Searching

Joy Mukherjee

Search Algorithms

Input:

A set of n integers $\{a_0, a_1, . . . , a_{n-1}\}$, and an integer key.

Output:

If key $= a_i$ for $0 <= i <= n - 1$, then the function prints "Key is found at i"; otherwise prints "Key is not 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.

Linear Search Example #0

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

element

Searching for -86.

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: found!

Linear Search Code

```
int LinearSearch(int a[], int n, int key)
{
    int i;
    for(i = 0; i < n; i++) {
            if(a[i] = = key) {
                    return i;
             }
    }
    return -1;
}
```

Linear Search: Worst Case

How long will our search take?

Worst case: key is the last element of the array.

Search takes a time proportional to the length of the array.

Computer scientists denote this as **O(n)**.

Binary Search

1. Initially, the 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 key, stop.
4. If key is less than middle data value, the new search region is the lower half of the data.
5. If key is greater than middle data value, the new search region is the higher half of the data.
6. Continue from Step 2.

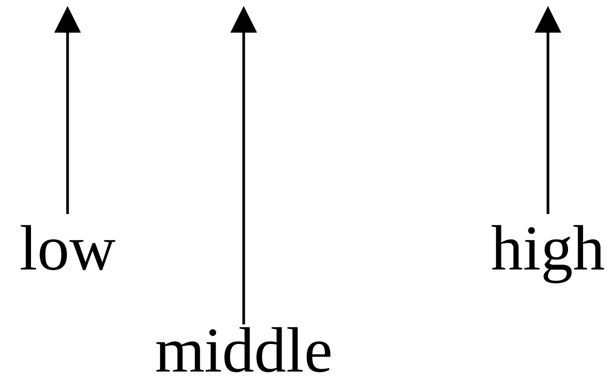
Binary Search Example

-86	-37	-23	0	5	18	21	64	97
low				middle				high

Searching for 18.

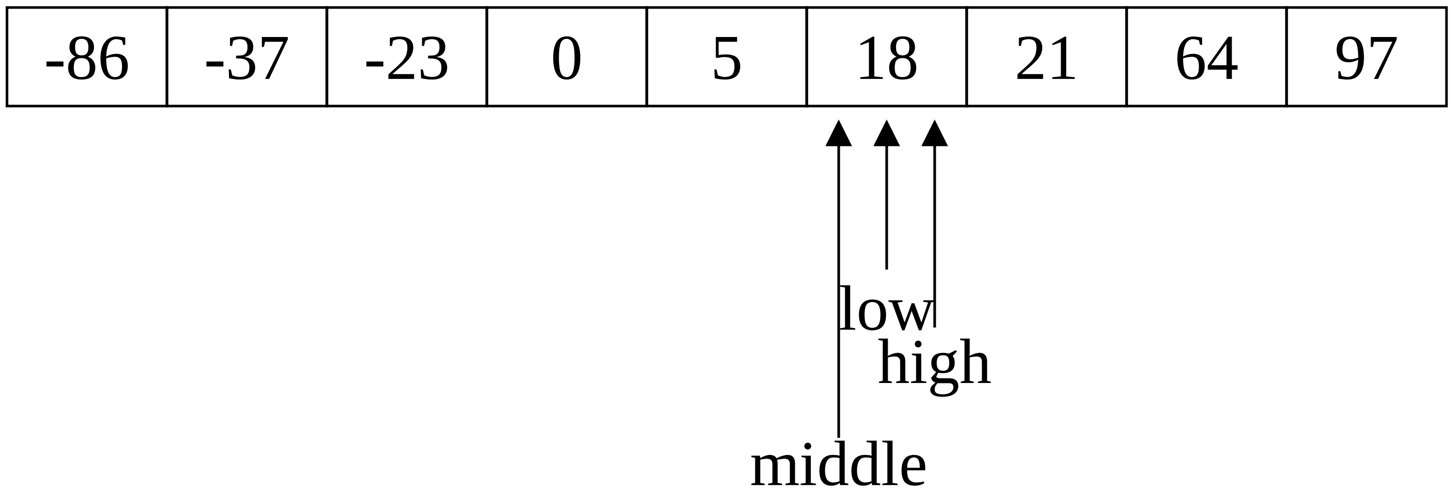
Binary Search Example

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



Searching for 18.

Binary Search Example



Searching for 18: found!

Binary Search Code

```
int BinarySearch (int a[], int n, int key)
{
 int low, high, mid;
 low = 0; high = n - 1;
 while(low <= high) {
  mid = (low + high) / 2;
  if(a[mid] = = key)
        return mid;
  else if(a[mid] > key)
        high = mid - 1;
  else
        low = mid + 1;
 }
 return -1;
}
```

Time Complexity: Binary Search

- $T(n)$ = Time to binary search for a key in array of $n = 2^k$ ^{i}ntegers, where $T(1) = 1$

  $T(n) = T(n/2) + c$

  $T(n/2) = T(n/2^2) + c$

  $T(n/2^2) = T(n/2^3) + c$

  …….

  $T(n/2^{k-1}) = T(n/2^k) + c$

  $T(n) = T(n/2^k) + c.\ k$ (Summation of k equations)

  $T(n) = T(1) + c.\log_2 n = O(\log_2 n)$

  $f(n) = O(g(n))$ iff for all $n>n0$, $c \in R^+$ $f(n) <= c.\ g(n)$

  $T(n) = O(\log_2 n)$ iff for all $n>0$, $c \in R^+$ $T(n) <= c.\ \log_2 n$