Searching Algorithms

Introduction

- Searching for
  - a book in library
  - Subscriber's telephone number in the telephone directory
  - Name in the electoral rolls

Linear Search

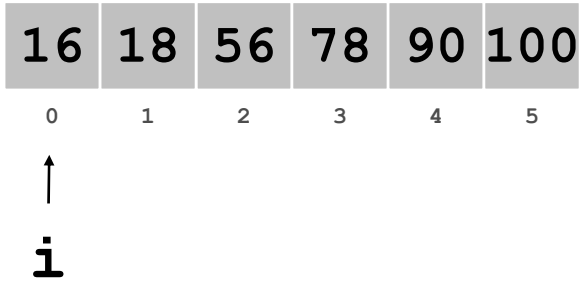
- Sequential search
- Searched for a key one after the other
- Ordered and unordered search

Ordered Linear Search

- L={k1,k1,k3,…,kn} such that k1<k2<k3<…<kn
- To search key k in the list
  - Compare key k with each element so long as k>ki
  - If k=ki then search is done
  - Otherwise search is unsuccessful
- L[0:5]={16,18,56,78,90,100}
  - Search 78 and 20

Ordered linear search for k= 78

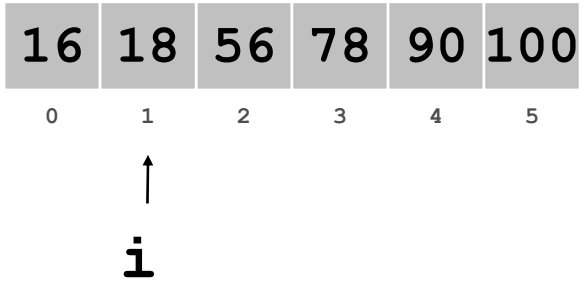
.16,18,56,78,90,100



i<n and k>L[i]

Ordered linear search for k= 78

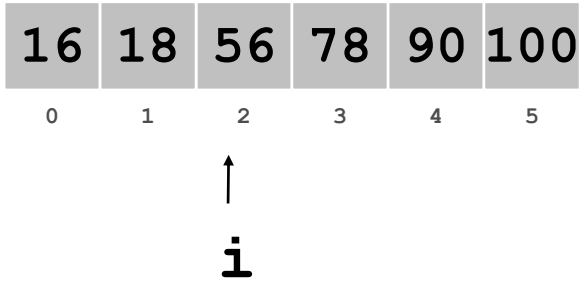
.16,18,56,78,90,100



i<n and k>L[i]

Ordered linear search for k= 78

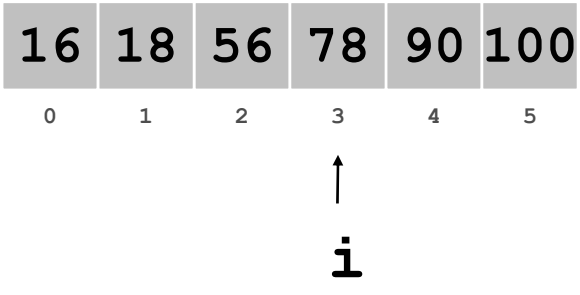
.16,18,56,78,90,100



i<n and k>L[i]

Ordered linear search for k= 78

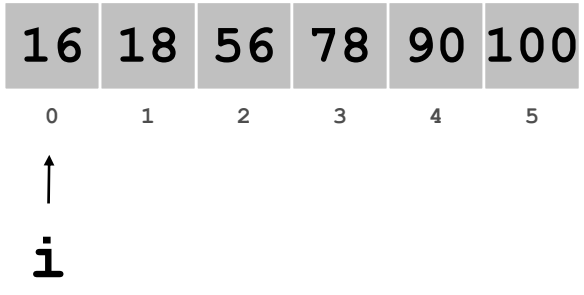
.16,18,56,78,90,100



K=L[i], so key is found

Ordered linear search for k= 20

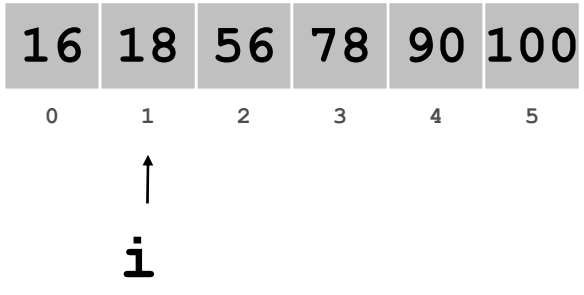
.16,18,56,78,90,100



i<n and k>L[i]

