Axtarış alqoritmləri

Table of Contents

Searching Algorithms

- **Searching Algorithm** is an algorithm made up of a series of instructions that retrieves information stored within some data structure, or calculated in the search space of a problem domain.

- There are many sorting algorithms, such as:
  - Linear Search, Binary Search, Jump Search, Interpolation Search, Exponential Search, Ternary Search

Linear Search

Linear Search

- **Linear Search** is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

Linear Search

❑ **Algorithm:**

- **Step1:** Start from the leftmost element of array and one by one compare x with each element of array.
- Step2: If x matches with an element, return the index.
- Step3: If x doesn't match with any of elements, return -1.

Linear Search

- Assume the following Array:
- Search for 9

8	12	5	9	2

Linear Search

- Compare
- X= 9

Linear Search

- Compare
- X= 9

Linear Search

- Compare
- X= 9

Linear Search

- Compare
- X= 9

Linear Search

- Found at index = 3

Linear Search

- Python Code

```
def LinearSearch(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1
```

Linear Search

```
arr = [12, 6, 5, 14, 3]
position=LinearSearch(arr,14)
print(position)
```

Linear Search

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

- Example of worst case: search for the last element

Binary Search

Binary Search

- **Binary Search** is the most popular Search algorithm. It is efficient and also one of the most commonly used techniques that is used to solve problems. Binary search use sorted array by repeatedly dividing the search interval in half.

Binary Search

- Algorithm:

- Step1: Compare x with the middle element.
- Step2: If x matches with middle element, we return the mid index.
- Step3: Else If x is greater than the mid element, search on right half.
- Step4: Else If x is smaller than the mid element. search on left half.

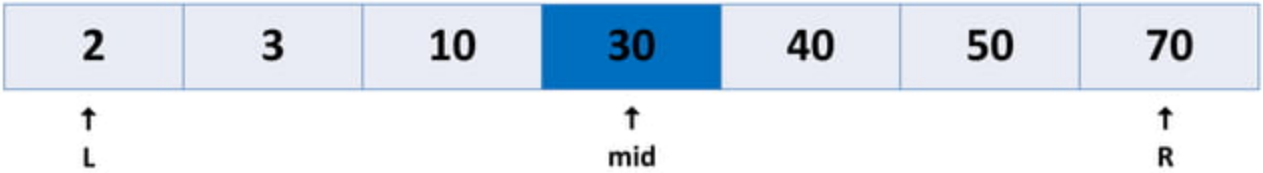
Binary Search

- Assume the following Array:
- Search for 40

2	3	10	30	40	50	70

Binary Search

- Compare
- X = **40**

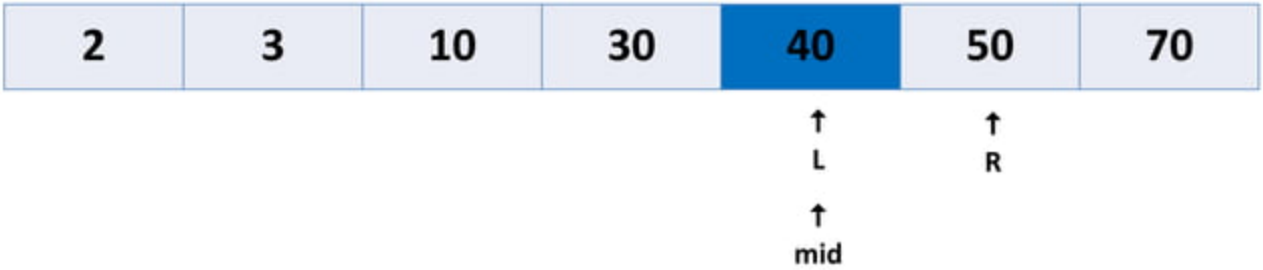


Binary Search

- Compare
- X = **40**

Binary Search

- Compare
- X = **40**



Binary Search

- x=40 , found at index = 4

Binary Search

- Iterative python implementation:

```
def BinarySearch(arr, l, r, x):
    while l <= r:
        mid = int(l + (r - l)/2)
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

Binary Search

```
arr = [ 2, 3, 10, 30, 40, 50,60]
x = 40

result = BinarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print("Element is present at index %d" % result)
else:
    print("Element is not present in array")
```

Binary Search

- Recursive Python implementation:

