CS 2

Introduction to Programming Methods

Last Week: Sorting

Introduced ways to sort, and complexity
- no optimal way, though

Before then...

Assuming a comparison-based method...
what is the best complexity we can get?
- for n numbers, how many permutations?
  - $n \cdot (n-1) \cdot (n-2) \dots 2 \cdot 1 = n!$
- the sorted list is only *one* of these n! combos
- each comparison kills half the permutations
  - e.g., for 1,2,3: (123), (132), (213), (231), (312), (321)
  - if A[1]>A[2], then we are left with (213), (132), (231)
  - like the "20 questions" game... (can find one out of 2^{20})
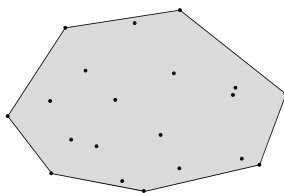- so sorting is $\Omega(\log_2 n!) = \Omega(n \log n)$ (stirling's)

Introducing Convex Hull

Given a set $S = \{p_1, p_2, ..., p_N\}$ of points in 2D, convex hull H(S) is the smallest convex polygon in the plane that contains all of S.

Q: find the ordered list of points from S defining the convex hull H(S)

Why is CH Interesting

Many reasons to look at it a bit
- extends sorting, in a way
  - in fact, easy to prove in 1D that the two problems share the same lower bound
    - from $\{x_i\}$, compute CH of $\{x_i, x_i^2\}$
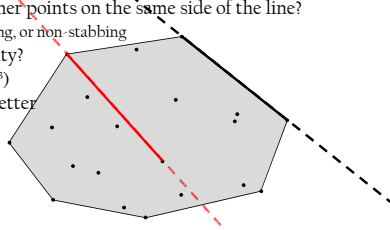- neat in its own right in Comp. Geo.
- results you can see

Convex Hull Construction

A first algorithm
- test every pair of points
  - are all other points on the same side of the line?
    - stabbing, or non-stabbing
  - complexity?
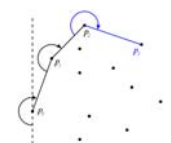   - in $O(n^3)$
  - let's do better

Gift Wrapping

Improving on previous algorithm
- find a point on convex hull
  - by going thru points and finding leftmost one (O(n))
- then find non-stabbing line (linear time)
- then repeat from new point
- Worst case?
  - n^2
- More precisely, O(nh)
  - output sensitive!

Can We Do Even Better

Yes, if we can sort in $O(n \log n)$...

So, let's revisit the kooky idea of "dividing" up the problem in smaller problems...

- actually pretty convenient in many cases
  - example: factorial
    - $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$
    - $n! = n \cdot (n-1)!$
  - or Fibonacci numbers
    - $Fib(n) = Fib(n-1) + Fib(n-2)$
    - $Fib(0)=0, Fib(1)=1$.

```
static int factorial(int k)
{ If (k == 0) return 1;   // 0! = 1
  else return k*factorial(k-1);   }
```

```
static int Fib(int k)
{ If (k < 2) return k;   //Fib(0/1)
  else return Fib(k-1)+Fib(k-2); }
```
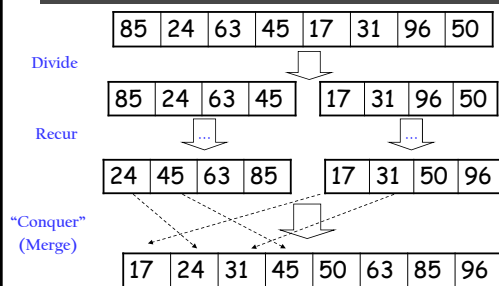

Recursive Sorting?

Could we:
- Divide input in half (trivial)
- Recursively sort each half (just two calls)
- Merge two halves together
  - this last step is less trivial, but...
  - how hard is it to merge two sorted lists?
    - compare "head elements" repeatedly → $O(n)$

Base case:
- $Sort(\{k\}) = \ldots \{k\}$.

