Searching Algorithms

Algorithms

An algorithm is a step by step procedure to solve a problem.

A successful Algorithm must

- Successfully solve the problem
- Be efficient - solve the problem in the least possible time.

It also should be finite i.e. It should stop when it is done.

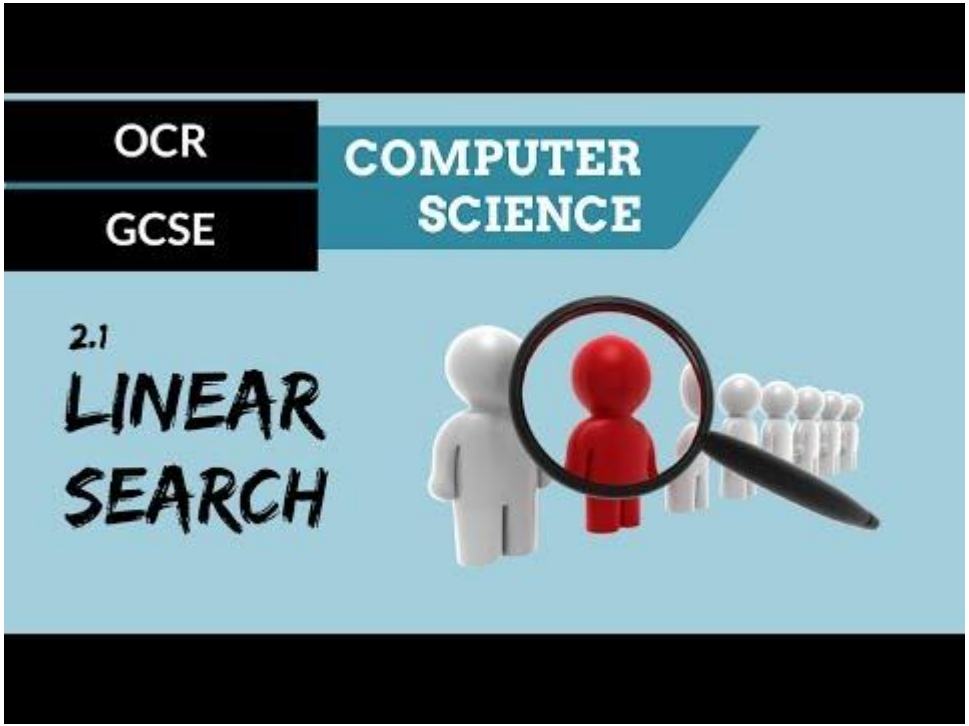
Searching Algorithms

There are many ways to search a list.

The two searching algorithms we will look at are:

- Linear search
- Binary search

Linear Search



https://www.youtube.com/watch?v=mce2XxIVkVU

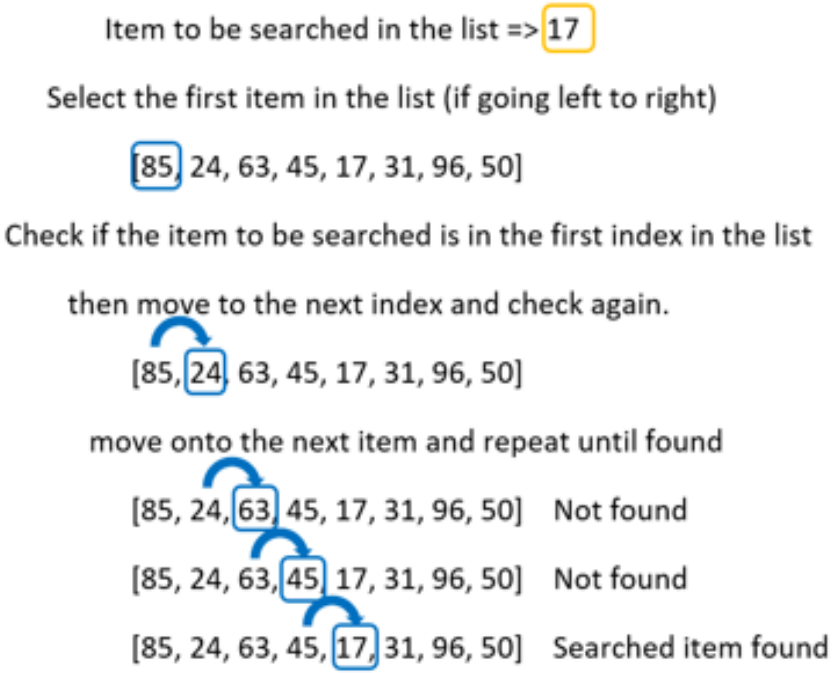
Linear Search

A linear search is an algorithm that iterates through each item in the list, comparing it to the item to be searched for.

