CSE 142
Computer Programming I

Linear & Binary Search

Concepts This Lecture

Searching an array

Linear search

Binary search

Comparing algorithm performance

Searching

Searching = looking for something

Searching an array is particularly common

- Goal: determine if a particular value is in the array

We'll see that more than one algorithm will work

Searching Problem: Specification

Let
- b be the array to be searched,
- n be the size of the array, and
- x be the value being searched for (the "target")

The question is, "Does x occur in b?"

If x appears in b[0..n-1], determine its index, i.e., find the k such that b[k]==x.

If x not found, return –1

Searching as a Function

The array b, the size n, and the target x are the parameters of the problem.

None of the parameters are changed by the function

Function outline:

```
int search (int b[ ], int n, int x) {
...
}
```

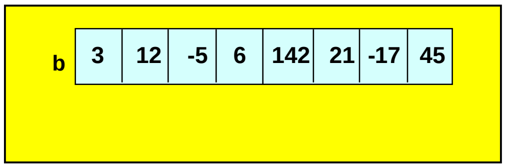
The details of the function depend upon the algorithm used.

Linear Search

Algorithm: start at the beginning of the array and examine each element until x is found, or all elements have been examined

```
int search (int b[ ], int n, int x) {
   int index = 0;
  while (index < n && b[index] != x)
    index++;
  if (index < n)
    return index;
  else return -1;
}
```

Linear Search



Test:

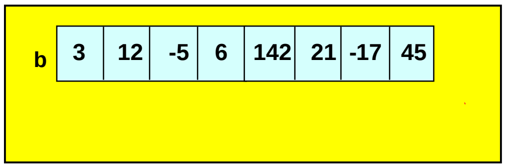
search(v, 8, 6)

```
while (index < n && b[index] != x)
    index++;
```

Found It!

P-7

Linear Search



Test:

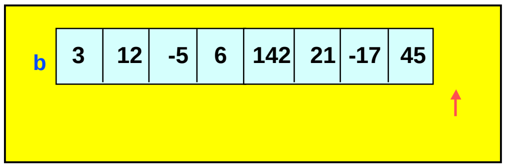
search(v, 8, 15)

```
while (index < n && b[index] != x)
    index++;
```

P-8

Ran off the end! Not found.

Linear Search



Note: The loop condition is written so *b[index]* is not accessed if *index>=n*.

while (index < n && b[index] != x)

P-9

(Why is this true? Why does it matter?)

Can we do better?

Time needed for linear search is proportional to the size of the array.

An alternate algorithm, "Binary search," works if the array is sorted

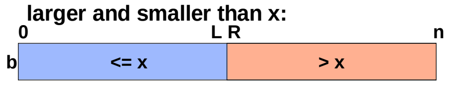
1. Look for the target in the middle.
2. If you don't find it, you can ignore half of the array, and repeat the process with the other half.

Example: Find first page of pizza listings in the yellow pages

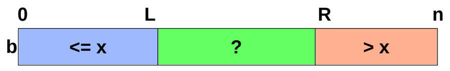
P-10

Binary Search Strategy

What we want: Find split between values larger and smaller than x:



Situation while searching:

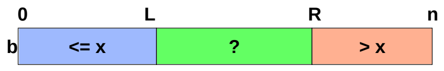


General strategy: shrink the green region; grow the blue and/or pink region

P-11

Binary Search Strategy

More precisely: at any time,



Values in b[0..L] <= x

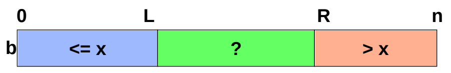
Values in b[R..n-1] > x

Values in b[L+1..R-1] are unknown, but sorted

P-12

Binary Search Strategy

Making progress:



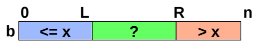
Step: Look at b[(L+R)/2]. Move L or R to the middle depending on test.

This shrinks the green region, and increases either the blue or the pink.