Execution

Divide

Recur

"Conquer" (Merge)

Complexity (I)

So, did we gain anything in doing so?
- let's call $T(n)$ the computational complexity
- $T(n) = T(n/2) + T(n/2) + n$
  - 2 recursive calls, and one $O(n)$ merge
  - and $T(1) = 1$

Complexity (II)
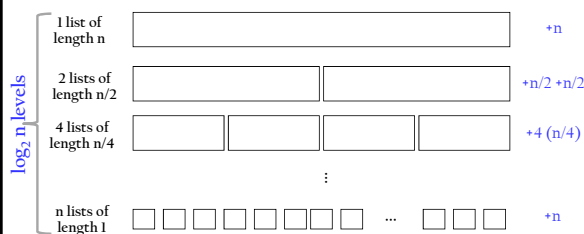
Let's trace a full execution

Complexity (III)

So, did we gain anything in doing so?
- let's all $T(n)$ the computational complexity
- $T(n) = T(n/2) + T(n/2) + n$
  - 2 recursive calls, and one $O(n)$ merge
  - and $T(1) = 0$
- $T(n) = O(n \log n)$

Code

```
public class MergeSorter
{
    private static int[] a, b;   // auxiliary array b

    public static void sort(int[] input)
    {
        a=input;
        int n=a.length; b=new int[n];
        mergesort(0, n-1);
    }

    private static void mergesort(int lo, int hi)
    {
        if (lo<hi)
        {
            int m=(lo+hi)/2;
            mergesort(lo, m);
            mergesort(m+1, hi);
            merge(lo, m, hi);
        }
    }
}
```

```
    private static void merge(int lo, int m, int hi)
    {
        int i, j, k;

        // copy both halves of a to auxiliary array b
        for (i=lo; i<=hi; i++)
            b[i]=a[i];

        i=lo; j=m+1; k=lo;
        // copy the next-greatest elmt, each time
        while (i<=m && j<=hi)
            if (b[i]<=b[j])
                a[k++]=b[i++];
            else
                a[k++]=b[j++];

        // copy remaining elmts of first half (if any)
        while (i<=m)
            a[k++]=b[i++];
    }
}  // end class MergeSorter
```

Recursion, More Generally

Common recursive strategies
- Handle first and/or last, recur on rest
- Divide in half, recur on halves
- process, recur on updated state

Complexity of divide-and-conquer methods
- use of master theorem [CLRS]
    - $T(n) = a\,T(n/b) + f(n)$
- and variants

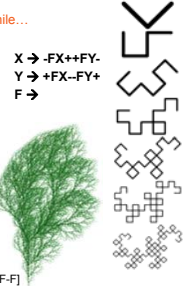## Recursion & Fractals

Snowflakes in "turtle graphics"
- F means "draw forward", + means "turn left by α", and - means "turn right by α"
- set $\alpha = 60°$, and start with F
- then apply rule: F $\rightarrow$ F+F--F+F

Using Recursion on Printers?

```
%!PS
/N 7 def          ← less than 20, or you'll wait  a while…
/X {  dup 0 ne
        {1 sub 4 {dup} repeat - F X + + F Y -}
        if pop } def
/Y {  dup 0 ne
        {1 sub 4 {dup} repeat + F X - - F Y +}
        if pop } def
/F {  0 eq { 10 0 rlineto } if } bind def
/- { -45 rotate } bind def
/+ { 45 rotate } bind def
1 setlinejoin 1 setlinecap
newpath
220 180 moveto
50 N { 2 sqrt div } repeat dup scale
90 rotate N X stroke
showpage
```

X $\rightarrow$ -FX++FY-
Y $\rightarrow$ +FX--FY+
F $\rightarrow$

angle 22.5
X $\rightarrow$ F
F $\rightarrow$ FF-[-F+F+F]+[+F-F-F]

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*