Introduction

- We have two sorting and two searching algorithms in A-levels:
- Searching algorithms:
    - Linear Search `O(n)`
    - Binary Search `O(log(n))`
- Sorting algorithms:
    - Bubble Sort `O(n^2)`
    - Insertion Sort `O(n^2)`

Searching

> Finding element in a collection. (if it exists)

Linear Search

> Linear Search compares item_to_find with each item in collection.

- `O(n)` time complexity means that time taken by linear search is proportional(roughly) to number of items in collection.
- Algorithm:

```
def linear_search(data, item_to_find):
    for i in range(len(data)):
        if item_to_find = data[i]:
            return i
    return -1
```

> Compare with each item, return index if found and -1 otherwise.

- Using the function above

```
numbers = [1,2,3,7,5,4]
print(linear_search(numbers,2))
print(linear_search(numbers,10))
```

Binary Search

> Binary Search divides the search space to half with each iteration.

bishaltwr@gmail.com

- `O(log(n))` means as you increase search space by factor of 2, time taken only increases linearly.
- Algorithm:

```
def binary_search(data, item_to_find):
    start = 0
    end = len(data) - 1
    mid = (low+high)//2 # or (low+high+1)//2
    while low<high:
        if item_to_find = data[mid]:
            return mid
        elif item_to_find > data[mid]:
            start = mid + 1 # start search from values after mid
        else:
            end = mid - 1 # Only search values before mid
    return -1 # If the loop completes
```

> Search at middle position, and limit search space to values after or before middle based on whether item_to_find is greater than or less than mid item.

- Using the function above

```
numbers = [1,2,3,4,5,6,7,8,9]
print(binary_search(numbers,2))
print(binary_search(numbers,10))
```

> Binary search can only be used for sorted data. The code above works for data sorted in ascending order and needs to be adjusted for data in descending order.

Recursive Binary Search

> Binary Search can also be implemented recursively. It also halves the search space with each recursive call.

```
def binary_search_recursive(arr, item_to_find, low, high):
    if low > high:
        return -1  # Item not found

    mid = (low + high) // 2

    if arr[mid] == item_to_find:
        return mid
    elif arr[mid] > item_to_find:
        return binary_search_recursive(arr, item_to_find, low, mid - 1)
    else:
        return binary_search_recursive(arr, item_to_find, mid + 1, high)
```

- Use the function like this

```
arr = [i for i in range(100)]
print(binary_search_recursive(arr, 20, 0, 99))
print(binary_search_recursive(arr, 105, 0, 99))
```

Sorting

> Sorting is the process of ordering elements based on a definite rule. For eg. Ascending, Descending, Chronological, Based on length etc.

Bubble Sort

> Compares adjacent items, swapping if necessary. Each time this is done, the largest unsorted element bubbles up to its correct position. This process is repeated until the list is sorted.

- Algorithm

```
def bubble_sort(data):
    n = len(data) - 1 # Last item gets sorted on its own
    for i in range(n):
        for j in range(n-i):
            if data[j] > data[j+1]:
                temp = data[j]
                data[j] = data[j+1]
                data[j+1] = temp
```

- This can be optimised:

```
def bubble_sort(data):
    n = len(data) - 1 # Last item gets sorted on its own
    no_swap = True
    for i in range(n):
        for j in range(n-i):
            if data[j] > data[j+1]:
                no_swap = False
                temp = data[j]
                data[j] = data[j+1]
                data[j+1] = temp
        if no_swap: # If there is no swap in a loop, you are done.
            return
```

Insertion sort

> Takes a elment and inserts it at its correct position among elements before it.

- Algorithm:

```
def insertion_sort(arr):
    for i in range(1,len(arr)):
        item = arr[i]
        j = i - 1
        while(j>-1 and arr[j]> item):
            arr[j+1] = arr[j]
            j = j - 1
        arr[j+1] = item
```

- Using the function:

```
arr = [1,5,4,3,2]
insertion_sort(arr)
print(arr)
```
