Scripting Language

UNIT-III

Searching

- Searching is a process of finding an object among a group of objects
- Here the element is also referred to as 'Key'
- Searching is one of the important application of Arrays i.e., we can search a particular element present in a list or in a record
- There are 2 important Searching Techniques

1. Linear Search [Sequential Search]
2. Binary Search [Logarithmic Search]

Searching Algorithms

Necessary components to search a list of data

- Array containing the list
- Length of the list
- Item for which you are searching

After search completed

- If item found, report "**success**," return location in array
- If item not found, report "**not found**" or "**failure**"

Searching Arrays

- Linear search
  - small arrays
  - unsorted arrays
- Binary search
  - large arrays
  - sorted arrays

Linear/Sequential searching

- It is the simplest of the searches
- It is simply searching through a line of objects in order to find a particular element
- This is similar to looking through a row of CD's to find our favorite
- This leads to the possibility of 2 variations of a linear search, the sorted LS and the Unsorted LS
- The linear search is used to find an item in a list. The items do not have to be in order. To search for an item, start at the beginning of the list and continue searching until either the end of the list is reached or the item is found.

Performance of the Linear Search

- Suppose that the first element in the array list contains the variable key, then we have performed **one comparison** to find the key.
- Suppose that the second element in the array list contains the variable key, then we have performed **two comparisons** to find the key.
- Carry on the same analysis till the key is contained in the last element of the array list. In this case, we have performed **N comparisons** (where N is the size of the array list) to find the key.
- Finally if the key is **NOT** in the array list, then we would have performed N comparisons and the key is NOT found and we would return -1.

Linear Search Algorithm

Step 1: Start at first element of array – Search Element.

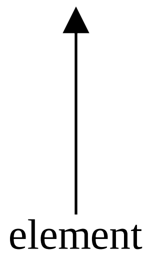
Step 2: Compare value to value (key) for which you are searching

Step 3: Continue with next element of the array until you find a match or reach the last element in the array. (iteration)

