Introduction to Searching

Linear Search

- The linear search is a sequential search, which uses a loop to step through an array, starting with the first element.
- It compares each element with the value being searched for, and stops when either the value is found or the end of the array is encountered.
- If the value being searched is not in the array, the algorithm will unsuccessfully search to the end of the array.

Linear Search

- Since the array elements are stored in linear order searching the element in the linear order make it easy and efficient.
- The search may be successful or unsuccessfully. That is, if the required element is found them the search is successful other wise it is unsuccessfully.

Advantages

- The linear search is simple - It is very easy to understand and implement
- It does not require the data in the array to be stored in any particular order

Disadvantages

- It has very poor efficiency because it takes lots of comparisons to find a particular record in big files
- The performance of the algorithm scales linearly with the size of the input
- Linear search is slower then other searching algorithms

Example 2.8 Linear Search

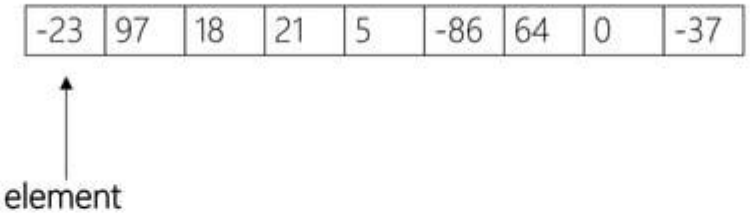
Suppose a linear array DATA contains *n* elements, and suppose a specific ITEM of information is given. We want either to find the location LOC of ITEM in the array DATA, or to send some message, such as LOC = 0, to indicate that ITEM does not appear in DATA. The linear search algorithm solves this problem by comparing ITEM, one by one, with each element in DATA. That is, we compare ITEM with DATA[1], then DATA[2], and so on, until we find LOC such that ITEM = DATA[LOC]. A formal presentation of this algorithm follows.

Algorithm 2.4: (Linear Search) A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC = 0.

```
1. [Initialize] Set K := 1 and LOC := 0.
2. Repeat Steps 3 and 4 while LOC = 0 and K ≤ N.
3.        If ITEM = DATA[K], then: Set LOC: = K.
4.        Set K := K + 1. [Increments counter.]
   [End of Step 2 loop.]
5. [Successful?]
   If LOC = 0, then:
          Write: ITEM is not in the array DATA.
   Else:
          Write: LOC is the location of ITEM.
   [End of If structure.]
6. Exit.
```

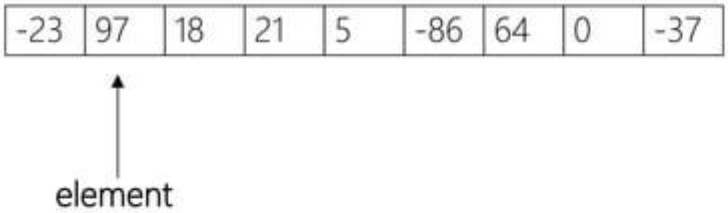
mparisons b

Linear Search Example



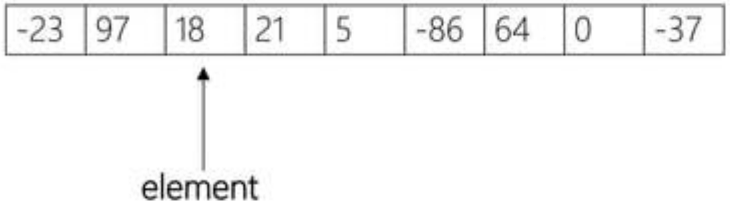
Searching for -86.

Linear Search Example



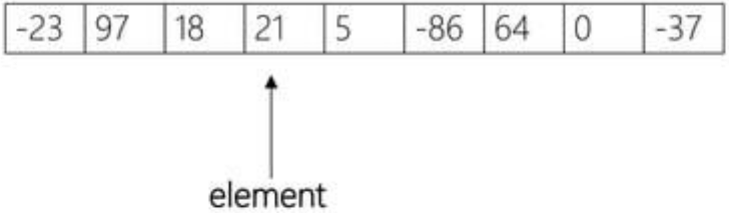
Searching for -86.

Linear Search Example



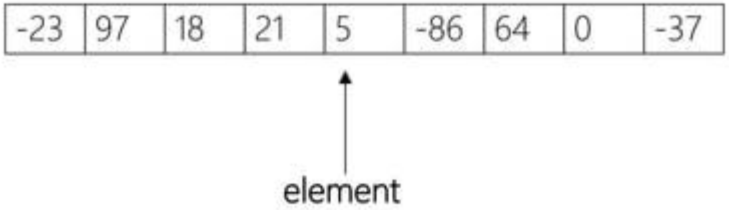
Searching for -86.