Ordered linear search for k= 20

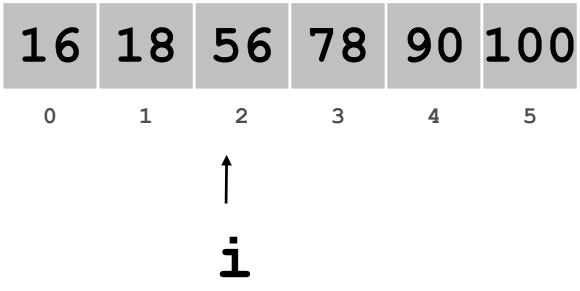
.16,18,56,78,90,100



i<n and k>L[i]

Ordered linear search for k= 20

.16,18,56,78,90,100



K<L[i], so key not found.

```
Procedure OrderedSearch(L, n, k)
    i=0;
    while ((i<n) and (k>L[i])) do
            i=i+1;
    endwhile
    if (k=L[i]) then {print ("Key found"); return (i); }
    else
            print ("Key not found");
    end OrderedSearch
```

Unordered Linear Search

.L={k1,k1,k3,…,kn}

.To search key k in the list

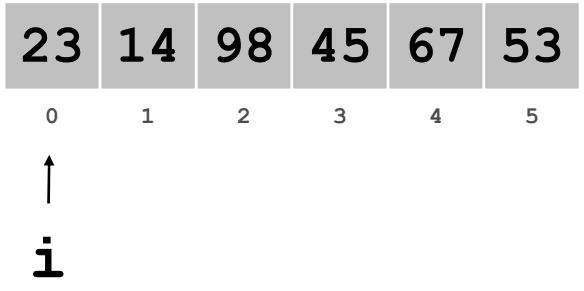
- .Compare key k with each element so long as k=ki
- .If k=ki then search is done
- .Otherwise search is unsuccessful

.L[0:5]={23,14,98,45,67,53}

- .Search for 98 and 110

Unordered linear search for k= 98

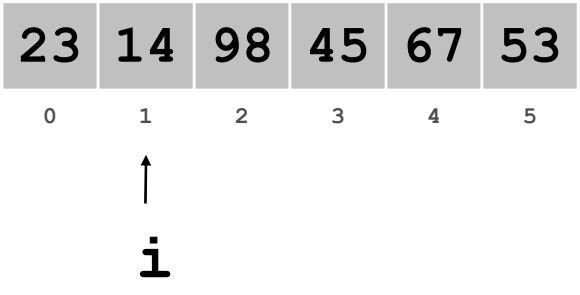
.23,14,98,45,67,53



i<n and k<>L[i]

Unordered linear search for k= 98

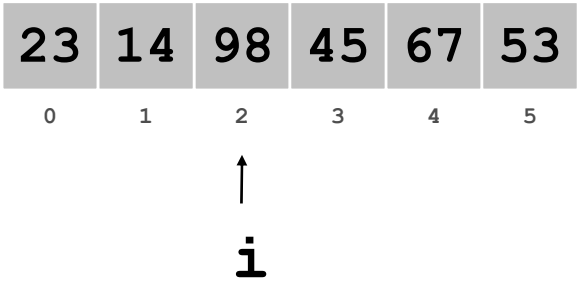
.23,14,98,45,67,53



i<n and k<>L[i]

Unordered linear search for k= 98

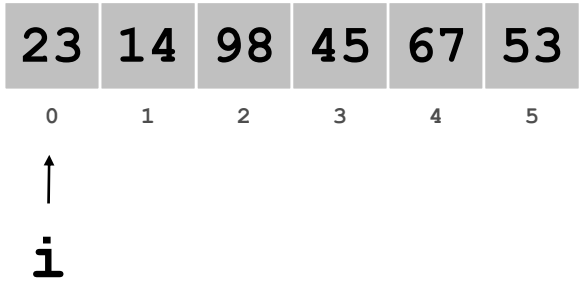
.23,14,98,45,67,53



K==L[i], so key is found.