Step 4: Return Failure , if element not found in the list

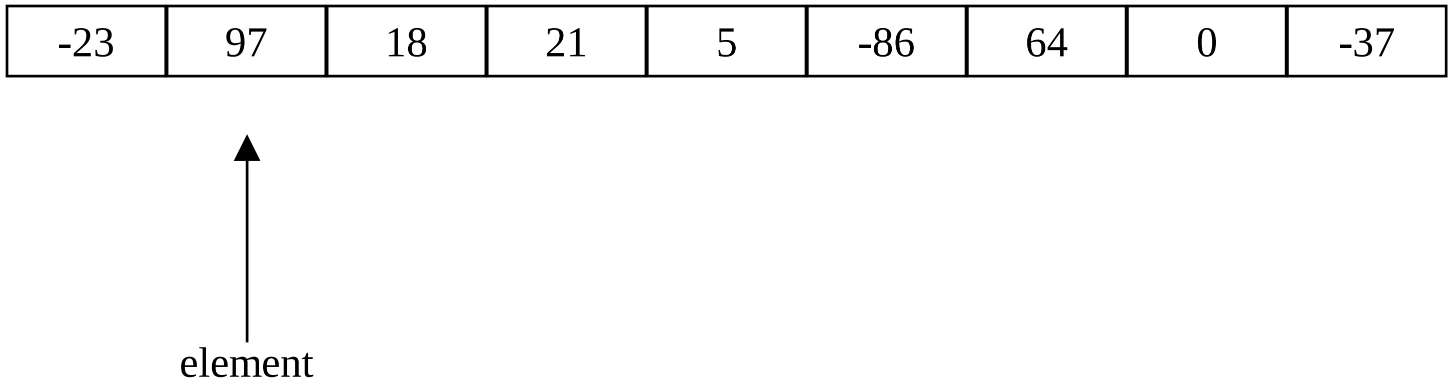
Linear Search Example #1

-23	97	18	21	5	-86	64	0	-37



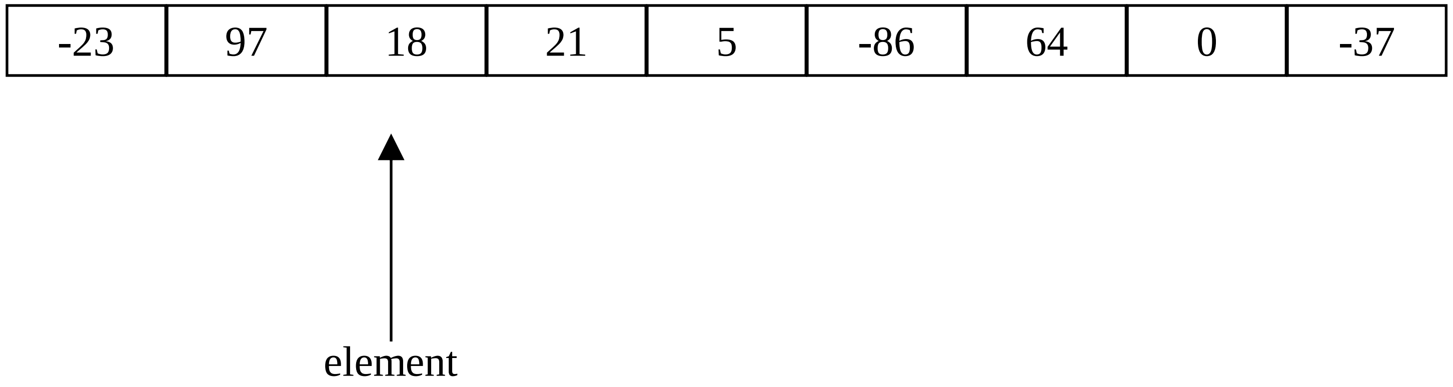
Searching for -86.

Linear Search Example #2



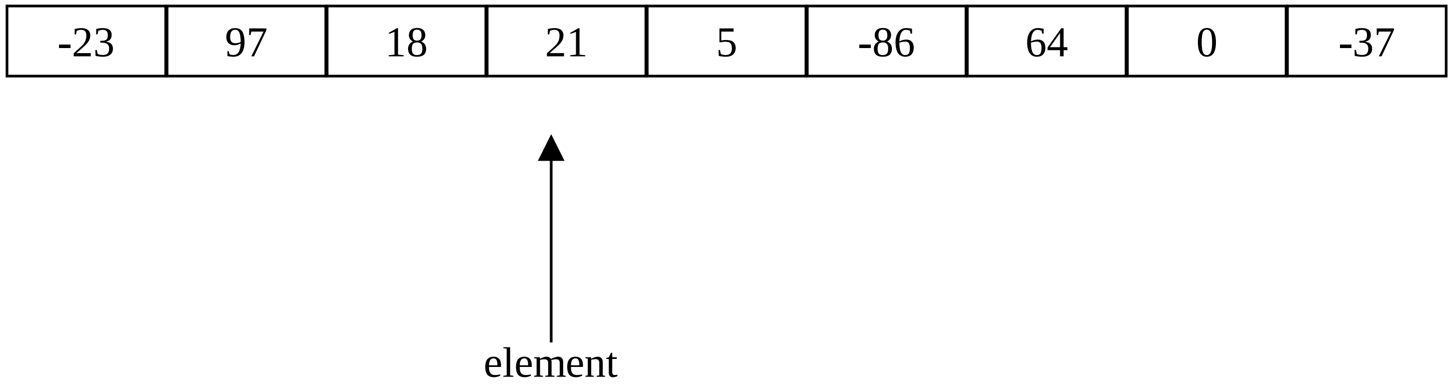
Searching for -86.

