Computer Programming I

Searching Algorithms

Introduction to Searching

- How can we tell if a given element is in a list of values?
- Are there some ways of answering this question better than others?
- When performing a search, the element we are searching for is called a search key or simply the key.
- A good search algorithm must yield either:
  - The location of the key, if it's found, or
  - A special value to indicate that it's not found.

Introduction to Searching

- For a list, the location is typically the index.
  - For example, searching for 45 (in the list below) should return 5
- The special value returned when we don't find the value is usually -1 since that cannot be a legal index.
  - For example, searching for 100 (in the list below) should return -1

0	1	2	3	4	5	6	7	8
20	35	37	40	45	50	51	55	67

Sequential Search

- In sequential search, we simply start at the beginning of the list, and check each element in order until the key is found, or the end of the list is reached.
- Sequential Search is also called Linear Search
- https://www.youtube.com/watch?v=-PuqKbu9K3U

Sequential Search Example

3	6	7	10	4	12	9	5	8

1.) let's find 5 with linear search algorithm

Linear Search Implementation

```
def linear_search(values, key):
    for index in range(len(values)):
        if values[index] == key:  #found, return key location
            return index
    return -1   # not found
```

Linear Search Implementation

```
def linear_search_V2 (values, key) :
    found = False
    index = 0
    foundIndex = -1
    while (found == False and index < len(values)) :
        if values[index] == key : #key found
            found = True
            foundIndex = index
        else :
            index = index + 1

    return foundIndex
```

Linear Search Efficiency

- An algorithm typically uses a number of steps proportional to the size of the input.
- For a list with 32 elements, linear search requires at most 32 comparisons:
  - 1 comparison if the search key is found at index 0,
  - 2 if found at index 1,
  - and so on,
  - up to 32 comparisons if the search key is not found.
- For a list with N elements, linear search requires at most N comparisons.
- The algorithm is said to require "on the order" of N comparisons.
  - The average number of comparisons would be **N/2**

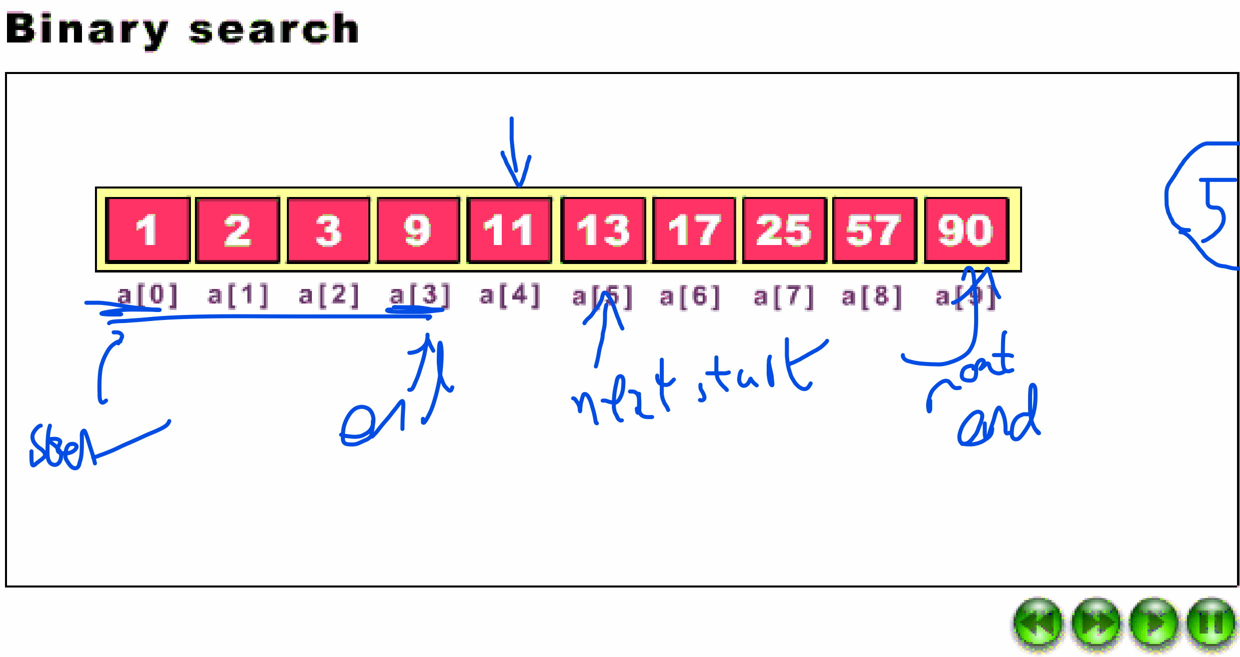
Binary Search

- Binary search is a “divide and conquer” algorithm.
- It works only on sorted lists.
  - But it is much faster than linear search

Binary Search

- First check the middle element:
  - If this element matches our key, we are done.
  - Otherwise, repeat the search on the:
    - Left sub-list if the key was less than the middle element
    - Right sub-list if the key was greater than the middle element
- https://www.youtube.com/watch?v=iP897Z5Nerk

