Recursion

recursion

Binary Search

Given a sorted array of size N

Find the value K in O ($\log_2$ N) time

1, 3, 4, 6, 8, 10, 13, 22, 40, 50

Start at middle:

if (K < middle value) search left half

if (K > middle value) search right half

can divide a list in half only $\log_2$ N times! *hence O ($\log_2$ N)*

Binary Search

```
public int binSearch (int[ ] a, int k) {
  int left = 0;
  int right = 0;
  while (left <= right) {
     int mid = (left + right) / 2;
     if (a[mid] == k)   return mid;
     if  ( a[mid] < k )  left = mid+1;    //left half starts here now
     else    right = mid-1;               //right half ends here now
  }
  return -1;   //K is not found
}
```

1, 3, 4, 6, 8, 10, 13, 22, 40, 50

Recursion

A process that solves larger problems by dividing the problem into similar, but simpler problems

solve that problem, by solving that problem!

Humor: Google "Recursion": *Did you mean Recursion?*

Russian *Matryoshka* dolls

N Factorial: = n * (n-1 factorial)

Fibonacci sequence: F(n) = F(n-1) + F(n-2)

sort (alist);

sort (Llist) && sort (Rlist) …

Divide and conquer

Recursion

Approaches to large problem:

Find the simple, **base-case**(s)
trivial cases that can simply return the answer

Find relationship from large case to simpler cases (Each sub-division of problem moves closer to the base-case)

Sometimes recursion is more clear than explicit looping *(this point is arguable)*

Examples

```
public int factorial (int n) {
   if (n == 0)  return 1;           //base case
   return   n * factorial(n-1);   //recursive call
}
```

Does it fulfill the requirements?

base case exists *(in this case, 0! = 1 by definition)*

each call moves closer to the base case

Examples

```
public int fibonacci (int n) {
    if (n == 0)  return 0;          //base case
    if (n == 1)  return 1;          //base case
    return   fibonacci(n-1) + fibonacci(n-2);   //recursive calls
}
```

Note multiple base cases

Recursion

Wikipedia:

Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem

Greatest Common Divisor

Euclid (ancient Greek guy)

$$gcd(x, y) = \begin{cases} x & \text{if } y=0 \\ gcd(y, remainder(x,y)) & \text{if } y > 0 \end{cases}$$

```
public int gcd (int x, int y)
{
      if (y == 0)  return x;
      return   gcd (y,  x % y);
}
```


Sierpinski Triangle

Draw an equilateral triangles within equilateral triangles…

```
sierpinski (a, b, c)
   find midpoints of each side  (divide by 2)
      midAB,  midBC,  midAC
   sierpinski (a, midAB, midBC)
   sierpinski (midAB, b, midBC)
   sierpinski (midAC, midBC, C)
```

Requires a base case - because theoretically there will always be a divide by 2

Usually a value signifying the "order" and each recursive call reduces the order by 1. Base case is order == 0

Could also make base case be when midpoint length < S

Sierpinski Applet

Show applet

Towers of Hanoi

Three pegs; N disks of increasing diameter

Start with all disks on left peg in order of decreasing size.
Cannot place larger diameter onto small diameter
Move all disks one-at-a-time

Goal: move all disks to right peg
- Challenging problem:
- 1) What is the base case?
  - move one disk from x to y
- 2) What is the recursive problem?
  - move (n-1) disks from **x** to **h** ; then move disk **n** to **y**
  - move (n-1 disks from **h** to **y**)

BEWARE
Time complexity is 2^{n-1}
Extreme growth rate!!!!
End of the world legend

Binary Search

Recursive Edition

```
public int binSearch (int [ ] a,  int k)        //"starter"  method (common for recursion)
    return  binSearch (a, k, 0, a.length-1); //begin the recursive
public int binSearch (int[] a, int k, int left, int right)
    if (left > right)  return   -1;
    int mid = (left + right) / 2;
    if (a[mid] == k)  return  mid;
    if (a[mid] < k)
       return binSearch (a, k, mid+1, right);
    else
       return binSearch (a, k, left, mid-1);
```

1, 2, 3, 5, 8,12,14,15, 20, 26

QuickSort

Naturally recursive algorithm

if list size > 1

- Pick an element (pivot) from the list
- Partitioning: Reorder so that all values less than pivot are on the left; all values greater than pivot on the right
- sorted = qsort (lefthalf) + pivot + qsort (righthalf)

base case: list size < 2

QuickSort

Partitioning

```
choose pivot & move to top of array  a[hi]
left = lo;  right = hi-1;
while (left < right)
   move left up finding un-sorted value
   move right down until find un-sorted value
   if (left < right)  swap a[left] and a[right]
swap pivot and a[right]
```


QuickSort

Worst case?

What about all equal values?

Best case?

Average runtime O(n $\log_2$ n)

slightly behind linear! (this is *very* good)