Unordered linear search for k= 110

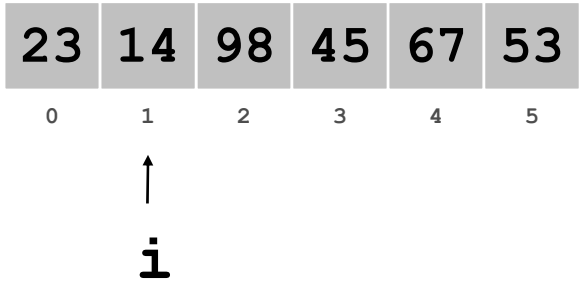
.23,14,98,45,67,53



i<n and k<>L[i]

Unordered linear search for k= 110

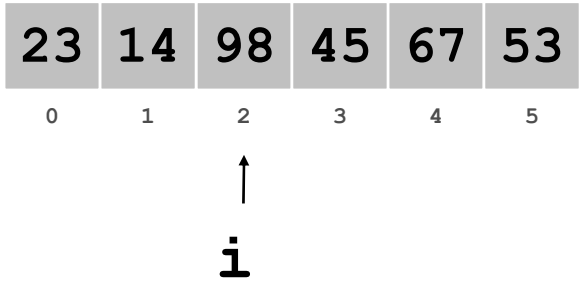
.23,14,98,45,67,53



i<n and k<>L[i]

Unordered linear search for k= 110

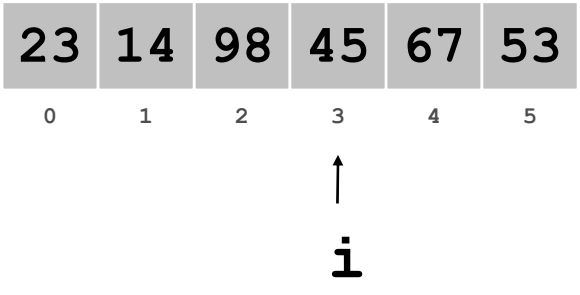
.23,14,98,45,67,53



i<n and k<>L[i]

Unordered linear search for k= 110

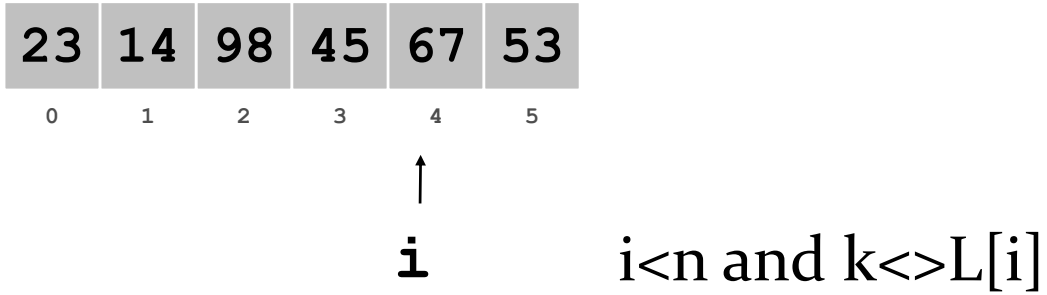
.23,14,98,45,67,53



i<n and k<>L[i]

