CSC 212: Data Structures and Abstractions

Recursion

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2020

Recursion

Recursion

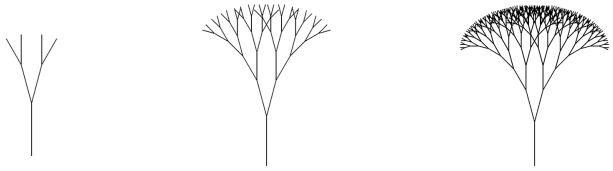
- Solve a task by reducing it to smaller tasks (of the same structure)
- Technically, a recursive function is one that **calls itself**
- General form:
  - **base case**
    - solution for a **trivial case**
    - it can be used to stop the recursion (prevents "*stack overflow*")
    - every recursive algorithm needs at least one base case
  - **recursive call(s)**
    - divide problem into **smaller instance(s)** of the **same structure**

General Form

```
function() {

    if (this is the base case) {

        calculate trivial solution

    } else {

        break task into subtasks
        solve each task recursively
        merge solutions if necessary

    }
}
```

Why recursion?

- Can we live without it?
  - yes, you can write "any program" with arrays, loops, and conditionals
- However …
  - some formulas are explicitly recursive
  - some problems exhibit a natural recursive solution



https://courses.cs.washington.edu/courses/cse120/17sp/labs/11/tree.html


```
int sum_array(int *A, int n) {
    // base case
    if (n == 1) {
        return A[0];
    }

    // solve sub-task
    int sum = sum_array(A, n-1);

    // return sum
    return A[n-1] + sum;
}
```


Recursion call tree

```
int sum_array(int *A, int n) {
    // base case
    if (n == 1) {
        return A[0];
    }

    // solve sub-task
    int sum = sum_array(A, n-1);

    // return sum
    return A[n-1] + sum;
}
```


Example: power of a number

$$b^n = \underbrace{b \cdot b \cdot \ldots \cdot b}_{n \text{ times}}$$

```
double power(double x, int n) {
    // base case
    if (n == 0) {
        return 1;
    }
    // recursive call
    return x * power(x, n-1);
}
```

Recursion call tree

```
double power(double x, int n) {
    // base case
    if (n == 0) {
        return 1;
    }
    // recursive call
    return x * power(x, n-1);
}
```


Is this faster?

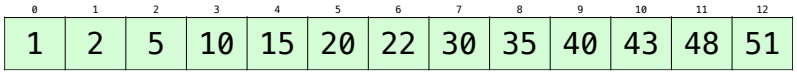
```
double power(double x, int n) {
    if (n == 0) {
        return 1;
    }
    double half = power(x, n/2);
    if (n % 2 == 0) {
        return half * half;
    } else {
        return x * half * half;
    }
}
```


Recursion call tree

```
double power(double x, int n) {
    if (n == 0) {
        return 1;
    }
    double half = power(x, n/2);
    if (n % 2 == 0) {
        return half * half;
    } else {
        return x * half * half;
    }
}
```


Binary Search

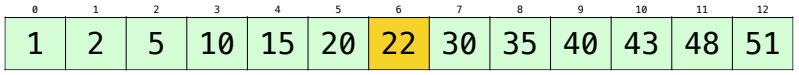
Binary Search



low high

k = 48?

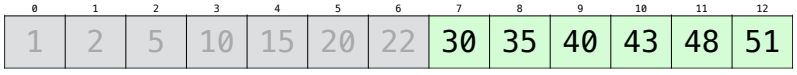
Binary Search



low mid high

k = 48?

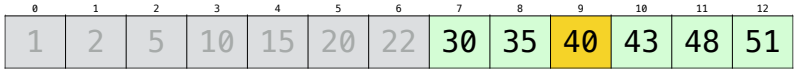
Binary Search



low high

k = 48?

