

CLASS XII - CS

SEARCHING ALGORITHMS

S. VIGNESH M.E (CS & IS)

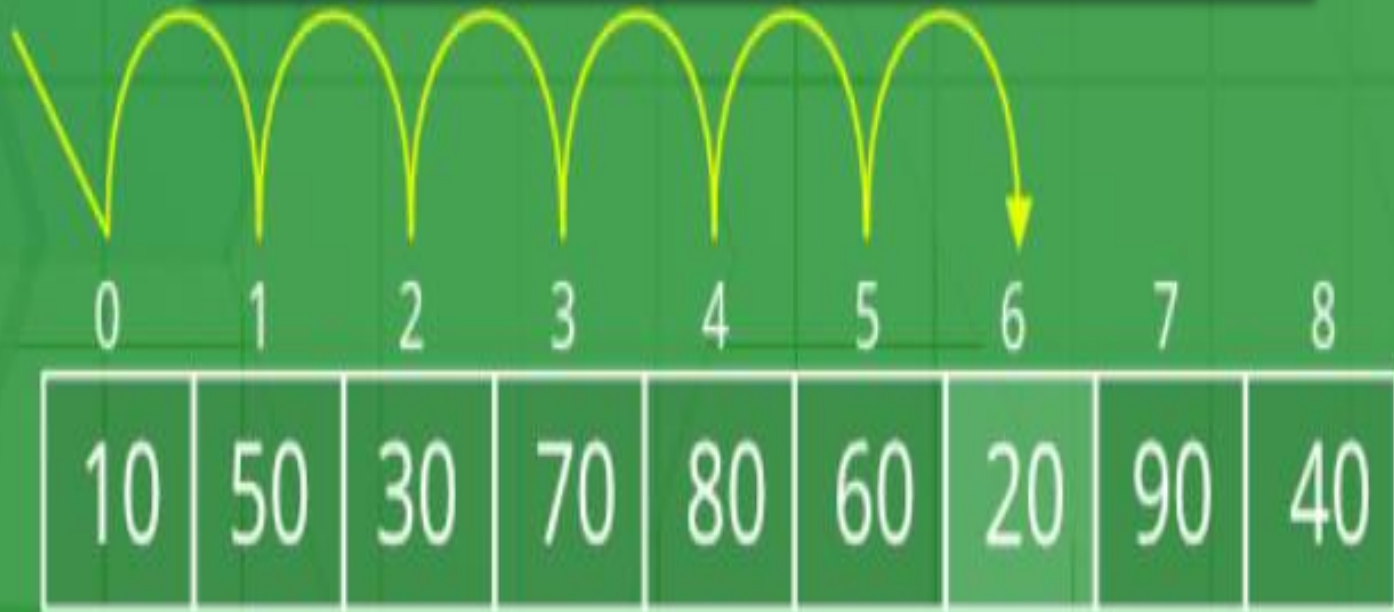
LINEAR SEARCH ALGORITHM

- ❖ A simple approach is to do **linear search**, i.e
- ❖ Start from the leftmost element of `arr[]` and one by one compare `x` with each element of `arr[]`
- ❖ If `x` matches with an element, return the index.
- ❖ If `x` doesn't match with any of elements, return -1.

Linear Search

SEARCH AT EVERY POSITION
TO FIND ELEMENT IS
FOUND OR NOT

Find '20'



```
def Linear_Search(list1, element_to_search):  
    for i in range(len(list1)):  
        if element_to_search == list1[i]:  
            print("ELEMENT", list1[i], "IS FOUND IN THE", i, "POSITION")  
        else:  
            print("ELEMENT", list1[i], "NOT FOUND")  
  
list1 = [] #def - empty list  
#read the no. of elements for list  
lt = int(input("ENTER THE NO. OF ELEMENTS"))  
for i in range(0, lt):  
    list1.append(int(input()))  
print("THE EXISTING LIST:", list1)  
element_to_search = int(input("ENTER THE ELEMENT TO SEARCH"))  
Linear_Search(list1, element_to_search)
```

ENTER THE NO. OF ELEMENTS10

55 ENTER THE ELEMENT TO SEARCH1587

66 ('ELEMENT', 55, 'NOT FOUND')

99 ('ELEMENT', 66, 'NOT FOUND')

44 ('ELEMENT', 99, 'NOT FOUND')

33 ('ELEMENT', 44, 'NOT FOUND')

111 ('ELEMENT', 33, 'NOT FOUND')

1587 ('ELEMENT', 111, 'NOT FOUND')

52 ('ELEMENT', 1587, 'IS FOUND IN THE', 6, 'POSITION')

63 ('ELEMENT', 52, 'NOT FOUND')

74 ('ELEMENT', 63, 'NOT FOUND')

('THE EXISTING LIST:', [55, 66, 99, 44, 33, 111, 1587, 52, 63, 74])

BINARY SEARCH ALGORITHM

- ❖ Binary search algorithm takes advantage of a collection of elements that is already sorted by ignoring half of the elements after just one comparison.
- ❖ Simple words:
- ❖ Element to be found by the algorithm is should become as an mid element of the list.