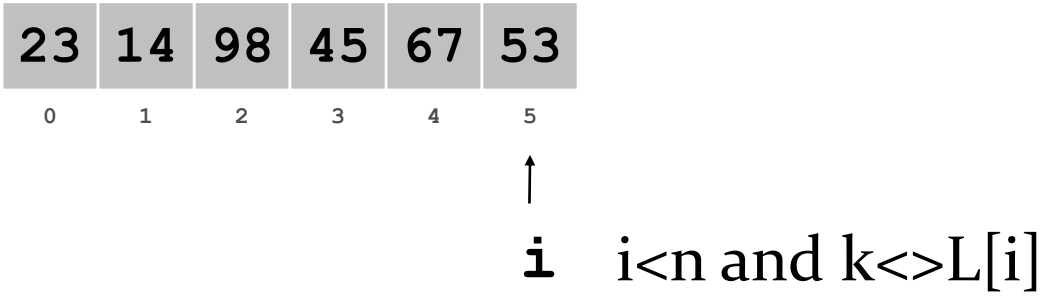
Unordered linear search for k= 110

.23,14,98,45,67,53



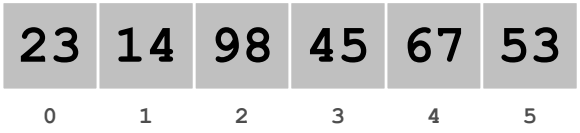
Unordered linear search for k= 110

.23,14,98,45,67,53



Unordered linear search for k= 110

.23,14,98,45,67,53



i

i>n, so key not found.

```
Procedure UnorderedSearch(L, n, k)
    i=0;
    while ((i<n) and (k<>L[i])) do
            i=i+1;
    endwhile
    if (k=L[i]) then {print ("Key found"); return (i); }
    else
            print ("Key not found");
    end UnorderedSearch
```

Transpose Sequential Search

.Self Organizing Sequential Search
.Searches a list of data items for a key, checking itself against the data items one at a time in a sequence
.If the key is found, then it is swapped with its predecessor and the search is termed successful.
.Favours faster search when one repeatedly looks for a key.

Search Key	List L before search	Number of element comparisons made during the search	List L after search
90	{34,21,89,45,12,90,76,62}	6	{34,21,89,45,**90,12**,76,62}
89	{34,21,89,45,90,12,76,62}	3	{34,**89,21**,45,90,12,76,62}
90	{34,89,21,45,90,12,76,62}	5	{34,89,21,**90,45**,12,76,62}
21	{34,89,21,90,45,12,76,62}	3	{34,**21,89**,90,45,12,76,62}
90	{34,21,89,90,45,12,76,62}	4	{34,21,**90,89**,45,12,76,62}
90	{34,21,90,89,45,12,76,62}	3	{34, **90**, **21,**89,45,12,76,62}

```
Procedure TransposeSequentialSearch(L, n, k)
    i=0;
    while ((i<n) and (k<>L[i])) do
            i=i+1;
    endwhile
    if (k=L[i]) then {print ("Key found");
                      swap(L[i], L[i-1]);
                      return (i); }
    else
            print ("Key not found");
    end TransposeSequentialSearch
```

Interpolation Search

.List must be ordered
.Like dictionary search – we turn sheaves of pages back or forth
.Suppose L={k1,k2,k3,…,kn} and k1<k2<k3<…<kn
.When it is known that k lies between Klow and khigh then the next element to be probed is (k-low)/(khigh-klow)

Implementation

.mid = i+((j-i).(k-ki)/(kj-ki))

.Comparison cases

.If (k=kmid) the the search is done.

.If (k<kmid) then continue the search in the sublist {ki,ki+1,ki+2,…,kmid-1}

.If (k>kmid) then continue the search in the sublist {kmid+1, kmid+2,…,kj)

Example

.L={24,56,67,78,79,89,90,95,99}
.Search for the keys 67 and 45

K	i	j	found	mid	< K=L[mid] >
67	1 1	9 4	False False True	1+((9-1)*(67-24)/(99-24)) = 5.58 = 5 3 Key found	67 < (L[5] = 79) 67 = (L[3] =67)
45					

K	i	j	found	mid	< K=L[mid] >
67	1 1	9 4	False False True	1+((9-1)*(67-24)/(99-24)) = 5.58 = 5 3 Key found	67 < (L[5] = 79) 67 = (L[3] = 67)
45	1 1 2 2	9 2 2 1	False False False False	1+((9-1)*(45-24)/(99-24))=3.24=3 1 2 Key not found	45 < (L[3] = 67) 45 > (L[1] = 24) 45 < (L[2] = 56)