First the algorithm starts at the first index (zero).

The algorithm tells the user the location (index) of the item in the list.

In this case 17 has an index of 4.



Linear Search

We can write this in Python (you can run the code via the link below).

```
myList = [85, 24, 63, 45, 17, 31, 96, 50]

search = 17
for index in range(len(myList)):
    if myList[index] == search:
        location = index

print(location)
```

Output:

```
4
```

https://repl.it/@suzannelinnane/Linear-Search-1

Linear Search

This works but it does not take into account some unpredicted outcomes.

For example, if the item to be searched is present more than once in the list or the algorithm does not find the item to be searched.

Linear Search - item present more than once

Let’s look at a list where the item is present more than once

```
myList = [85, 24, 63, 45, 17, 31, 96, 17]
```

If we ran the python code with this list then the output would be 7.

The index printed is index 7, the last item in the list.

This is because the algorithm continues through the list replacing the index value each time the item to be searched for is found.

Linear Search - item present more than once

The following code would return the lowest index of the item to be searched.

This is because we have reversed the direction that the list is searched.

It still technically returns the last index, but as the linear search direction is reversed, this is the index with the smallest value: 4

```
myList = [85, 24, 63, 45, 17, 31, 96, 17]

search = 17
for index in range(len(myList)-1, -1, -1):
        if myList[index] == search:
            location = index

print(location)
```

Output:

```
4
```

Linear Search - item not found

What would the output be if the searched item had a value of 117?

You can try it using the same code as before (in the Repl link) but change the search to 117.

```
myList = [85, 24, 63, 45, 17, 31, 96, 50]

search = 117
for index in range(len(myList)):
        if myList[index] == search:
                location = index

print(location)
```

Linear Search - item not found

The reason for the error is that the conditional statement (if myList[index] == search) was never entered because the item to be searched for was not present in the list. Therefore the location variable was never created.

In many other programming languages, instead of an error, you get an index of -1. This is because the lowest index in a list is 0, so -1 represents an index that does not exist.

Linear Search

The following code solves our issues by using return to return the first index of the searched item as soon as it is found.

If the item to be searched is not present in the list, and the loop has finished, the function returns -1.

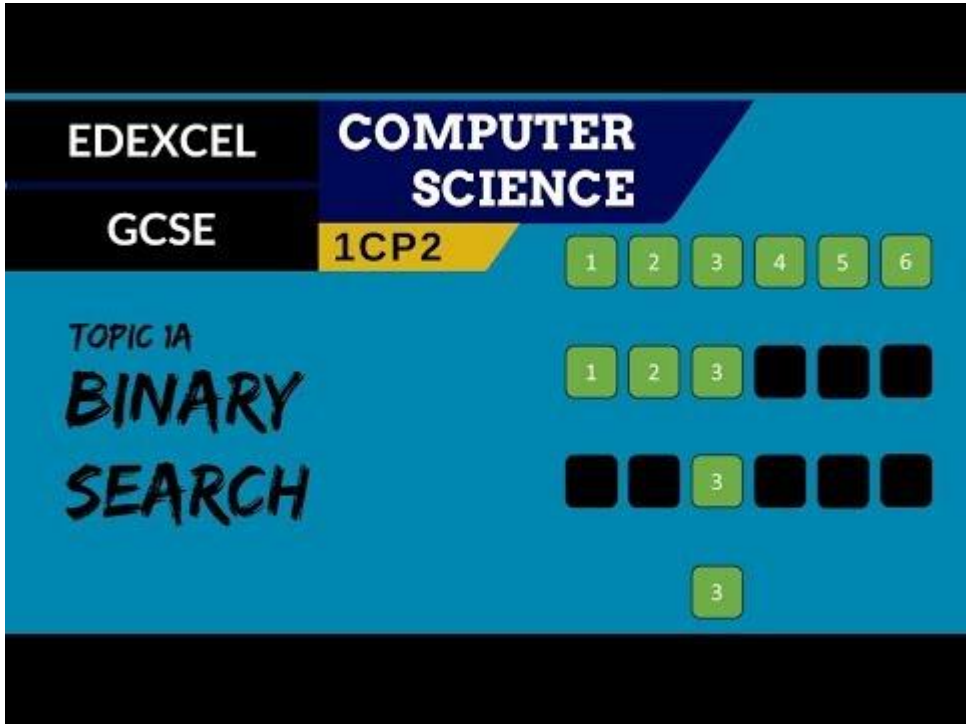
```
myList = [85, 24, 63, 45, 17, 31, 96, 50]

def linearSearch(listIn, searchItem):
    for index in range(len(listIn)):
        if listIn[index] == searchItem:
            return index
    return -1

print(linearSearch(myList, 17))
```

Output:

```
4
```

Try the code: https://repl.it/@suzannelinnane/Linear-Search-2

Binary Search



https://www.youtube.com/watch?v=8RPkQvkBBnQ

Binary Search

A binary search firstly assumes that the list is already sorted.