Linear Search Example #3



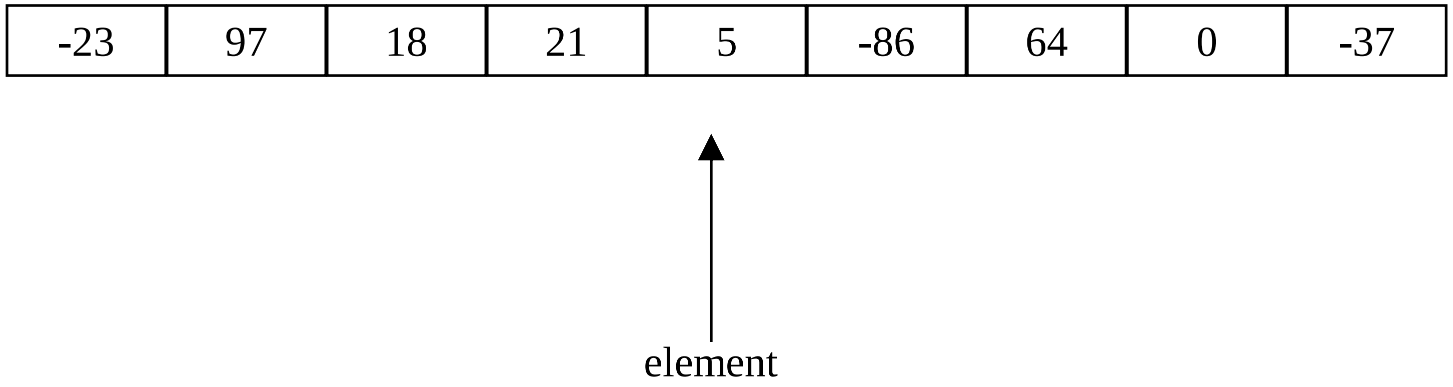
Searching for -86.

Linear Search Example #4



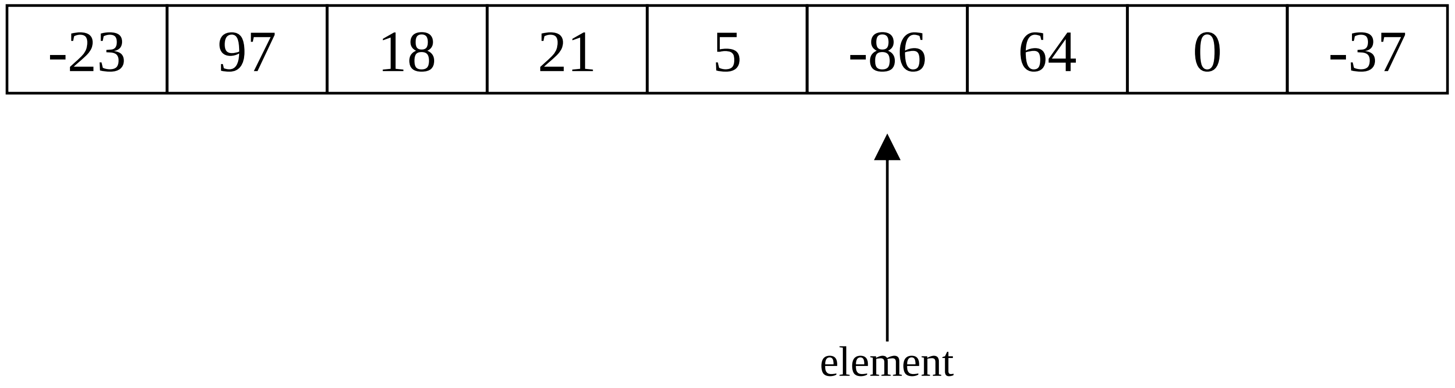
Searching for -86.

Linear Search Example #5



Searching for -86.

Linear Search Example #6



Searching for -86: found!

# Source Code

```
def linear_search(search_ele, list_ele):
    count=0
    for i in range(len(list_ele)):
        if list_ele[i] == search_ele: # item found, return the index value
            print ("Position of the searched element: ",i)
            count=count+1
    print ("count of repeated search elements are: ",count)

    if (count==0):
        print( "Element is not found.")
size = int(input("Enter a size of the array: "))                # size of random list
array=[]
for i in range(0,size):
              element=input("Enter a element: ")
              array.append(element)
print(array)          # show us what you've got

target=input("Enter a search element: ")
linear_search(target,array)
```

Efficiency of the Linear Search

The advantage is its simplicity.

- It is easy to understand
- Easy to implement
- Does not require the array to be in sorted order

The disadvantage is its inefficiency

- If there are 20,000 items in the array and what you are looking for is in the 19,999th element, you need to search through the entire list.