Linear Search Example



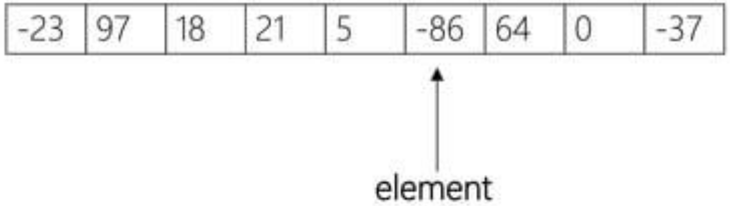
Searching for -86.

Linear Search Example



Searching for -86.

Linear Search Example



Searching for -86: found!

Analysis of Linear Search

How long will our search take?

In the best case, the target value is in the first element of the array.

So the search takes some tiny, and constant, amount of time.

In the worst case, the target value is in the last element of the array.

So the search takes an amount of time proportional to the length of the array.

Analysis of Linear Search

In the average case, the target value is somewhere in the array.

In fact, since the target value can be anywhere in the array, any element of the array is equally likely.

So on average, the target value will be in the middle of the array.

So the search takes an amount of time proportional to half the length of the array

Binary Search

The general term for a smart search through sorted data is a ***binary search***.

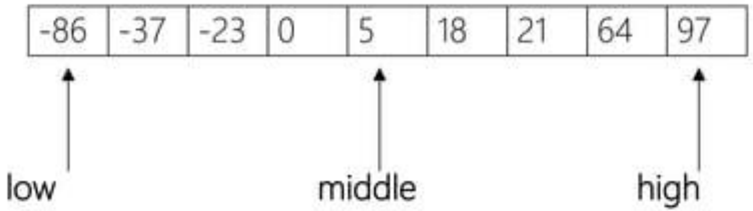
1. The initial search region is the whole array.
2. Look at the data value in the middle of the search region.
3. If you've found your target, stop.
4. If your target is less than the middle data value, the new search region is the lower half of the data.
5. If your target is greater than the middle data value, the new search region is the higher half of the data.
6. Continue from Step 2.

Algorithm 4.6: (Binary Search) BINARY(DATA, LB, UB, ITEM, LOC)
Here DATA is a sorted array with lower bound LB and upper bound UB, and ITEM is a given item of information. The variables BEG, END and MID denote, respectively, the beginning, end and middle locations of a segment of elements of DATA. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL.

```
1. [Initialize segment variables.]
   Set BEG := LB, END := UB and MID = INT((BEG + END)/2).
2. Repeat Steps 3 and 4 while BEG ≤ END and DATA[MID] ≠ ITEM.
3.       If ITEM < DATA[MID], then:
             Set END := MID – 1.
         Else:
             Set BEG := MID + 1.
         [End of If structure.]
4.       Set MID := INT((BEG + END)/2).
   [End of Step 2 loop.]
5. If DATA[MID] = ITEM, then:
       Set LOC := MID.
   Else:
       Set LOC := NULL.
   [End of If structure.]
6. Exit.
```

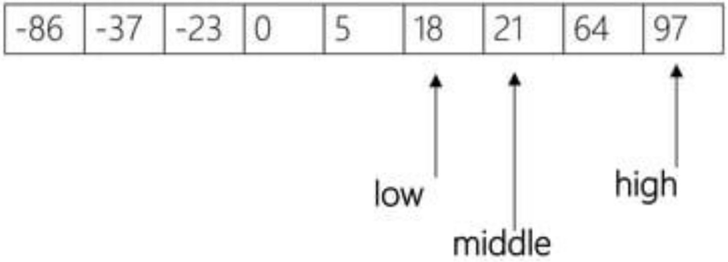
Remark: Whenever ITEM does not appear in DATA, the algorithm eventually arrives at the stage that BEG = END = MID. Then the next step yields END < BEG, and control transfers to Step 5 of the algorithm. This occurs in part (b) of the next example.

Binary Search Example



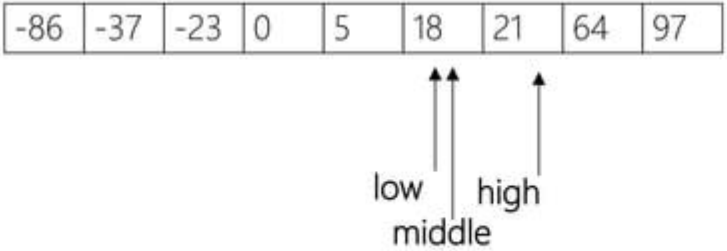
Searching for 18.

Binary Search Example



Searching for 18.

Binary Search Example



Searching for 18: found!

Time Complexity of Binary Search

How fast is binary search?

Think about how it operates: after you examine a value, you cut the search region in half.

So, the first iteration of the loop, your search region is the whole array.

The second iteration, it's half the array.

The third iteration, it's a quarter of the array.

...

The k^{th} iteration, it's $(1/2^{k-1})$ of the array.

Need to Sort Array

Binary search only works if the array is already sorted.

It turns out that sorting is a huge issue in computing – and the subject of our next lecture.