```
def BinarySearch (arr, l, r, x):
    if r >= l:
        mid = int(l + (r - l)/2)
        print(mid)
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return BinarySearch(arr, l, mid-1, x)
        else:
            return BinarySearch(arr, mid+1, r, x)
    else:
        return -1
```

Binary Search

```
arr = [ 2, 3, 10, 30, 40, 50,60]
x = 40

result = BinarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print("Element is present at index %d" % result)
else:
    print("Element is not present in array")
```

Binary Search

- **Time Complexity:** $O(\log_2 n)$

Jump Search

Jump Search

- Jump Search is a searching algorithm for sorted arrays. The basic idea is to check fewer elements (than linear search) by jumping ahead by fixed steps or skipping some elements in place of searching all elements.

Jump Search

❑ **Algorithm:**

- **Step1:** Calculate Jump size
- Step2: Jump from index i to index i+jump
- Step3: If x = = arr[i+jump] return x
- Else jump back a step
- Step4: Perform linear search

Jump Search

- Assume the following sorted array:
- Search for 77

0	1	1	2	3	5	8	13	21	34	55	77	89	91	95	110

Jump Search

❑ Calculate:

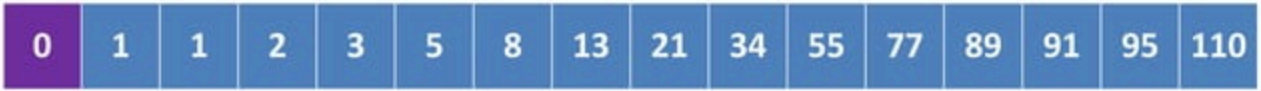
- Size of array n =16
- Jump size = sqrt(n)= 4

0	1	1	2	3	5	8	13	21	34	55	77	89	91	95	110

Jump Search

- Jump size = 4
- Search from index 0
- Compare index value with search number 0<77

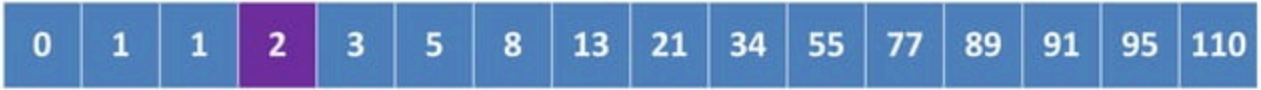
77



Jump Search

- Jump size = 4
- Jump from index 0 to index 3
- Compare index value with search number 2<77

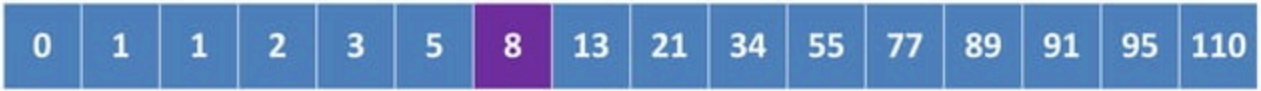
77



Jump Search

- Jump size = 4
- Jump from index 3 to index 6
- Compare index value with search number 8<77

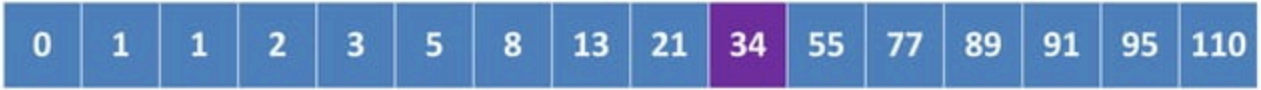
77



Jump Search

- Jump size = 4
- Jump from index 6 to index 9
- Compare index value with search number 34<77

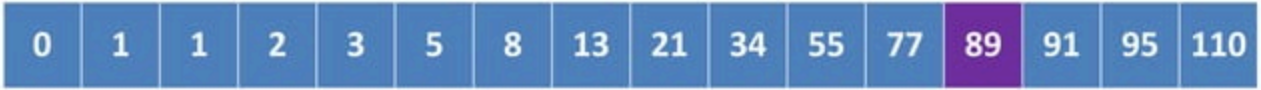
77



Jump Search

- Jump size = 4
- Jump from index 9 to index 12
- Compare index value with search number 89>77

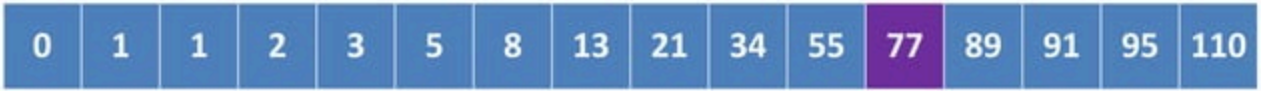
77



Jump Search

- jump back a step
- Perform linear search
- Compare found at index 11

77



Jump Search

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

Jump Search

