i in range (0, length(List01) – 2):

                        IF List01[i] > List01[i +1]     THEN
                                temp = List01[i]
                                List01[i] = List01[i + 1]
                                List01[i+1] = temp

                                complete = FALSE
                        END IF
                NEXT i
        END WHILE


        print("SORTED LIST: ", List01)

END PROGRAM
```

We have up to length of list – 1 times as the list begins at position 0 and goes UPTO the upper bound. We have further changed the -1 to a -2 since in bubble sort we don't have to iterate through the list length of times as just after one iteration the highest item will be at the end of the list and the last item can't be compared with any item in front of it.

Insertion Sorting

A method of sorting a list of items of the same data type by: inserting/moving one item at a time, into the correct position, until all of the items in the list have been checked and a sorted list is built up.

The list is considered as two portions: an unsorted portion and a sorted portion. Elements are moved to the correct position in the sorted portion one at a time by meeting conditions.

The pseudocode:

```
PROGRAM START

        List01 = [9,6,5,8]

        FOR index in range (1, length(list01)):
                current_item = List01[index]
                position = index


                WHILE position > 0 AND current_item < List01[position - 1]:

                        List01[position] = List01[position - 1]
                        position =position – 1

                END WHILE

                List01[position] = current_item

        NEXT INDEX


        Print(“The sorted list is:, “ List01)

END PROGRAM
```