Binary Search Example



Binary Search Implementation

```
def binary_search(values, key):
    start = 0
    end = len(values) - 1

    while end >= start:
        mid = (end + start) // 2  #index of the middle value
        if values[mid] < key:  #search on the right
            start = mid + 1
        elif values[mid] > key: #search on the left
            end = mid - 1
        else:     #found
            return mid

    return -1 # not found
```

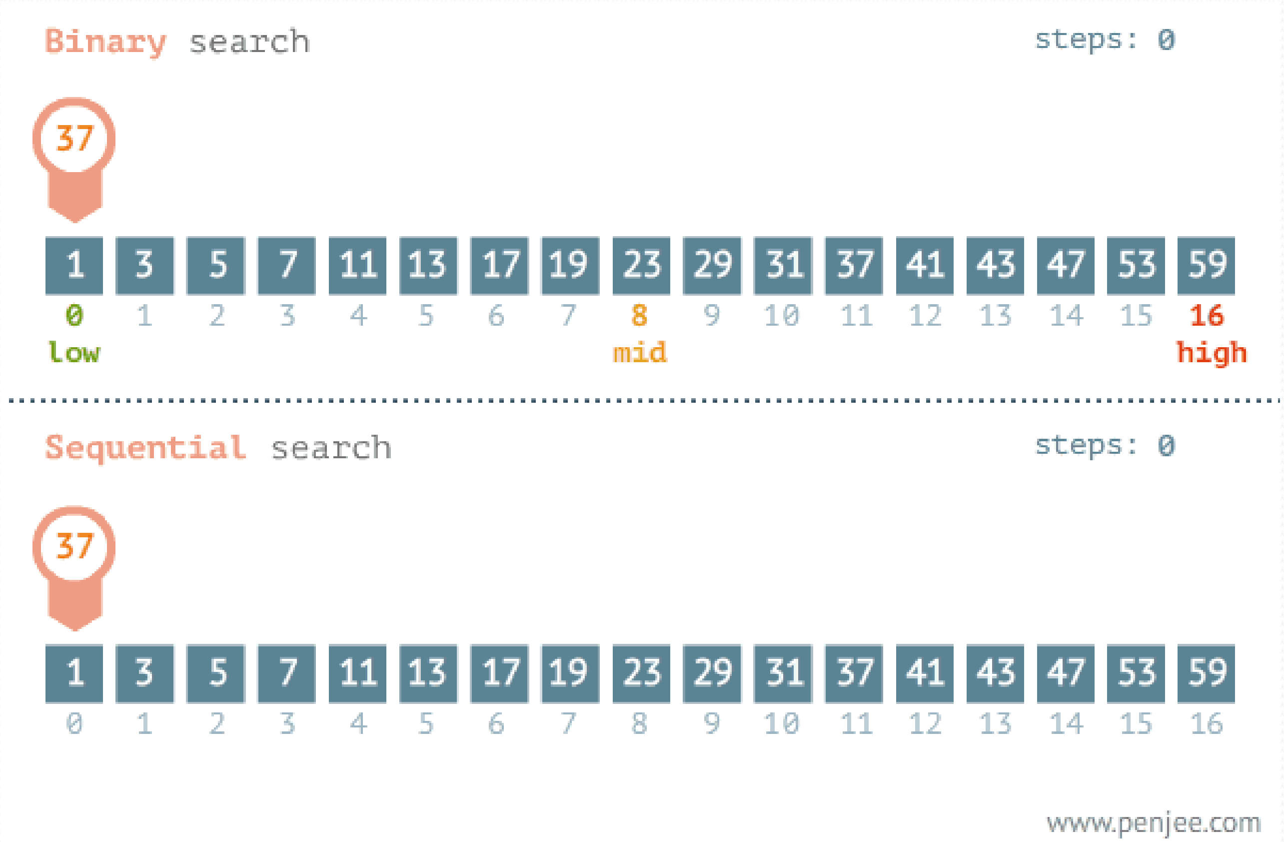
Binary Search Implementation

```
def binary_search_V2(values, start, end, key) :
    if start > end :    # Not Found!
        return -1

    mid = (start + end) // 2    # index of middle value

    if  values[mid] == key   :   # Found it!
        return mid
    elif values[mid] < key :    # key is in the upper half
        return binary_search_V2(values, mid + 1, end, key)
    else :                      # key is in the lower half
        return binary_search_V2(values, start, mid - 1, key);
```

Binary Search Efficiency

- Binary search is incredibly efficient in finding an element within a sorted list.
  - During each iteration of the algorithm, binary search reduces the search space by half.
- The search terminates when the element is found, or the search space is empty (element not found).
- For a 32-element list, if the search key is not found, the search space is halved to have 16 elements, then 8, 4, 2, 1, and finally none, requiring only 6 steps.
- For an N element list, the maximum number of steps required to reduce the search space to an empty sub-list is $\lfloor \log_2 N \rfloor + 1$. and the average number of comparisons would be $\log_2 N$

Linear Search vs. Binary Search



Only with numbers?

Absolutely not!

Search for the word: said