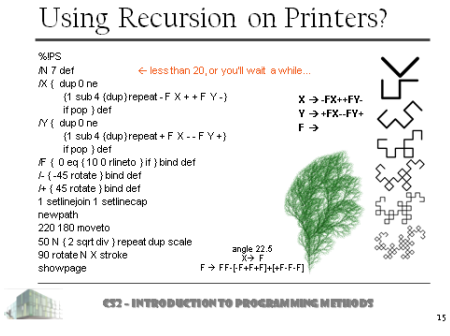
CS 2

Introduction to Programming Methods

Last Week: Convex Hull

Introduced recursion

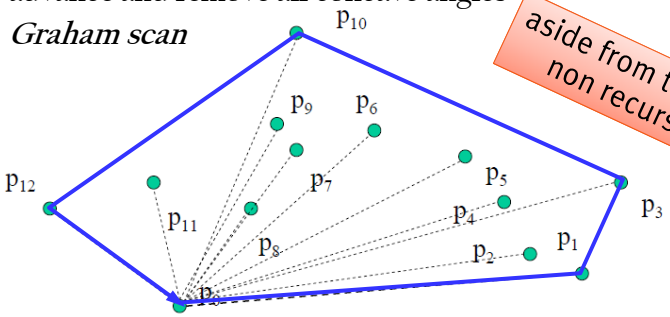
- and first algorithm for sorting in n log n



Back to Convex Hull

Got n log n sorting algorithm, now what?

- find lowest point
- order points by angle
- advance and remove all concave angles
- *Graham scan*



aside from the sort, non recursive!

Recursive Convex Hull?

You guessed it...

- split points in two sets
- find convex hulls of each
- then "zipper" them up
  - a bit tedious, so left as home exercise.

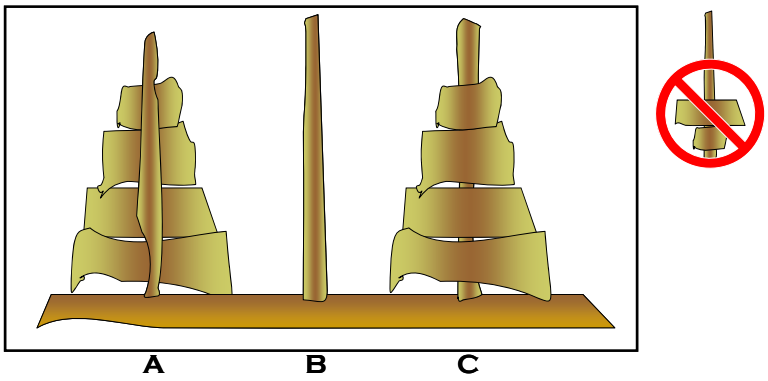
Conclusion: Recursion is a great concept

- help solving lots of problems
  - but has its own problems as well...

Towers of Hanoi

Move 4 disks from pole A to pole C

- one disk at a time, never on top of a smaller disk



Visual Solution

Find a recursive way to do that for n rings...

Matrix multiplication?

Normal way

Complexity?

```
public static double[][] mult(double a[][], double b[][])
{       // early checks
        if (a.length == 0) return new double[0][0];
        if (a[0].length != b.length) return null; //invalid dimensioms

        // a is mxn, b is nxp
        int n = a[0].length; int m = a.length; int p = b[0].length;

        double ans[][] = new double[m][p];

        for(int i = 0;i < m;i++){
                for(int j = 0;j < p;j++){
                        for(int k = 0;k < n;k++){
                                ans[i][j] += a[i][k] * b[k][j]; }
                }
        }
        return ans;
}
```


Strassen Idea

For 2x2 matrices, can factor out operations

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{bmatrix}$$

- with:
$$\begin{aligned} P_1 &= A_{11}(B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12})B_{22} \\ P_3 &= (A_{21} + A_{22})B_{11} \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21})(B_{11} + B_{12}) \end{aligned}$$
- not worth it!! 7 x and 18 + instead of 8 x and 4 +. Right?
- but… for larger matrices, + is/was much faster than x
- using Strassen idea recursively leads to $O(n^{2.81})$
  - why? $T(n) = 7\,T(n/2) + O(n^2)$ [for nxn matrices]