Complexity

- There exists 3 time complexities, namely worst, best & average cases
- If there are N ele.s in the list, then it is obvious that in the worst case i.e., when there is no target ele. in the list, N comparisons are required
- Hence W C TC =O(N) (Oh(N) or big-Oh (N))
- The best case, in which the 1st comparison returns a match, it requires a single comparison
- Hence B C TC =O(1)
- The average time depends on the prob. that the key will be found in the list. Thus the avg. case roughly requires N/2 comparision to search the element(i.e., depends on N)
- Hence A C TC=O(N)

Binary Search

Binary searching

- Search as the name suggests, is an operation of finding an item from the given collection of items. Binary Search algorithm is used to find the position of a specified value (an 'Input Key') given by the user in a sorted list.
- Binary Search can only be performed if the data is sorted, whether it is in form of a Array, list or any other structure
- This concept is generally used by Electricians to locate a fused bulb in a serial set & in searching a dictionary, phonebook
- The concept is splitting the list into two halves & then checking the middle item
- Binary Search is also referred to as a Logarithmic Search

Algorithm

Step 1: It starts with the middle element of the list.

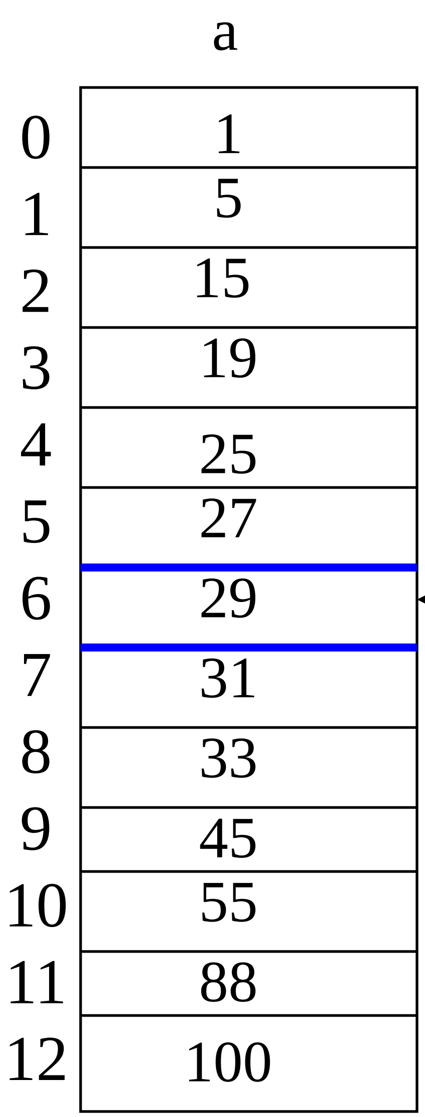
Step 2: If the middle element of the list is equal to the ‘input key’ then we have found the position the specified value.

Step 3: Else if the ‘input key’ is greater than the middle element then the ‘input key’ has to be present in the last half of the list.

Step 4: Or if the ‘input key’ is lesser than the middle element then the ‘input key’ has to be present in the first half of the list.

Step 5: Failure if element not found or empty

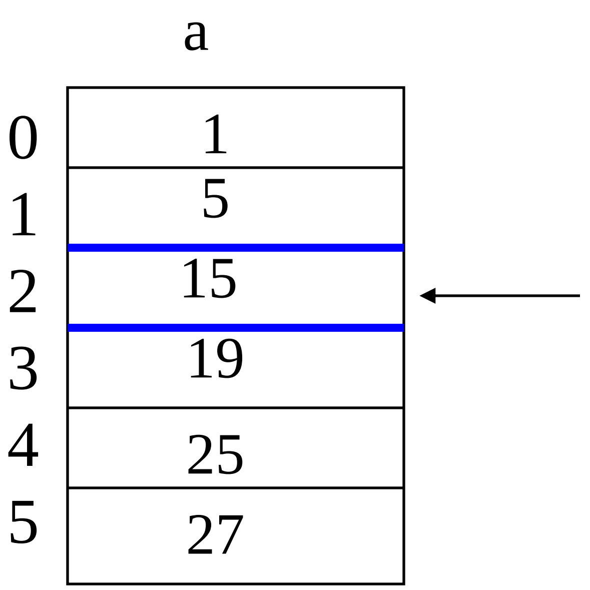
•Binary Search Example



search key = 19

middle of the array
compare a[6] and 19
19 is smaller than 29 so the next
search will use the lower half of the array

