Algorithms - Binary Search

Advanced Higher Computing Science

Learning Objectives

By the end of this lesson you will be able to:

- ❏ Describe, exemplify, and implement a standard binary search algorithm

Linear Search using conditional loop

Will stop searching if it finds an item or reaches the end of the list

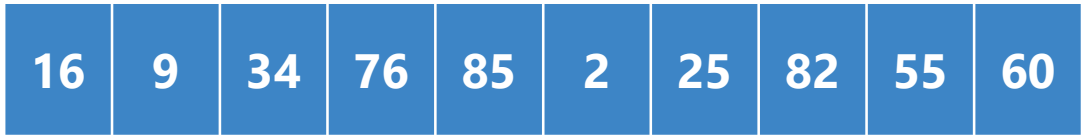
```
def FindName (names):

    choice = input("Please enter name: ")
    counter = 0
    matchindex  = 0
    found = False

    while counter < len(names) and found == False:
        if names[counter] == choice:
            found = True
            matchindex = counter

        counter += 1

    if found == True:
        print("Name found at position:",matchindex)
    else:
        print("No item found")



#-------------Main Program----------------
#set initial values
names = ["Joanna","Mike","Sarah","Mark","Matthew"]

FindName(names)
```

How many times

Assume your search goal is **76**.

The search length would be **4**

We are already looking at the first item



Analysis of the Linear Search

1 comparison is required to access the 1st item
2 comparisons are required to access the 2nd item
N comparisons are required to access the last item.

The average search length for N items is given by the expression N/2

- Where N is the number of items in the list
- (1+2+3+……+N)/N = N/2

Binary Search

A Binary Search algorithm works by splitting the list in two until the item is found:

Like when looking at a real dictionary:

- You find a page - if the current word is smaller (earlier in the alphabet) you pick another page **after** your current point
- if the current word is smaller (later in the alphabet) you pick another page **before** your current point

- Divide and conquer!
- Generally faster than linear search
- Main drawback is the list has to be sorted first (will look at sorting later)

Binary Search Pseudocode

```
SET found = FALSE
RECEIVE Goal FROM (INTEGER) KEYBOARD
REPEAT
        SET middle = INTEGER((startpos+ endpos)/2)
        IF array(middle)  = goal THEN
                SET Found = TRUE
        ELSE IF array(middle) < goal THEN
                SET startpos = middle + 1
        ELSE SET endpos = middle - 1
UNTIL found = TRUE OR (startpos>endpos)
```

Python Implementation

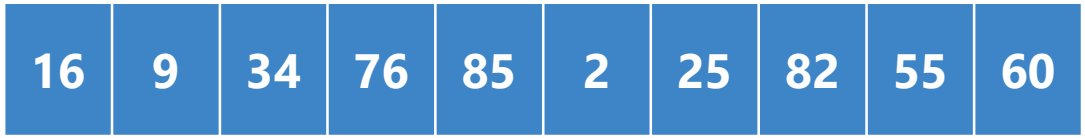
```
found = False
startpos = 0
endpos = len(searchlist) -1

print ("Endpos at beginning = ",endpos)

while (startpos <= endpos) and found == False:
    middle = (startpos+endpos)//2 #Integer Division
    if searchlist[middle] == goal:
        found = True
    elif searchlist[middle]<goal:
        startpos = middle + 1
    else:
        endpos = middle -1
```

How many times

Assume your search goal is **76**.

