

CSE 142 Computer Programming I

Recursive Binary Search

© 2000 UW CSE

W2-1

Recursive Binary Search

Binary Search
Recursive Algorithm
Iteration vs. Recursion

W2-2

A Familiar Search Algorithm

Binary search works if the **array is sorted**

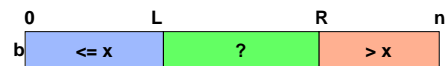
1. Look for the target in the middle.
2. If you don't find it, you can ignore half of the array, and repeat the process with the other half.

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

Let's solve this again, recursively

W2-3

Binary Search Strategy



Values in $b[0..L] \leq x$
Values in $b[R..n-1] > x$
Values in $b[L+1..R-1]$ are unknown

$$\text{mid} = (L + R) / 2$$

Compare $b[\text{mid}]$ and x

Replace either L or R by mid

W2-4

Recursive Binary Search

Key idea – do a little bit of work, and make recursive call to do the rest

Binary search has value restricted to a range

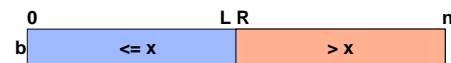
Look at midpoint, and decide which half of the range is of interest

Use binary search to find value in reduced range. **Recursion.**

W2-5

Base Case

No remaining unknown area:



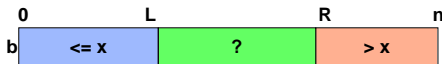
We recognize the base case when

$$L+1 == R$$

W2-6

Recursive Case

Situation while searching



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

Each recursive call is given L and R as parameters

W2-7

The Search Function

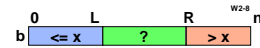
The original search problem called for a function with 3 parameters:

```
int bsearch (int b[], int n, int x);
```

Our recursive approach requires L and R as parameters

Let's call this function by a different name:

```
int bsearchHelper (int b[], int L, int R, int x) {  
    ...  
}
```



Recursive Search Function

```
int bsearchHelper (int a[], int L, int R, int x) {  
    int mid;  
    if (L+1 == R) /*base case*/  
        return L;  
  
    mid = (L+R)/2; /*recursive case*/  
    if (a[mid] <= x)  
        L = mid;  
    else  
        R = mid;  
    return bsearchHelper(a, L, R, x);  
}
```

W2-9

Initialization Dilemma

The proper initial values for L and R are:

L = -1;

R = n;

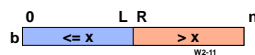
These initializations cannot be inside the bsearchHelper function, since L and R are parameters!

W2-10

Termination Dilemma

After the base case is reached, we must make the final decision about what value to return: -1 if not found, L if found

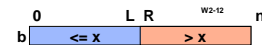
This decision cannot be placed inside bsearchHelper!



W2-11

Solution: A "Wrapper" Function

1. It sets the recursion in motion
Calls the recursive function with the correct initial parameters
2. After the recursion completes,
determines the correct final action



W2-12

Non-Recursive Wrapper

```
int bsearch (int a[], int asize, int x) {
    int L = -1;
    int R = asize;

    L = bsearchHelper (a, L, R, x); /*kickoff*/

    if (a[L] == x)    /* final */
        return L;
    else
        return -1;
}
```

W2-13

Trace

-17 -5 3 6 12 21 45 142

```
bSearch(a, 8, 5)                -1
  bsearchHelper(a, -1, 8, 5)      2
    bsearchHelper(a, -1, 3, 5)    2
      bsearchHelper(a, 1, 3, 5)   2
        bsearchHelper(a, 2, 3, 5) 2
```

W2-14

Iteration vs. Recursion

Turns out *any* iterative algorithm can be reworked to use recursion instead (and vice versa).

There are programming languages where recursion is the only choice(!)

Some algorithms are more naturally written with recursion

But *naïve* applications of recursion can be inefficient

W2-15

When to Use Recursion?

Problem has one or more simple cases

These have a straightforward nonrecursive solution, and:

Other cases can be redefined in terms of problems that are closer to simple cases

By repeating this redefinition process one gets to one of the simple cases

W2-16

Recursion Wrap-up

Recursion is a programming technique

It works because of the way function calls and local variables work

Recursion is more than a programming technique

W2-17