



Search Algorithms



Brute Force Approach - Linear Search
Divide and Conquer Approach - Binary Search



Problem Statement: Search

- Given a list of records, where each record has an associated key
- Give efficient algorithm for searching for a record containing a particular key.
- Efficiency is quantified in terms of average time analysis (number of comparisons) to retrieve an item.

Linear / Sequential Search

It is used for unsorted and unordered small list of elements.

It has a time complexity of $O(n)$, which means the time is linearly dependent on the number of elements.

How to Find a Value in an Array?

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

Suppose you have a big array full of data, and you want to find a particular value.

How will you find that value?

Linear Search Example #1

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



element

Searching for -86.

Linear Search Example #2

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



element

Searching for -86.

Linear Search Example #3

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

↑
element

Searching for -86.

Linear Search Example #4

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

↑
element

Searching for -86.

Linear Search Example #5

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

↑
element

Searching for -86.

Linear Search Example #6

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

↑
element

Searching for -86: found!

Linear Search

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

Linear search means looking at each element of the array, in turn, until you find the target value.

Binary Search

- Used with sorted array or list.
- Steps to search 'x' in list[l..r]:
 1. Compute $\text{mid} = (l+r) / 2$
 2. Compare x with list[mid].
 3. If matched, return mid.
 4. If not, check whether x is less or greater than list[mid].
 5. If $x > \text{list}[\text{mid}]$, then pick the elements on the right side of the middle element, i.e., set $l = \text{mid} + 1$ (as the list/array is sorted, hence on the right, we will have all the numbers greater than the middle number), and goto step 1.
 6. If $x < \text{list}[\text{mid}]$, then pick the elements on the left side of the middle element, i.e., set $r = \text{mid} - 1$ and start again from the step 1.

Algorithm

Algorithm binary_search

$A \leftarrow$ sorted array

$n \leftarrow$ size of array

$x \leftarrow$ value to be searched

Set lowerBound = 1

Set upperBound = n

```
while x not found
  if upperBound < lowerBound
    EXIT: x does not exists.
```

```
  set midPoint = lowerBound + ( upperBound - lowerBound ) / 2
```

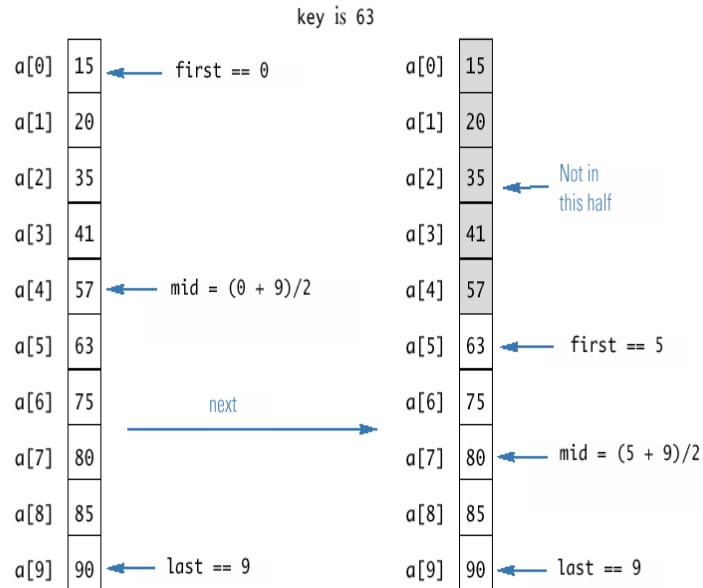
```
  if  $A[\text{midPoint}] < x$ 
    set lowerBound = midPoint + 1
```

```
  if  $A[\text{midPoint}] > x$ 
    set upperBound = midPoint - 1
```

```
  if  $A[\text{midPoint}] = x$ 
    EXIT: x found at location midPoint
end while
```

```
end procedure
```

Display 11.7 Execution of the Method search



Display 11.7 Execution of the Method search (continued)