•Binary Search Pass 2



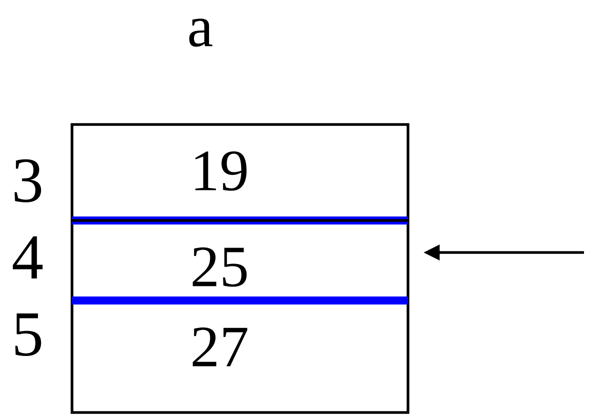
search key = 19

use this as the middle of the array
Compare a[2] with 19

15 is smaller than 19 so use the top half for the next pass

•Binary Search Pass 3

search key = 19

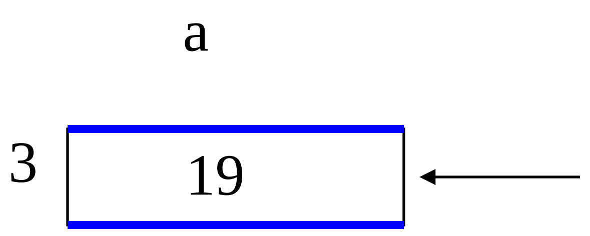


use this as the middle of the array
Compare a[4] with 19

25 is bigger than 19 so use the bottom half

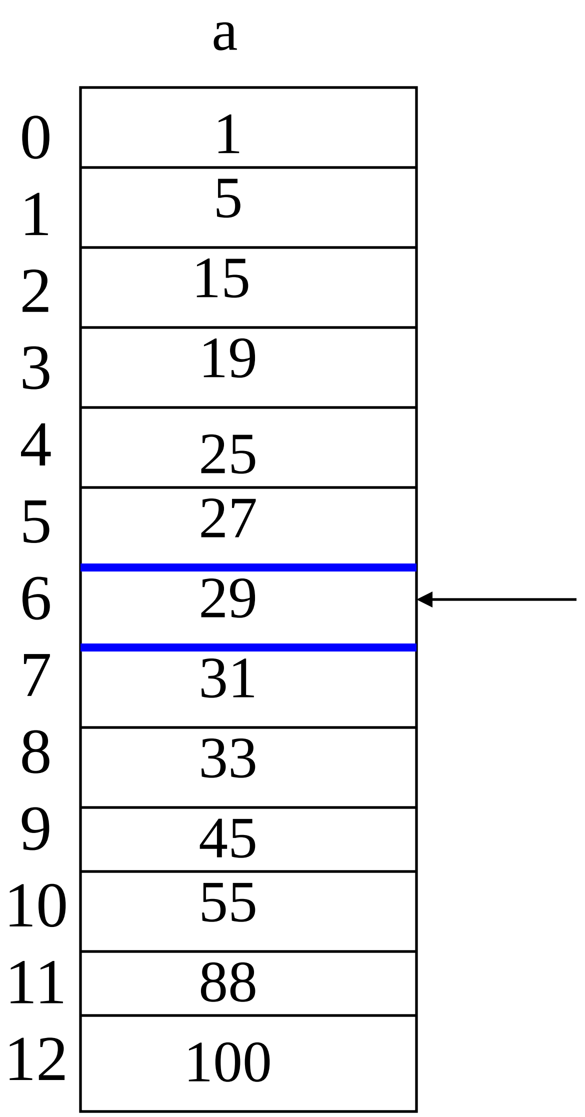
•Binary Search Pass 4

search key = 19



use this as the middle of the array
Compare a[3] with 19
Found!!