```
def jump_search(arr, val):
    jump = int(math.sqrt(len(arr)))
    left, right = 0, 0
    while left < len(arr) and arr[left] <= val:
        right = min(len(arr) - 1, left + jump)
        if arr[left] <= val and arr[right] >= val:
            break
        left += jump;

    if left >= len(arr) or arr[left] > val:
        return -1

    right = min(len(arr) - 1, right)
    i = left
    while i <= right and arr[i] <= val:
        if arr[i] == val:
            return i
        i += 1

    return -1
```

Jump Search

❑ Python Code

```
arr = [1, 3, 5, 7, 9, 11, 13, 15, 17]
print(jump_search(arr, 5)) # 2
print(jump_search(arr, 7)) #  3
print(jump_search(arr, 10)) #  -1
print(jump_search(arr, 15)) # 7
```

Interpolation Search

Interpolation Search

- The **Interpolation Search** is an improvement over Binary Search for instances. On the other hand interpolation search may go to different locations according the value of key being searched.

Interpolation Search

❑ **Algorithm:**

- **Step1:** In a loop, calculate the value of "pos" using the position formula.
- **Step2:** If it is a match, return the index of the item, and exit.
- **Step3:** If the item is less than arr[pos], calculate the position of the left sub-array. Otherwise calculate the same in the right sub-array.
- **Step4:** Repeat until a match is found or the sub-array reduces to zero.

Interpolation Search

// The idea of formula is to return higher value of pos

// when element to be searched is closer to arr[hi]. And

// smaller value when closer to arr[lo]

pos = lo + [(x-arr[lo])*(hi-lo) / (arr[hi]-arr[Lo])]

arr[] ==> Array where elements need to be searched

x ==> Element to be searched

lo ==> Starting index in arr[]

hi ==> Ending index in arr[]

Interpolation Search

10	12	13	16	18	19	20	21	22	23	24	33	35	42	47

Interpolation Search

- Assume the following sorted array:
- Search for x= 18

10	12	13	16	18	19	20	21	22	23	24	33	35	42	47

Interpolation Search

- Calculate **pos = lo + [(x-arr[lo])*(hi-lo) / (arr[hi]-arr[Lo])]**
- **Lo=0, hi=14, arr[lo]=10, arr[hi]= 47**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
10	12	13	16	18	19	20	21	22	23	24	33	35	42	47

Interpolation Search

- Calculate **pos = 3**
- **Compare with x=18**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
10	12	13	16	18	19	20	21	22	23	24	33	35	42	47

Interpolation Search

- Calculate **pos = lo + [(x-arr[lo])*(hi-lo) / (arr[hi]-arr[Lo])]**
- **Lo=4, hi=14, arr[lo]=18, arr[hi]= 47**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
10	12	13	16	18	19	20	21	22	23	24	33	35	42	47

Interpolation Search

- Calculate **pos = 4**
- **Compare with x=18 , found at index 4**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
10	12	13	16	18	19	20	21	22	23	24	33	35	42	47

Interpolation Search

- **Time Complexity:** If elements are uniformly distributed, then **O (log log n))**. In worst case it can take up to **O(n)**.

Interpolation Search

❑ Python Code

```
def interpolationSearch(arr, x):
    lo = 0
    hi = len(arr)-1
    while lo <= hi and x >= arr[lo] and x <= arr[hi]:
        pos  = lo + int(((float(hi - lo) /
            ( arr[hi] - arr[lo])) * ( x - arr[lo])))
        if arr[pos] == x:
            return pos
        if arr[pos] < x:
            lo = pos + 1;
        else:
            hi = pos - 1;

    return -1
```

Interpolation Search

❑ Python Code

```
arr = [10, 12, 13, 16, 18, 19, 20, 21, 22, 23, 24, 33, 35, 42, 47]
x = 18 # Element to be searched
index = interpolationSearch(arr, x)

if index != -1:
    print("Element found at index",index)
else:
    print("Element not found")
```

Contact Me


facebook.com/mloey
mohamedloey@gmail.com
twitter.com/mloey


linkedin.com/in/mloey
mloey@fci.bu.edu.eg
mloey.github.io


THANKS FOR
YOUR TIME