The search length would be **4**

We are already looking at the first item



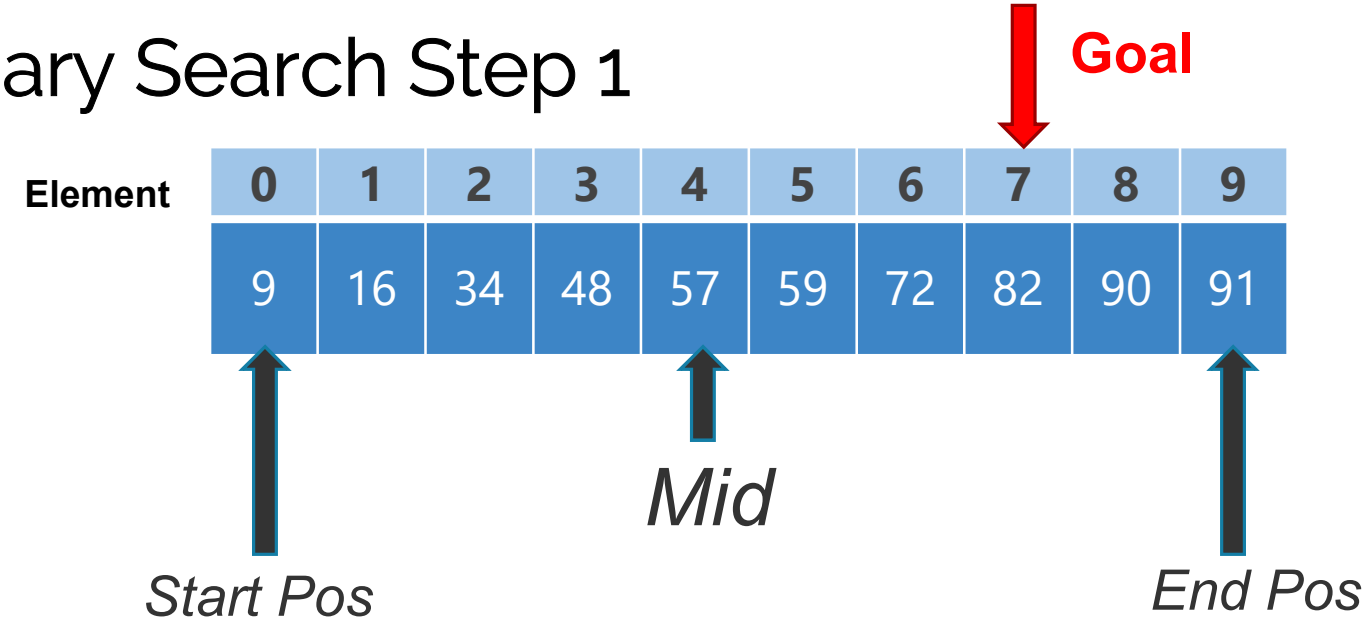
Binary Search Walkthrough

Element	0	1	2	3	4	5	6	7	8	9
	9	16	34	48	57	59	72	82	90	91

Assume our search goal is **82**.