Binary Search

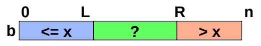
```
/* If x appears in b[0..n-1], return its location, i.e.,
   return k so that b[k]==x.  If x not found,
   return -1 */
int bsearch (int b[ ], int n, int x) {
    int L, R, mid;
    _________________ ;
    while ( _______________ ) {


   }
   _________________ ;
}
```



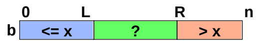
Binary Search

```
/* If x appears in b[0..n-1], return its location, i.e.,
   return k so that b[k]==x.  If x not found, return -1 */
int bsearch (int b[ ], int n, int x) {
    int L, R, mid;
    _________________ ;
    while ( _______________ ) {
       mid = (L+R) / 2;
       if (b[mid] <= x)
          L = mid;
       else  R = mid;
   }
   _________________ ;
}
```



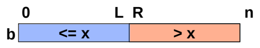
Loop Termination

```
/* If x appears in b[0..n-1], return its location, i.e.,
   return k so that b[k]==x.  If x not found,
   return -1 */
int bsearch (int b[ ], int n, int x) {
    int L, R, mid;
    _________________ ;
    while ( L+1 != R ) {
       mid = (L+R) / 2;
       if (b[mid] <= x)
          L = mid;
       else  R = mid;
   }
   _________________ ;
}
```



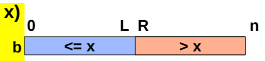
Initialization

```
/* If x appears in b[0..n-1], return its location, i.e.,
   return k so that b[k]==x.  If x not found, return -
    1 */
int bsearch (int b[ ], int n, int x) {
    int L, R, mid;
    L = -1; R = n;
    while ( L+1 != R ) {
       mid = (L+R) / 2;
       if (b[mid] <= x)  L = mid;
       else  R = mid;
   }
   _________________ ;
}
```



Return Result

```
/* If x appears in b[0..n-1], return its location, i.e.,
   return k so that b[k]==x.  If x not found, return -1 */
int bsearch (int b[ ], int n, int x) {
    int L, R, mid;
    L = -1; R = n;
    while ( L+1 != R ) {
       mid = (L+R) / 2;
       if (b[mid] <= x)  L = mid;
       else  R = mid;
   }
   if (L >= 0 && b[L] == x)
      return L
   else return -1;
}
```



Binary Search

	0	1	2	3	4	5	6	7
b	-17	-5	3	6	12	21	45	142

Test: bsearch(v,8,3);

```
L = -1; R = n;
while ( L+1 != R ) {
  mid = (L+R) / 2;
  if (b[mid] <= x)
    L = mid;
  else
    R = mid;
}
```

P-19

Binary Search

	0	1	2	3	4	5	6	7
b	-17	-5	3	6	12	21	45	142

Test: bsearch(v,8,17);

```
L = -1; R = n;
while ( L+1 != R ) {
  mid = (L+R) / 2;
  if (b[mid] <= x)
    L = mid;
  else
    R = mid;
}
```

P-20

Binary Search

	0	1	2	3	4	5	6	7
b	-17	-5	3	6	12	21	45	142

Test: bsearch(v,8,143);

```
L = -1; R = n;
while ( L+1 != R ) {
  mid = (L+R) / 2;
  if (b[mid] <= x)
    L = mid;
  else
    R = mid;
}
```

P-21

Binary Search

	0	1	2	3	4	5	6	7
b	-17	-5	3	6	12	21	45	142

Test: bsearch(v,8,-143);

```
L = -1; R = n;
while ( L+1 != R ) {
  mid = (L+R) / 2;
  if (b[mid] <= x)
    L = mid;
  else
    R = mid;
}
```

P-22

Is it worth the trouble?

Suppose you had 1000 elements

Ordinary search would require maybe 500 comparisons on average

Binary search

- after 1st compare, throw away half, leaving 500 elements to be searched.
- after 2nd compare, throw away half, leaving 250. Then 125, 63, 32, 16, 8, 4, 2, 1 are left.
- After at most 10 steps, you're done!

What if you had 1,000,000 elements??

P-23

How Fast Is It?

Another way to look at it: How big an array can you search if you examine a given number of array elements?

# comps	Array size
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128
…	…
11	1,024
…	…
21	1,048,576

P-24

Time for Binary Search

Key observation: for binary search: size of the array *n* that can be searched with *k* comparisons: $n \sim 2^k$

Number of comparisons *k* as a function of array size *n*: $k \sim \log_2 n$

This is fundamentally faster than linear search (where $k \sim n$)

P-25

Summary

Linear search and binary search are two different algorithms for searching an array

Binary search is vastly more efficient

- But binary search only works if the array elements are in order

Looking ahead: we will study how to sort arrays, that is, place their elements in order

P-26

QOTD: Multiple Madness

Sometimes more than one array element will match what we are looking for.

Sometimes we want to get all of those matches.

Try this: ***design a function that searches an array of students and "returns" everyone who got a score of 100 on the midterm***

P-27