Searching

Searching

It is a process for finding an item with specified properties (e.g. specified text) among a collection of items using a search algorithm.

Two types of searching algorithms:

(1) Linear Search

(2) Binary Search

Linear Search Algorithm

Definition:

It is a method for finding a particular value in a **list** that checks each element in sequence until the desired element is found or the list is exhausted.

It is a collection of data items to be searched is organized in a list ***x_1, x_2, … x_n.***

Assume **= =** operator defined for the type Linear Search (and **<** operator defined for Binary Search)

Linear search **begins with item 1** and it will continue through the list until target found or reach end of list.

Linear Search Algorithm

The **Linear search implementation** is based on the following:

(a) Array / Vector
(b) Singly-linked list
(c) Iterative

Algorithmic Steps:

1) Start with the first element in the array or list.
2) Compare it with the given key, if key and value at current index are same, return the current index.
3) Else increase the index value and repeat step 2 until end of list is reached.

Time Complexity:

The worst-case time complexity is **O(n)**

The best-case time complexity is **O(1)**

Linear Search Algorithm

(a) Array/Vector based search algorithm

```
1. set found = false
2. set loc = 0
3. While loc < n and not found
   If item ==a[loc] then
   set found = true
   Else
   Increment loc by 1
   End while
```

Linear Search Algorithm

(b) Singly-linked list based search algorithm

```
/* A linked list with first node pointed to by
firstNode for item; found = true and locptr =
pointer to a node containing item if the search is
successful otherwise found is false */

1. Set found = false
2. Set locptr = firstNode
3. While locptr # null and not found
     If item == locptr->data
        set found = true
     Else
        set locptr = locptr ->next
     End while
```

Linear Search Algorithm

(c) Iterator based search algorithm

```
/*Iterators begin and end positioned at the first
  element and beyond the last element for item
    LocIter is positioned at itme if the search is
  successful otherwise locIter==end */

1. Set iterator locIter = begin
2. While(locIter # end) and (*locIter #
          item) do
          ++locIter
   End while
```

The worst case time complexity → O(n)

Binary Search Algorithm

- Search a *sorted array* by repeatedly dividing the search interval in half.
- If the value of the search *key* is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise narrow it to the upper half.
- Repeatedly check until the value is found or the interval is empty.
- It is also called as **divide-and-conquer search algorithm**.
- Types:
  a) Array / Vector algorithm
  b) Linked List algorithm

Binary Search Algorithm

(a) Array/Vector based algorithm

```
1.set found = false
2.set first = 0
3.set last = n-1
4.While first <=last < n and not found
   If item ==a[loc] then
      set found = true
   Else
      Increment loc by 1
   End while
```

Binary Search Algorithm

(a) Array/Vector based algorithm

Set *found* = false.
Set *first* = 0.
Set *last* = $n - 1$.
While *first* $\leq$ *last* and not *found* do the following:
    Calculate *loc* = (*first* + *last*) / 2.
    If *item* < *a* [*loc*] then
        Set *last* = *loc* – 1.   // search first half
    Else if *item* > *a* [*loc*] then
        Set *first* = *loc* + 1.   // search last half
    Else
        Set *found* = true.   // *item* found
End while.

The worst case computing time for binary search → O(log2n)

Binary Search Algorithm

(a) Array/Vector based algorithm

Advantages:

(a) Usually **outperforms** a linear search

(b) Very easy to implement

Disadvantage:

(a) Requires a **direct-access** storage

(b) Not appropriate for linked lists

Binary Search Algorithm

(b) Linked List based Algorithm

1. Set *found* = false.
2. Set *first* = 0.
3. Set *last* = *n* – 1.
4. Set *locptr* = *firstNode*.
5. While *first* ≤ *last* and not *found* do the following:
    a. Calculate *loc* = (*first* + *last*) / 2.
    b. For *i* ranging from *first* to *loc* – 1 do the following:
        locptr = *locptr*->*next*.
       End for.
    c. If *item* < *locptr*->*data* then
        Set *last* = *loc* – 1.   // search first half
       Else if *locptr*->*data* < *item* then
        Set *first* = *loc* + 1.   // search last half
       Else
        Set *found* = true.   // *item* found
   End while

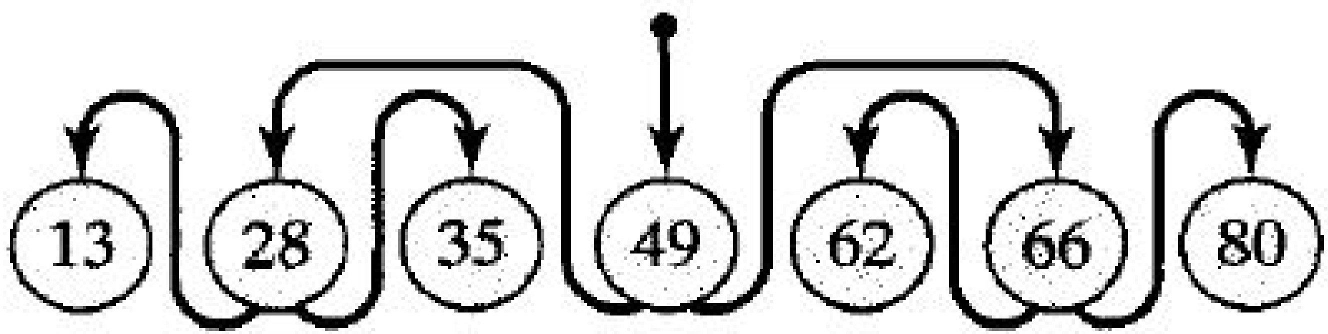
Binary Search Algorithm

(b) Linked List based Algorithm

In the worst case, when item is greater than all elements in the list, the for loop will move the pointer *locptr through the entire list, which means that the computing time* is *O(n) instead of O(log2 n).*

Binary Search Tree

Consider the following ordered list of integers



1. Examine middle element
2. Examine left, right sublist (maintain pointers)
3. (Recursively) examine left, right sublists

Binary Search Tree

Redraw the previous structure so that it has a tree like shape – a binary tree