Search length using **linear search = 8**

Binary Search Step 1

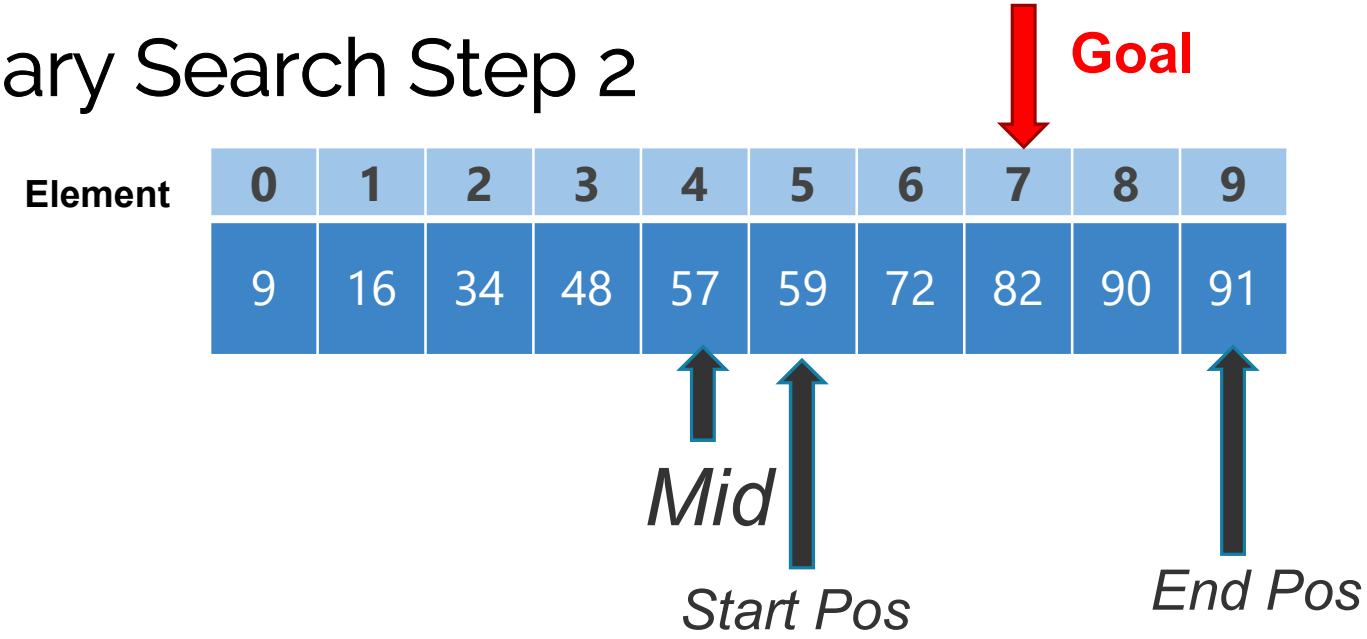


First step is to find the middle value of the list.

(start + end)/2 = (0 + 9)/2 = 4.5 = 4

This creates two lists (left and right of the Mid point)

Binary Search Step 2

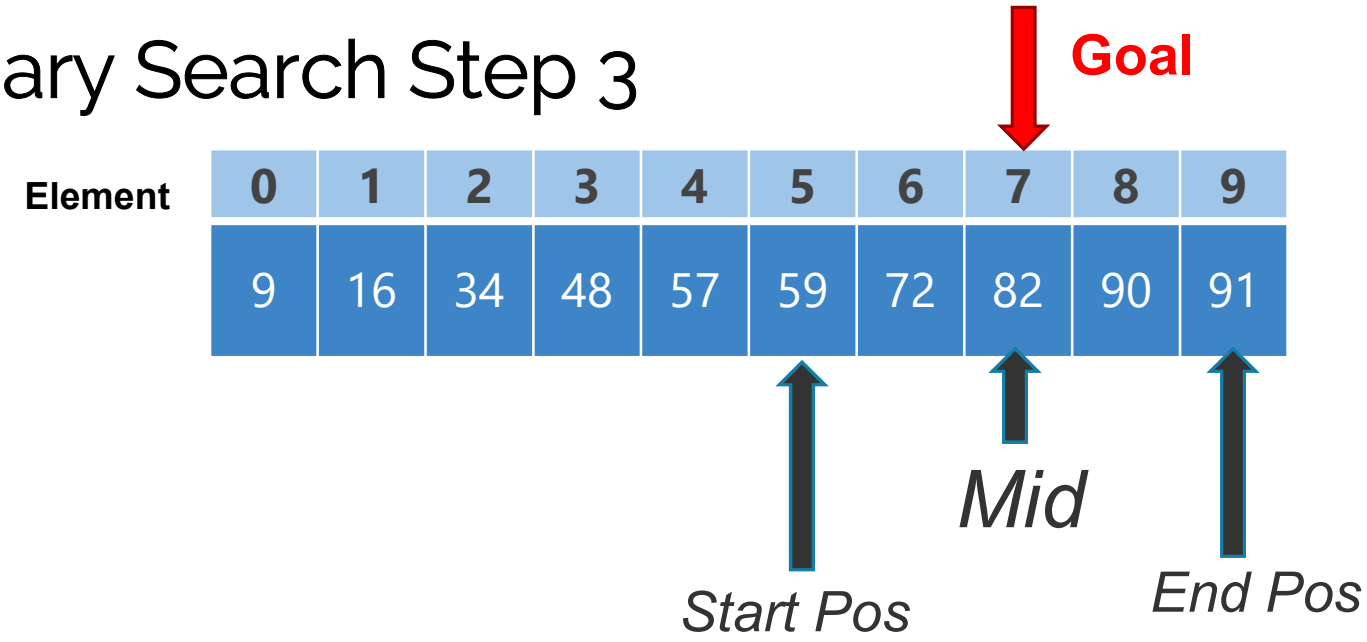


Now compare the mid value with the goal

57 is less than 82 so we can **ignore the left list**

We will adjust the **new Start Position to be one element above our current Mid Point**