Algorithm

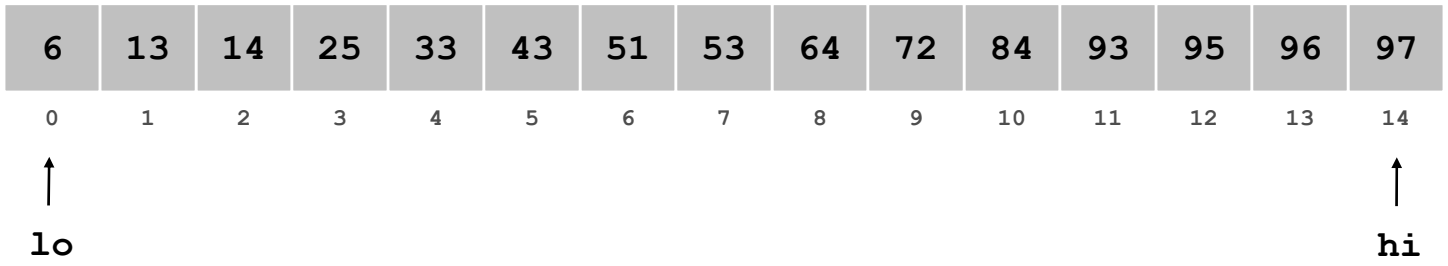
.Procedure Interpolation_Search(L, n, k)
i=1; j=n;
If (k<L[i]) or (k>L[j]) then {print (“Key not found”) ;
exit();}
while ((i<=j) and (found=false)) do
mid= i+(j-i).(k-L[i])/L[j]-L[i]);
case
: k=L[mid] : {found=true; print (“Key found”);
: k<L[mid] : j=mid-1;
: k>L[mid] : i=mid+1;
endcase
if (found=false) then print(“Key not found”);
end Interpolation_Search

Binary Search

- Locates a target value in a *sorted* array/list by successively eliminating half of the array from consideration.
- Binary search. Given value and sorted array a[], find index i such that a[i] = value, or report that no such index exists.
- Algorithm maintains $a[low] \leq value \leq a[high]$.

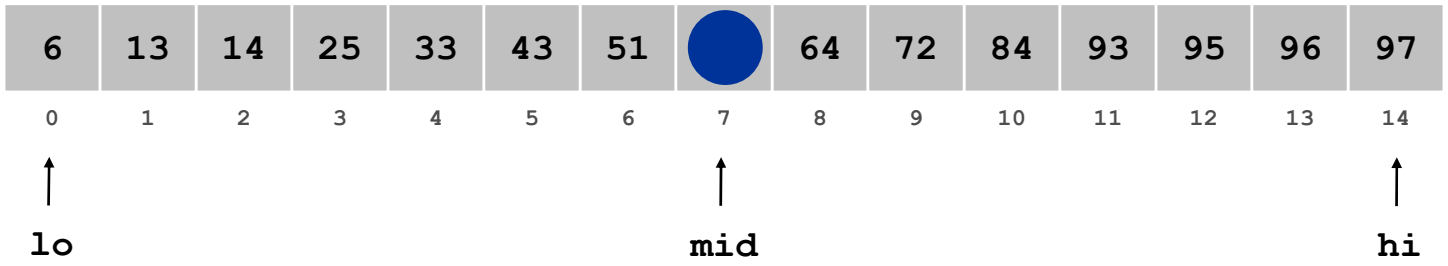
Binary Search

Ex. Binary search for 33.



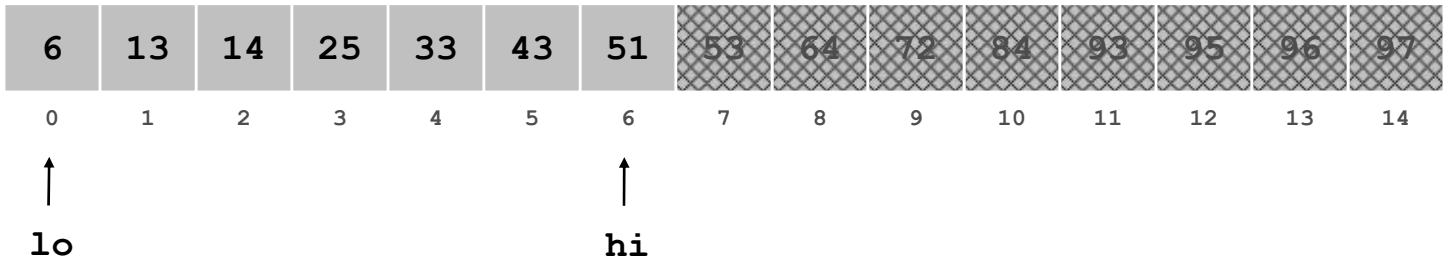
Binary Search

Ex. Binary search for 33.