We can use the Python inbuilt sort() function to do this.

```
myList = [85, 24, 63, 45, 17, 31, 96, 50]

myList.sort()

print(myList)
```

Output:

```
[17, 24, 31, 45, 50, 63, 85, 96]
```

Binary Search

Once the list is sorted, we will look at the algorithm.

The binary search is often referred to as the half-interval search.

The algorithm employed is similar to how you might search through a deck of sorted cards. If you were searching for an 8 for example, you would select the middle card. If you card, 8 was the middle card, you found it. Otherwise, if your card was less than the middle card, you would focus your next search on the set of cards lower than the middle card. The same would apply if your card was larger than the middle card, you would focus on the cards greater than the middle card. This is repeated until either your card is found or it’s not in the list. This is how the binary search works.

Binary Search

When writing this program in python, there are several things we need to keep track of.

- First and last index of where we are looking (to start off this will be the first and last index of the list). These will change as we focus on different parts of the list.
- Half interval index (middle index between first and last index).

We start by getting the First and Last index:

[17, 24, 31, 45, 50, 63, 85, 96]

From here we can calculate the half-interval index, (note: Python floor division rounds down).

[17, 24, 31, 45, 50, 63, 85, 96]

Then we can check if the item to be searched is equal to, less than or greater to the half-interval value. If the item to be searched is equal to the interval value, we return this index. If it is greater or less then the item to be searched, we reset the First or last Last to focus on the new subsection of the list and repeat.

Item to be searched in the list => 17

Set the First to index 0 of the list, and Last to the last index of the list.

[17, 24, 31, 45, 50, 63, 85, 96]

Calculate the half-interval of the list => (Last – First) // 2

[17, 24, 31, 45, 50, 63, 85, 96]

Check if the item to be searched equals the half-interval value, if so return this index, else check if the item to be searched is less or greater than the half-interval.

If less than, set Last to the half-interval index -1.

If greater than, set First to the half-interval index + 1

Repeat using the new range of First and Last index's until Last – First is less than 0, next is the example completed:

[17, 24, 31, 45, 50, 63, 85, 96] = Initial Step, 17 < half-interval

[17, 24, 31, 45, 50, 63, 85, 96] = Set Last to half-interval Index -1

[17, 24, 31, 45, 50, 63, 85, 96] = Set half-interval (Last-First) // 2

[17, 24, 31, 45, 50, 63, 85, 96] = Check if 17 == half-interval (False)

[17, 24, 31, 45, 50, 63, 85, 96] = Set Last to half-interval Index -1 (index 0)

[17, 24, 31, 45, 50, 63, 85, 96] = Set half-interval (Last-First) // 2 (index 0)

[17, 24, 31, 45, 50, 63, 85, 96] = Check if 17 == half-interval (True) Return 0.

Binary Search - Python Code

```
myList = [17, 24, 31, 45, 50, 63, 85, 96]

def binarySearchLoop(listIn, searchItem):
    first = 0
    last = len(listIn) - 1
    while (last - first) >= 0:
        middle = first + ((last - first) // 2)
        if listIn[middle] == searchItem:
            return middle
        elif searchItem < listIn[middle]:
            last = middle - 1
        else:
            first = middle + 1
    return -1

print(binarySearchLoop(myList, 17))
```

Output:

```
0
```

Try the code: https://repl.it/@suzannelinnane/Binary-Search

Searching Algorithms

1. **Linear Search**

- Straightforward and easy to understand
- Data does not have to be sorted
- Only suitable for very small datasets

1. **Binary Search**

- Data must be sorted
- More efficient than Linear Search on large datasets

Example: Step through how the binary search algorithm would find 99 in the following list.

7, 21, 52 ,59 ,68 ,92 ,94 ,99 ,133

Example: Step through how the binary search algorithm would find **cherry** in the following list.

apple cherry fig grape kale mango pear tomato

Example: Step through how the linear search algorithm would find **13** in the following list.

2 3 7 5 13 11