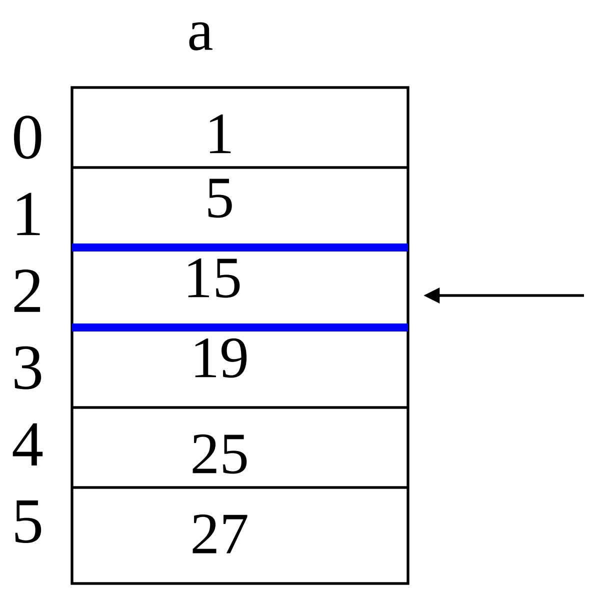
•Binary Search Example



search key = 18

middle of the array
compare a[6] and 18
18 is smaller than 29 so the next
search will use the lower half of the array

•Binary Search Pass 2



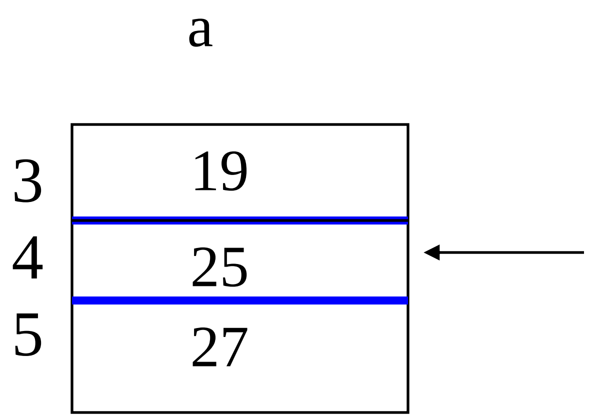
search key = 18

use this as the middle of the array
Compare a[2] with 18

15 is smaller than 18 so use the top half for the next pass

•Binary Search Pass 3

search key = 18

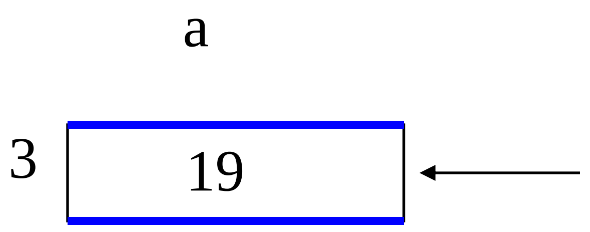


use this as the middle of the array
Compare a[4] with 18

25 is bigger than 18 so use the bottom half

•Binary Search Pass 4

search key = 18



use this as the middle of the array
Compare a[3] with 18
Does not match and no more elements to compare.
Not Found!!

# Source Code

```
def binary_search(seq, element):  # Function Definition
    min = 0
    max = len(seq) - 1
    while True:
        if max < min:           # Failure Case
            print("Element not found")
            return
        m = (min + max) // 2  # Find the mid position
        if seq[m]==element:
            return m
        elif seq[m] < element:   # Search on to the First Half [ Less than]
            min = m + 1
        elif seq[m] > element:   # Search on to the Last Half [ Greater than]
            max = m - 1
size = int(input("Enter a size of the array: "))
array=[]
for i in range(0,size):
                element=int(input("Enter a element: "))
                array.append(element)
array=sorted(array)
search = int(input("Enter a number to search: "))   # Search Element or Key
print ("The sorted elements of the given elements: ",(array))   # Elements after sorting
print ("The element",search,"is founded at position: ",(binary_search(array,search)))
```

Properties

- Best Case performance – The middle element is equal to the ‘input key’ O(1).
- Worst Case performance - The ‘input key’ is not present in the list O(logn).
- Average Case performance – The ‘input key’ is present, but it’s not the middle element O(logn).
- The List should be sorted for using Binary Search Algorithm.
- It is faster than Linear Search algorithm, and its performance increases in comparison to LS

IIIT BASARA

Infosys | Campus Connect

For Details Contact Me @ :
9247448766
ravikanth27787@gmail.com

python
Programming Language