

**Islamic University of Technology (IUT)**  
Organization of Islamic Cooperation (OIC)  
Department of Electrical and Electronic Engineering (EEE)

Course No.	Math 4522
Course Name	Numerical Methods Lab.
Experiment No.	02
Experiment Name	Introduction to Different Types of Searching Algorithms

---

## Objectives

- To be familiarized with different searching algorithms and how to implement them in MATLAB.
- To understand the difference between different searching techniques based on their time complexity.

## Theory

The purpose of search algorithms is to locate or retrieve an element from data. The data can be stored in any kind of data structure like a list or a dictionary. These searching algorithms can be divided into two categories –

1. Sequential Search
2. Interval Search

The sequential search algorithms traverse the list sequentially to find the desired value. The interval search algorithms work for sorted data. Instead of checking every element in the list like sequential search, interval search algorithms look for the desired element heuristically. It is based on the divide and conquer algorithm.

Different searching algorithms –

- Linear Search
- Binary Search
- Jump Search
- Interpolation Search
- Exponential Search
- Breadth First Search (BFS) etc.

Prepared by  
**ASIF NEWAZ**  
*Lecturer, Department of EEE, IUT*

## Linear Search

In this algorithm, you search each element in the list sequentially to find the desired value. You start with the first element and continue until a match is found.

The time complexity of the algorithm is:  $O(n)$ .

Best-case scenario:  $O(1)$  → 1<sup>st</sup> element is the desired one

Worst-case scenario:  $O(n)$  → value not in the array

It is not the most efficient approach when the sample size is very large.

### Programming Exercise

- Write a code to check whether an element exist in an array or not using linear search algorithm.

```
% Linear search
```

```
tic
```

```
a= randi(100000,1,1000000); % creating a random 1D array with 1 million parameters  
val= 1200000;
```

```
for i= 1:length(a)  
    if a(i)==val  
        out='value found';  
        break  
    else  
        out='value not found';  
    end  
end  
out  
toc
```

### Built-in functions

MATLAB has built-in functions to allow you to search for an item in an array. However, here we are more interested in finding out the actual approach behind MATLAB's built-in functions.

- `find (a==val)` will return the indices where the element is located.
- `ismember (val,a)` will return logical true/false whether the value exist in the array or not.

Prepared by

**ASIF NEWAZ**  
Lecturer, Department of EEE, IUT

## Binary Search

This is an interval search technique. It only works in a sorted array. So, the original list needs to be arranged in an ascending order (numeric) or dictionary order (string) if not originally sorted. This method works based on decrease and conquer technique to find the desired value with less time complexity. In case of large data, the number of iterations required to find the value is significantly reduced in binary search compared to sequential search algorithm.

In binary search, we start with the middle element in the sorted array. If that is the desired element, then search is complete with minimum steps (best-case). If not, then we find whether the desired value is smaller or larger than the middle element. Based on that, we narrow the search to only half of the original array. Then the middle element of the new array (size =  $n/2$ ) is found, and the process is repeated. This way, with each step, the size of the array is halved, and the total number of iterations required to find the value is reduced.

The time complexity of the algorithm is:  $O(\log(n))$ , which makes it much more efficient than linear search.

If searching for 23 in the 10-element array:

	2	5	8	12	16	23	38	56	72	91
23 > 16, take 2 <sup>nd</sup> half	L									H
	2	5	8	12	16	23	38	56	72	91
23 < 56, take 1 <sup>st</sup> half										
	2	5	8	12	16	23	38	56	72	91
Found 23, Return 5										
	2	5	8	12	16	23	38	56	72	91

## Algorithm

- Start by assigning index value  $\rightarrow$  low = 1, high = number of elements in the array.
- Find the mid element index of the array using  $\text{mid} = (\text{low} + \text{high}) / 2$ . Take the floor value in case of fraction.
- Check whether  $\text{array}(\text{mid})$  is lower or higher than the item we are searching for.

Prepared by

**ASIF NEWAZ**  
Lecturer, Department of EEE, IUT

- iv. If  $\text{array}(\text{mid}) < \text{search\_item}$ , then the item exists in the upper half. So, we can narrow the search in the upper half portion only by moving low to mid+1.
- v. If  $\text{array}(\text{mid}) > \text{search\_item}$ , then the item exists in the lower half. So, we can narrow the search in the lower half portion of the array only by moving high to mid-1.
- vi. Repeat the steps 2 to 5 until the element is found. Otherwise, the item is not on the list.