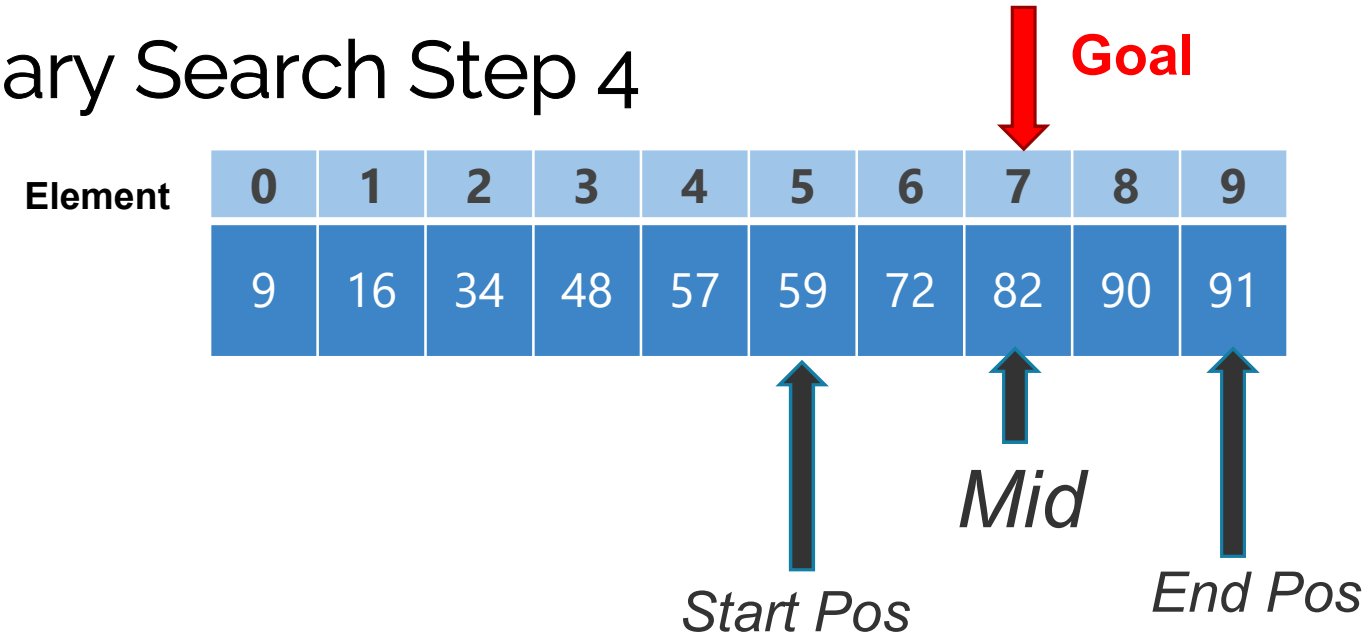
Binary Search Step 3



Now find the **new mid value**

Middle is between item 5 and item 9. So element 7(82) is the new **Mid**

Binary Search Step 4



Now find the **new mid value**

Create new mid location between location 6 and location 9.

This is location 7 (82).

Current Mid = Goal state so position = 7

What to do now

Implement a Binary Search using a hard coded list of 32 items

Test the amount of comparisons if the item is at:

- The centre of the list
- The extreme high and low ends

What is the average amount of comparisons?

Binary Search - Analysis

For a list of **N** data items the average search length would be $\mathbf{Log_2 N}$

So, for a list of 64 items, the average search length will be 6.

Problem with calculating the midpoint

There can be a potential problem when using the binary search on extremely large lists

When the low + high value exceeds the storage limits for an integer

https://research.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

To correct a possible overflow you can change the midpoint calculation to:

```
middle = low + ((high - low) / 2)
```

This will avoid any overflow

Comparison

Linear Search	Binary Search
Is simple to code and implement	Is more complex to code
Quite efficient for short length data lists	Efficient for any length of data lists
Very slow on large lists since each data element has to be compared	Fast on any length of data list since it only deals with half sub-lists.
Does not require data to be sorted	Data has to be sorted.
Average search length is **N/2** where N	Search length is $\mathbf{\log_2 N}$
Plays a part in other algorithms such as finding maximum, minimum and also in the selection sort	Binary chop is used in fast sorting routines