CSE 142
Programming I

Linear & Binary Search

5/11/00

K1-1

Searching

- **Searching = looking for something**
- **Searching an array is particularly common**
  - Goal: determine if a particular value is in the array
- **If the array is unsorted:**
  - start at the beginning and look at each element to see if it matches

5/11/00

K1-2

Linear Search

```
/* If x appears in a[0..n-1], return its index, i.e.,
   return k so that a[k]==x.  If x not found, return -1 */
int search (int a[ ], int n, int x) {
   int index = 0;
   while (index < n && a[index] != x)
     index++;
   if (index < n)
     return index
   else return -1;
}
```

5/11/00

K1-3

Linear Search

v | 3 | 12 | -5 | 6 | 142 | 21 | -17 | 45 |

- **Test:**

    search(v, 8, 12)
    search(v, 8, 15)

- **Note: Condition in while relies on short-circuit evaluation of && (i.e., a[index] might not be defined if index>=n).**

5/11/00

K1-4

Can we do better?

- **"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**

5/11/00

K1-5

Binary Search Strategy

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

```
0                  L R              n
a | <= x          |     > x        |
```
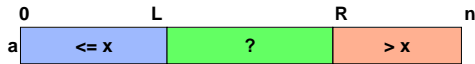
- **Situation while searching**

```
0        L            R           n
a | <= x |    ?      |    > x     |
```

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

5/11/00

K1-6

K.1

Binary Search Strategy

- **More precisely**

```
   0        L        R        n
a [ <= x  |    ?    |   > x  ]
```

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

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

Values in a[L+1..R-1] are unknown

Binary Search

/* If x appears in a[0..n-1], return its location, i.e.,
return k so that a[k]==x. If x not found, return -1 */

```
int bsearch (int a[ ], int n, int x) {
   int L, R, mid;
   _____ ;
   while ( _____ ) {



   }
   _____ ;
}
```

Binary Search

/* If x appears in a[0..n-1], return its location, i.e.,
return k so that a[k]==x. If x not found, return -1 */

```
int bsearch (int a[ ], int n, int x) {
   int L, R, mid;

   _____ ;
   while ( _____ ) {
      mid = (L+R) / 2;
      if (a[mid] <= x)
         L = mid;
      else  R = mid;
   }
   _____ ;
}
```

Loop Termination

/* If x appears in a[0..n-1], return its location, i.e.,
return k so that a[k]==x. If x not found, return -1 */

```
int bsearch (int a[ ], int n, int x) {
   int L, R, mid;

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

Initialization

/* If x appears in a[0..n-1], return its location, i.e.,
return k so that a[k]==x. If x not found, return -1 */

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

Return Result

/* If x appears in a[0..n-1], return its location, i.e.,
return k so that a[k]==x. If x not found, return -1 */

```
int bsearch (int a[ ], int n, int x) {
   int L, R, mid;
   L = -1; R = n;
   while ( L+1 != R ) {
      mid = (L+R) / 2;
      if (a[mid] <= x)  L = mid;
      else  R = mid;
   }
   if (L >= 0 && a[L] == x) return L
   else return -1;
}
```

K.1

Binary Search

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

v

• **Test: bsearch(v,8,3);**

Binary Search

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

v

• **Test: bsearch(v,8,17);**

Binary Search

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

vec

• **Test: bsearch(vec,8,143);**

Binary Search

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

vec

• **Test: bsearch(vec,8,-143);**

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??