- ❖ Search a sorted array by repeatedly dividing the search interval in half.
- ❖ Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- ❖ Otherwise narrow it to the upper half.
- ❖ Repeatedly check until the value is found or the interval is empty.

ALGORITHM – A THEORETICAL EXPLANATION

- ❖ Compare x with the middle element.
 - ❖ X – the element to be search and found
 - ❖ Find the middle element of the given list / array using integer division method.
 - ❖ $\text{Mid} = (\text{Lower Bound} + \text{Upper Bound}) // 2$
 - ❖ Lower Bound – first element in list (or) array [possibly smallest element – because, the sorted list is only accepted by (work) Binary search algorithm to process]
 - ❖ Upper Bound – last element in list (or) array [possibly largest element]

- ❖ If x matches with the middle element, we return the mid index.
- ❖ Else if x is greater than the mid element,
 - ❖ then x can only lie in the right (greater) half sub array after the mid element.
 - ❖ Then we apply the algorithm again for the right half.
- ❖ Else if x is smaller,
 - ❖ the target x must lie in the left (lower) half.
 - ❖ So we apply the algorithm for the left half.

Search 23

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

23 > 16
take 2nd half

L=0	1	2	3	M=4	5	6	7	8	H=9
2	5	8	12	16	23	38	56	72	91

23 > 56
take 1st half

0	1	2	3	4	L=5	6	M=7	8	H=9
2	5	8	12	16	23	38	56	72	91

Found 23,
Return 5

0	1	2	3	4	L=5, M=5	H=6	7	8	9
2	5	8	12	16	23	38	56	72	91

"""

arr - collection of elements

low - lower bound index

high - higher bound index

x - target element

"""

RECURSIVE METHOD

```
def binary_search(arr, low, high, x):
```

```
    if high >= low:
```

```
        mid = (high+low) // 2
```

```
        if arr[mid] == x:
```

```
            return mid
```

```
        elif arr[mid] > x:
```

```
            return binary_search(arr, low, mid - 1, x)
```

```
        else:
```

```
            return binary_search(arr, low, mid + 1, x)
```

```
    else:
```

```
        return -1
```

ENTER THE LIST ELEMENT - COUNT 5

ENTER THE LIST ELEMENT 44

ENTER THE LIST ELEMENT 63

ENTER THE LIST ELEMENT 2

ENTER THE LIST ELEMENT 33

ENTER THE LIST ELEMENT 57

('LIST IS:', [44, 63, 2, 33, 57])

enter the element in the list to find 2

('ELEMENT PRESENT IN THE INDEX', '2')

#define an empty list

```
arr = []
```

```
lt = int(input("ENTER THE LIST ELEMENT - COUNT"))
```

#create list

```
for i in range(lt):
```

```
    arr.append(int(input("ENTER THE LIST ELEMENT")))
```

```
print("LIST IS:", arr)
```

#element to search & find

```
x = int(input("enter the element in the list to find"))
```

#function call

```
result = binary_search(arr, 0, len(arr)-1, x)
```

```
if result != -1:
```

```
    print("ELEMENT PRESENT IN THE INDEX", str(result))
```

```
else:
```

```
    print("ELEMENT IS NOT PRESENT IN THE INDEX")
```

ENTER THE LIST ELEMENT - COUNT 5

ENTER THE LIST ELEMENT 44

ENTER THE LIST ELEMENT 63

ENTER THE LIST ELEMENT 2

ENTER THE LIST ELEMENT 33

ENTER THE LIST ELEMENT 57

('LIST IS:', [44, 63, 2, 33, 57])

enter the element in the list to find 2

('ELEMENT PRESENT IN THE INDEX', '2')

```
def binary_search(list1, item):
    first = 0
    last = len(list1) - 1
    found = False
    while(first <= last and not found):
        mid = (first + last) // 2
        if list1[mid] == item:
            found = True
        else:
            if item < list1[mid]:
                last = mid - 1
            else:
                first = mid + 1
    return found
```

```
list1 = []
lt = int(input("ENTER THE LIST ELEMENT COUNT"))
for i in range(lt):
    list1.append(int(input("ENTER THE ELEMENTS")))

print("THE LIST IS:", list1)
element_to_search = int(input("ENTER THE ELEMENT TO FIND"))
print(binary_search(list1, element_to_search))
print(binary_search([1,2,3,5,8], 8))
```

ENTER THE LIST ELEMENT COUNT5

ENTER THE ELEMENTS11

ENTER THE ELEMENTS22

ENTER THE ELEMENTS33

ENTER THE ELEMENTS44

ENTER THE ELEMENTS55

('THE LIST IS:', [11, 22, 33, 44, 55])

ENTER THE ELEMENT TO FIND33

True



True

Thank You!