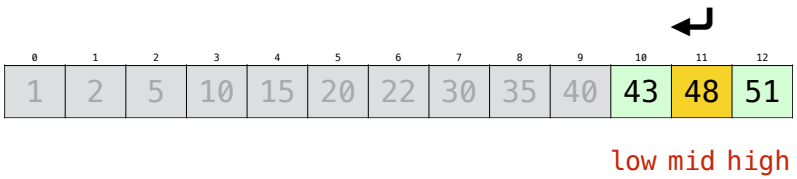
Binary Search



low mid high

k = 48?

Binary Search



k = 48?


```c
int bsearch(int *A, int lo, int hi, int k) {
    // base case
    if (hi < lo) {
        return NOT_FOUND;
    }
    // calculate midpoint index
    int mid = lo + ((hi-lo)/2);
    // key found?
    if (A[mid] == k)
        return mid;
    // key in upper subarray?
    if (A[mid] < k)
        return bsearch(A, mid+1, hi, k);
    // key is in lower subarray?
    return bsearch(A, lo, mid-1, k);
}
```


Recursion Tree (binary search)


```c
int bsearch(int *A, int lo, int hi, int k) {
    // base case
    if (hi < lo) {
        return NOT_FOUND;
    }
    // calculate midpoint index
    int mid = lo + ((hi-lo)/2);
    // key found?
    if (A[mid] == k)
        return mid;
    // key in upper subarray?
    if (A[mid] < k)
        return bsearch(A, mid+1, hi, k);
    // key is in lower subarray?
    return bsearch(A, lo, mid-1, k);
}
```


Complexity? Best-case, Worst-case, Average-case?

Example: find peak in unimodal arrays

Unimodal arrays

- An array is (**strongly**) **unimodal** if it can be split into an increasing part followed by a decreasing part

1	2	5	16	20	18	17	16	15	12	10	8	5

How to efficiently find the max?

1	2	5	16	20	18	17	16	15	12	10	8	5

Find the **max** (strongly unimodal)

1	2	5	8	15	20	22	20	15	12	10	8	5
1	2	5	8	15	20	22	20	15	12	10	8	5
1	2	5	8	15	20	22	20	15	12	10	8	5
1	2	5	8	15	20	22	20	15	12	10	8	5
1	2	5	8	15	20	22	20	15	12	10	8	5
1	2	5	8	15	20	22	20	15	12	10	8	5
1	2	5	8	15	20	22	20	15	12	10	8	5
1	2	5	8	15	20	22	20	15	12	10	8	5

Find the **max** (strongly unimodal)

- Recursion Tree?

- Complexity? Best-case, Worst-case, Average-case?

Unimodal arrays

- An array is (**weakly**) **unimodal** if it can be split into a nondecreasing part followed by a nonincreasing part

1	2	5	5	15	20	22	22	35	38	13	8	5

How to efficiently find the max?

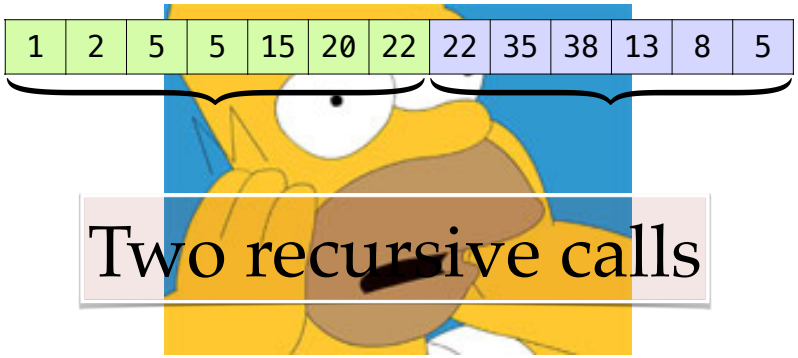
1	2	5	5	15	20	22	22	35	38	13	8	5

Find the **max** (weakly unimodal)

1	2	5	5	15	20	22	22	35	38	13	8	5

1	2	5	5	15	20	22	22	35	38	13	8	5

1	2	5	5	15	20	22	22	35	38	13	8	5



Find the **max** (weakly unimodal)

- Recursion Tree?

- Complexity?