Recursion Not a Panacea

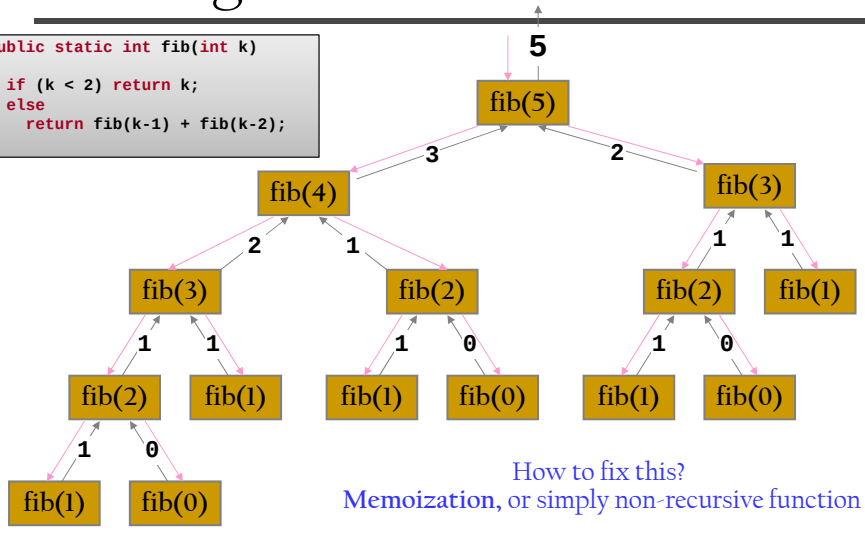
Recursion is admittedly fun... but not always a good idea

- sometimes just too slow
  - example: Fibonacci!
    - factorial was ok
    - but Fib has lots of redundant computations, lots of memory use to keep track of ongoing computations

factorial(5)
factorial(4)
factorial(3)
factorial(2)
factorial(1)
return 1
return 2*1 = 2
return 3*2 = 6
return 4*6 = 24
return 5*24 = 120

Tracing A Fibonacci Call

```
public static int fib(int k)
{
  if (k < 2) return k;
  else
    return fib(k-1) + fib(k-2);
}
```



How to fix this?
Memoization, or simply non-recursive function

Recursion Not a Panacea

Recursion is admittedly fun... but not always a good idea

- sometimes just too slow
  - example: Fibonacci!
    - factorial was ok
    - but Fib has lots of redundant computations, lots of memory use to keep track of ongoing computations
- sometimes unnecessary
  - example: binary search

factorial(5)
factorial(4)
factorial(3)
factorial(2)
factorial(1)
return 1
return 2*1 = 2
return 3*2 = 6
return 4*6 = 24
return 5*24 = 120

Search

You want to find a value in a **sorted** list

- like a list of existing clients or students
  - example: where is "3" in [1,3,6,9,10]
    - in second position
  - example: where is "4" in [1,3,6,9,10]
    - not found
- complexity?
  - at most n comparisons
  - will recursion help?
    - sure

```
public int Search(int[] arr, int target)
{
        for (int i=0; i<arr.size; i++)
        {
                if (target == arr[i]) { return i; }
        } //for
        return -1; //target not found
}
```

Recursive *Binary* Search

Can we exploit the sortedness of the list?

- look at the middle of the list…
- if target higher than the mid element, we saved quite a bit of comparisons!
  - divide and conquer all over again

```
public int binSearch(int[] arr, int lower, int upper, int x)
{
        if (lower > upper) {// empty interval
                return -1;  }// base case
        int mid = (lower + upper) / 2;
        if(arr[mid] == x){ return mid; } // second base case
        else if(arr[mid] < x)  { return binSearch(arr, mid + 1, upper, x); }
        else // arr[mid] > target
                        return binSearch(arr, lower, mid - 1, x);
}
```

How Much Better?

Complexity is easy to analyze

- best case?
  - lucky guess = O(1)
- worst case?
  - each time, eliminate ½
  - so O($\log_2$(N))

	Linear	Binary
N	*N*	$\log_2(N)$
10	10	4
100	100	7
1,000	1000	10
10,000	10,000	14
100,000	100,000	17
1,000,000	1,000,000	20
10,000,000	10,000,000	24

Do We Really Need Recursion?

Not in this case… Just unroll the recursion

```
public int binSearch(int[] arr, int target){
        int lower = 0;
        int upper = arr.length - 1;
        while (lower <= upper){
                int mid = (lower + upper) / 2;
                if (target == arr[mid]) {
                        return mid;
                } else if (target > arr[mid]) {
                        lower = mid + 1;
                } else { //target < arr[mid]
                        upper = mid - 1;
                }
        } //while
        return -1; //target not found
}
```

Recursion Pitfalls

Typical issues you may encounter

- infinite loop
  - forgot base case…
- memory running out
  - "stack overflow" (we'll see about that later)
- obfuscation
  - what is this function?

```
public static int mystery(int a, int b) {
  if (b == 0)    return 0;
  if (b % 2 == 0) return mystery(a+a, b/2);
  return mystery(a+a, b/2) + a;
}
```

```
public static void bad(int a, int b) {
  if (a != b) {
    int m = (a + b) / 2;
    bad(a, m); StdOut.println(m); bad(m, b);
  }
}
```

```
public static void bad(int a, int b) {
  if (a != b) {
    int m = (a + b) / 2;
    bad (a, m - 1); StdOut.println(m); bad(m + 1, b);
  }
}
```