3

003 - Searching and Sorting

SEARCHING:

Searching is the process of finding a particular piece of data within a data structure.

Linear Search:

The linear search algorithm is given a list of values and a key. It returns the first index of where the key is found or -1 if the key is not part of the list. We use -1 to indicate the state of not finding the value because 0 is a perfectly valid index into the list.

The code for this is pretty straight forward. Start at the beginning of the list and search until you either find the item or you reach the end of the list.

Python Copy Caption

```python
def linear_search(my_list, key):
    for i in range(0, len(my_list)):
        if my_list[i] == key:
            return i

    return -1
```

- **Time Complexity:** O(n)
- Linear Time Complexity

Binary Search:

The binary search algorithm goes like this: *Track the range of possible indexes by storing the first/last index where key may be found* initially this is 0 and (length of list) - 1. *Calculate the mid point index between those first/last indexes* look at the value at the middle element *if we found it we are done* if the key is smaller than middle element, then key can only be found between first index and element before middle element. Thus, set last index to middle index - 1. * if key is greater than middle element then key can only be found after that middle element, thus set first index to middle index + 1.

```python
def binarySearch(arr, key):
    # Calculate the start and end index of the array
    start = 0
    end = len(arr) - 1

    # start should always be less than the end
    while (start <= end):
        # Calculate the middle index
        middle = start + ((int)((end - start) / 2))
        # Check whether the element in the middle is the kwy
        if arr[middle] == key:
            return middle
        # Check whether middle element is greater than key
        elif arr[middle] > key:
            # The element has to be towards the left
            end = middle - 1
        # Check whether the middle element is smaller than the key
        elif arr[middle] < key:
            # The element has to be on the right of the middle
            start = middle + 1
    return -1

print(binarySearch([100, 200, 300, 400, 500, 600, 700, 800], 700))
```

- **To perform binary search:**
  - The list must be sorted. This allows us to split the array into two pieces, one where the key might be found and the other where the key definitely can't be found.
  - The second requirement is the list must allow fast random access (ie be array-like). That is getting to any element of the list takes the same amount of time. In the list from the C++ standard library this is not the case.