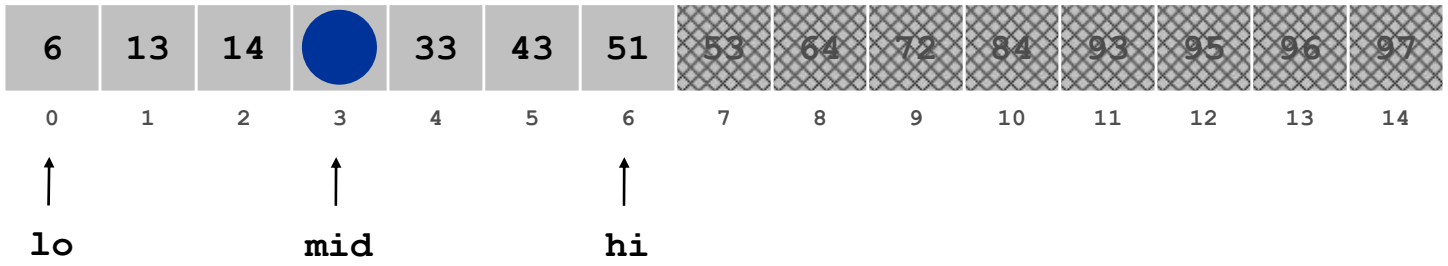
Binary Search

Ex. Binary search for 33.



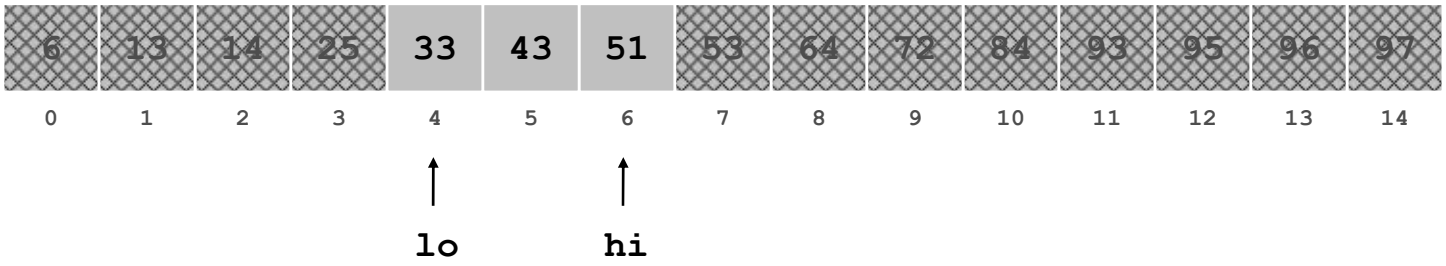
Binary Search

Ex. Binary search for 33.



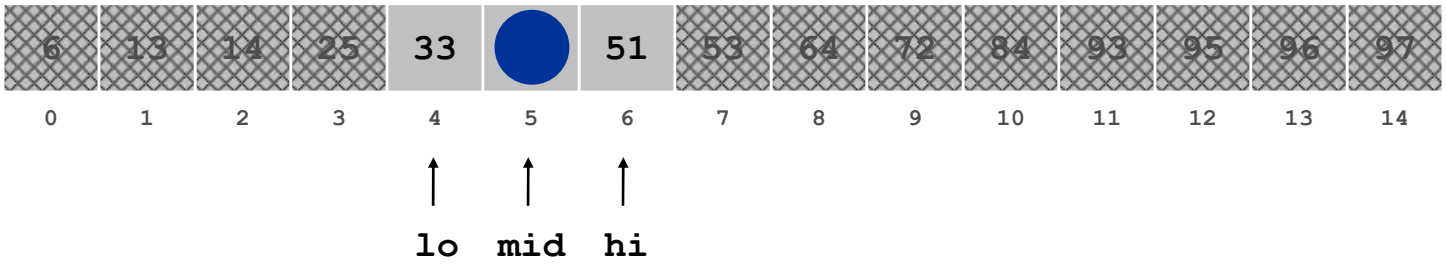
Binary Search

Ex. Binary search for 33.