## Programming Exercise

- Write a code to check whether an element exist in an array or not using binary search algorithm.

```
rng(1, 'twister'); % generating same numbers on each run
s=rng; % for more info -
https://www.mathworks.com/help/matlab/math/controlling-random-number-generation.html

a= randi(100000,1,1000000); % creating a random 1D array with 1 million parameters
a= sort(a, 'a'); % binary search only works on sorted array
%a=[1,2,3,3,4,4,5,5,10,20];
val= 1031;

low=1;
high= length(a);
mid= floor((low+high)/2);

while low<=high
    if a(mid)<val
        low= mid+1;
    elseif a(mid)>val
        high= mid-1;
    else
        out= 'value is found';
        val_index= mid;
        break;
    end

    mid= floor((low+high)/2);
end

if low>high
    out= 'value not found';
    val_index=nan;
end

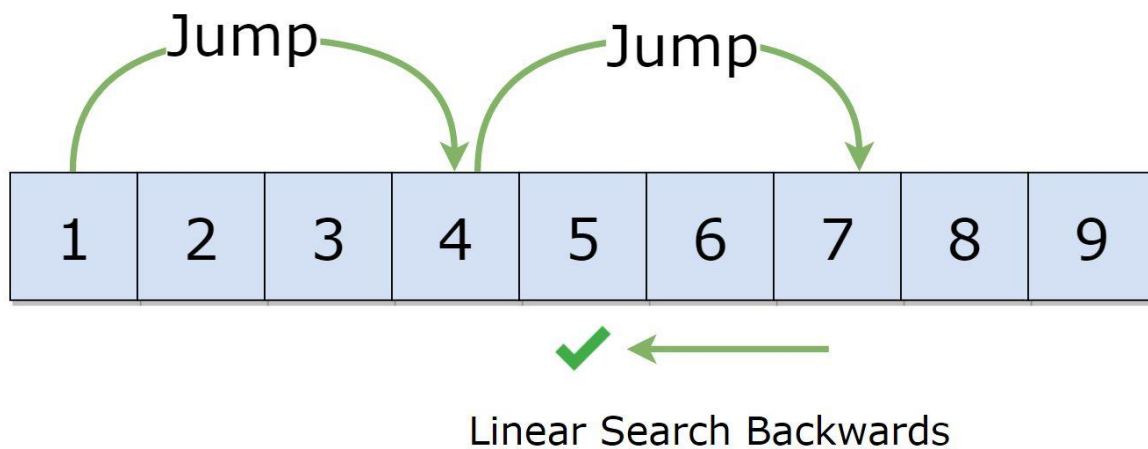
out
val_index
```

Prepared by

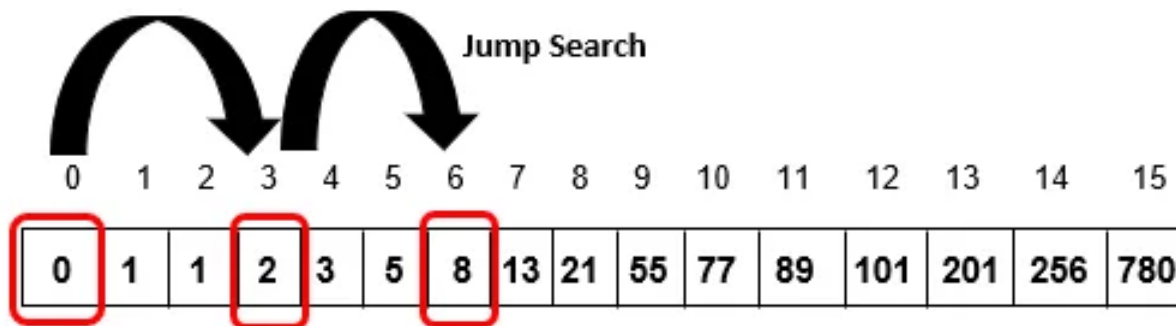
**ASIF NEWAZ**  
Lecturer, Department of EEE, IUT

## Jump Search

Jump search is an improvement over linear search, but less efficient than binary search. Unlike linear search, it works only on sorted array. Instead of searching through all elements sequentially, it jumps a few step/skips a few elements to reduce the number of iterations required.



In the figure, a jump step of 3 is utilized. So, the method checks the index – 1, 4, 7, and so on. The step size is taken depending on the size of the original array.



Say, the element we are looking for is 77. Using jump search, we will take four jumps first and reach index 12 and find the value larger than the search key. Then we will go backwards and perform a linear search operation (two steps required) to find the desired value. As can be seen, the number of iterations required to find the value is lower than linear search.

The time complexity of the algorithm is:  $O(\sqrt{n})$

The time complexity is in between linear search and binary search.

Prepared by

**ASIF NEWAZ**  
Lecturer, Department of EEE, IUT

## Interpolation Search

This is an improvement over binary search. While binary search always goes for the middle element, interpolation search can go in different locations according to the search item. If the search item is closer to the last element in the list, interpolation search will start searching towards the end side.

The time complexity of the algorithm is:  $O(\log(\log(n)))$  - (best case).

## Assignment

1. Implement MATLAB's find function's operation using 'linear search' and 'binary search' algorithms. The 'find' function returns all the matches in the array along with the indices.  
What do you think is the most efficient way to find all the indices that has the search element?
2. Write a code to check whether an element exist in an array or not using 'jump search' algorithm.

Prepared by

**ASIF NEWAZ**  
*Lecturer, Department of EEE, IUT*