

Searching in an array

CS10003 PROGRAMMING AND DATA STRUCTURES



Searching

Check if a given element (called **key**) occurs in the array.

- **Example:** array of student records; **rollno** can be the key.

Two methods to be discussed:

- a) If the array elements are unsorted.
 - **Linear search**
- b) If the array elements are sorted.
 - **Binary search**

Basic Concept of Linear Search

Basic idea

- Start at the beginning of the array.
- Inspect elements one by one to see if it matches the **key**.
- If a match is found, return the array index where the match was found.
- If no match is found, a special value is returned (like -1).

Linear Search (contd.)

Function `linear_search` returns the array index where a match is found.

It returns `-1` if there is no match.

```
int linear_search (int a[], int size, int key)
{
    int pos = 0;
    while ((pos < size) && (key != a[pos]))
        pos++;
    if (pos < size)
        return pos;
    return -1;
}
/* Return the position of match */
/* No match found */
```

Time Complexity of Linear Search

A measure of **how many basic operations** an algorithm needs to perform before terminating.

Example of basic operation: **match / compare two elements.**

- If there are n elements in the array:
 - **Best case:**
match found in first element (1 search operation)
 - **Worst case:**
no match found, or match found in the last element (n search operations)
 - **Average case:** $(n + 1) / 2$ search operations

Binary Search

Basic Concept

Binary search is applicable if the array is *sorted*.

Basic Idea

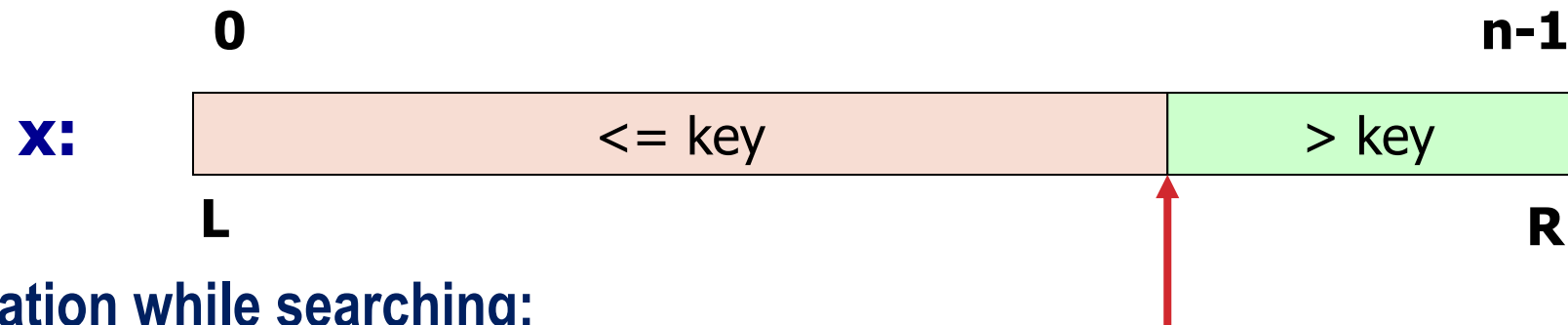
- Look for the target in the middle.
- If you don't find the key, you can ignore half of the array, and repeat the process with the other half.

In every step, we reduce, by a factor of two, the number of elements to search from.

The Basic Strategy

What do we want?

- Plan to find the array index between values larger and smaller than **key**:



- Situation while searching:
 - Initially, the search window is the entire array, that is, L and R are initialized to the indices of the first and the last elements.
- Look at the element at index $[(L+R)/2]$.
 - Discard one half of the search window depending on the outcome of test.

Initialization and Return Value

```
int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    L = 0; R = size - 1;
    while (L != R)
    {
        mid = (L + R) / 2;
        if (key <= x[mid]) R = mid;
        else L = mid + 1;
    }
    if (key == x[L])
        return L;
    else
        return -1;
}
```

If **key** appears in $x[0\dots\text{size}-1]$, return its location, pos such that $x[\text{pos}]==\text{key}$.

If not found, return -1

Binary Search Examples

Sorted array

-17	-5	3	6	12	21	45	63	50
0	1	2	3	4	5	6	7	8

Trace

```
bin_search (x, 9, 3);
binsearch(x, 9, 2);
```

L	R	M	key <=
x[M]?			
0	8	4	3 <=
12?	[True]		
0	4	2	3 <= 3?
	[True]		

L	R	M	key <=
x[M]?			
0	8	4	2 <=
12?	[True]		
0	4	2	2 <= 3?
	[True]		

We can modify the algorithm by checking equality with $x[mid]$

5? [False]
2 2 [Loop

terminates]

key == x[L]?
[True]

5? [False]
2 2 [Loop

terminates]

key == x[L]?
[False]

Another Version of Iterative Binary Search

```
int bin_search_1 (int x[], int size, int key)
{
    int L, R, mid;
    L = 0; R = size-1;
    while (L <= R)
    {
        mid = (L + R) / 2;
        if (key == x[mid]) return mid;
        if (key < x[mid]) R = mid - 1;
        else L = mid + 1;
    }
    return -1;
}
```

Unsorted vs Sorted Array Search: Where's the difference?

Suppose that the array **x** has 1000 elements.

Linear search

If **key** is a member of **x**, it would require about 500 comparisons on the average.

Binary search

- After 1st compare, left with 500 elements.
- After 2nd compare, left with 250 elements.
- After at most 10 steps, you are done.

Time Complexity

If there are n elements in the array.

- Number of iterations required:

$$\log_2 n$$

For $n = 64$ (say).

- Initially, list size = 64.
- After first compare, list size = 32.
- After second compare, list size = 16.
- After third compare, list size = 8.
- ...
- After sixth compare, list size = 1.

$2^k = n$, where k is the number of steps.

$$\log_2 64 = 6$$

$$\log_2 1024 = 10$$

Recursive Version of Binary Search

The algorithm for binary search directly leads to a recursive formulation.

- **The algorithm is called recursively by adjusting the left or right pointers, as applicable.**
- **The base condition is: the element is found, or the left and right pointers cross.**

```

int binarySearch (int x[], int L, int R, int key)
{
    int mid;
    if (L <= R) {
        mid = (L + R) / 2;
        if (key == x[mid])    // If the element is present at the middle
            return mid;
        if (key < x[mid])    // Look into the left subarray
            return binarySearch (x, L, mid-1, key);
        else                 // Look into the right subarray
            return binarySearch (x, mid+1, R, key);
    }
    // Element is not present in array
    return -1;
}

```

Returns location of key in given array arr[L ... R] if present, otherwise -1

```

int result = binarySearch (arr, 0, n-1, key);
if (result == -1)
    printf ("Key is not present in array\n");
else
    printf("Key is present at index %d\n", result);

```