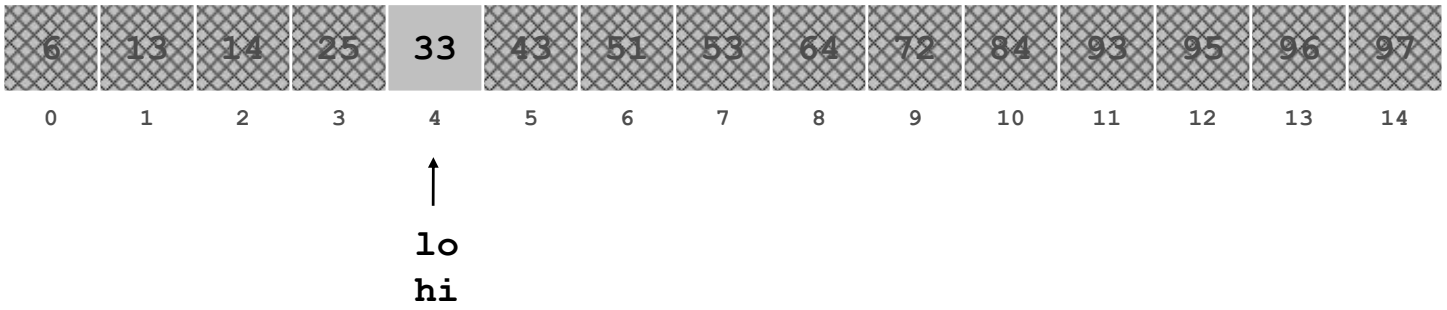
Binary Search

Ex. Binary search for 33.



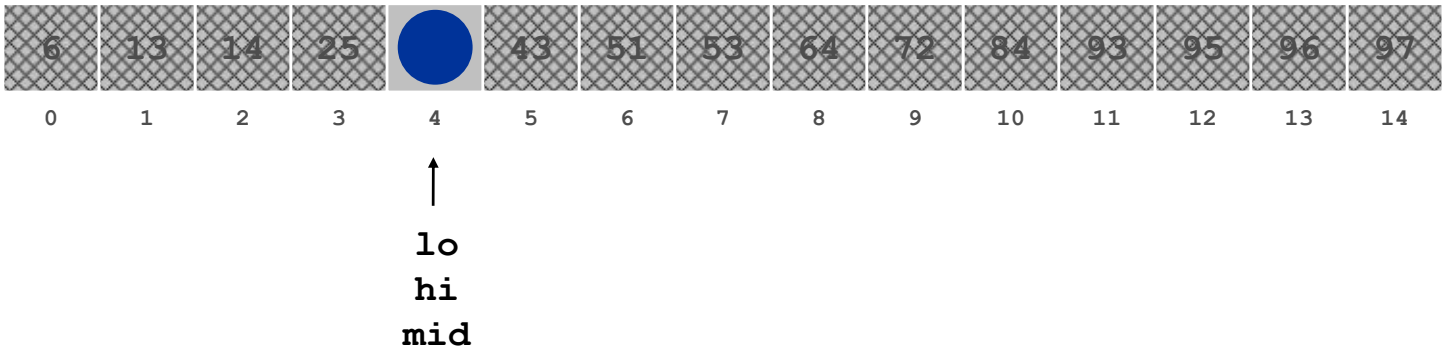
Binary Search

Ex. Binary search for 33.



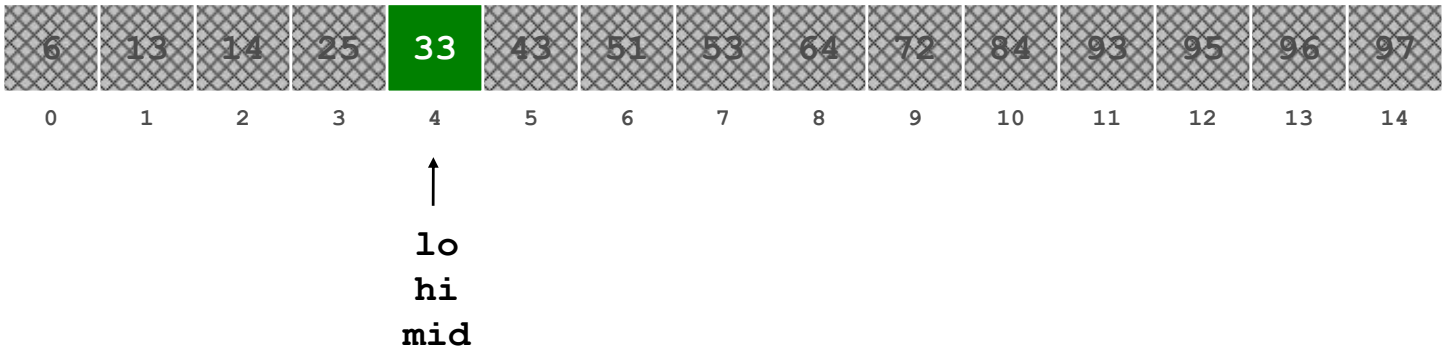
Binary Search

Ex. Binary search for 33.



Binary Search

Ex. Binary search for 33.



```
.Procedure Binary_Search(L, low, high, k)
if (low>high) then { binary_search=0;
                          print("Key not found"); exit(); }
else  {
        mid=(low+high)/2;
        case
        : k = L[mid] : {print("Key found");
                          binary_search=mid;  return L[mid]; }
        : k < L[mid] : binary_search =
                                Binary_Search(L, low, mid-1, k);
        : k > L[mid] : binary_search =
                                Binary_Search(L, mid+1, high, k);
        endcase
}
end Binary_Search
```

Fibonacci Search

.Works only on ordered list

.Fibonacci sequence {0, 1, 1, 2, 3, 5, 8, 13, 21,…}

.$F_0 = 0$

.$F_1 = 1$

.$F_i = F_{i-1} + F_{i-2}$

.Next element of comparison as dictated by Fibonacci number sequence

.Number of elements in the list is one less than a Fibonacci number i.e. $n=F_k - 1$

Example

Search Key K	< K = L[p] >	T	p	q	r	Remarks
434			13	8	5	n=20
	k>L[13] = 221	1	13	8	5	m=0 since F8 + 0 = n+1
	k>L[13] = 221		18	3	2	Since k>L[p], p=p+m
	k>L[18] = 401		20	1	1	Key is found
	k>L[20] = 536		19	1	0	
	k>L[19] = 434					
66			13	8	5	n=20
	K<L[13]=221	8	8	5	3	m=0
	k>L[8]=55	2	11	2	1	Since r=0, p is set to 0.
	k<L[11]=94	1	10	1	1	Key is not found.
	k<L[10]=84		9	1	0	
	k<L[9]=81					

Algorithm

```
Procedure Fibonacci_Search(L, n, k)
Obtain the largest fibonacci number Fk closest to n=n+1;
p  = Fk-1;
q = Fk-2;
r  Fk-3;
m = (n+1)-(p+q);
if (k > L[p]) then p = p+m;
found = false;
while ((p <> 0) and  (not found)) do
        case
        : k = L[p] :        { print("Key found");  found  =  true; }
        : k < L[p] :        if (r=0) then p=0;
                            else { p=p-r;  t=q; q=r; r=t-r; }
        : k > L[p] :        if (q=1) then p=0
                            else { p=p+r; q=q-r; r=r-q; }
        endcase
endwhile
if (found=false)then print("Key not found");
end Fibonacci_Search
```

Other Search Techniques

- Internal Searching
- External Searching
  - Tree Search
  - Graph Search
  - Indexed Sequential Search