1. Recursion
   1. Sample code: power functions
      1. int mypow(int base, int exp) {
         1. if (exp == 0)
            1. return 1;
         2. return base \* mypow(base, exp – 1)
            1. //e = b \* b^(e-1)
            2. //Call to the function itself
      2. } //Recursion works!
   2. Sample code 2: Fibonacci sequence
      1. //f(*n*) = f(*n*-1)+f(*n*-2)
      2. int fib(int n) {
         1. if (n == 1 || n == 2) return 1;
         2. return (fib(n-1) + fib(n-2));
      3. }//Not very efficient when compared to the for loop method
      4. //Branching pattern of 2
      5. //Large values are obtained by adding 1+1+…
   3. Sample code: Factorials
      1. int factorial(int n) {
         1. if (n == 0) return 1;
         2. return (n \* factorial(n-1));
      2. } //O(*n*) = ϕ*n*
   4. Sample code: Tip chart
      1. void tip\_chart(int start, int end, double tip\_rate) {
         1. if (start < = end) {
            1. tip\_chart(start, mid-1, tip\_rate);
            2. printf(“$%d.00\t$%.21f\n”, start, start \* tip\_rate);
            3. tip\_chart(mid+1, end, tip\_rate);
         2. //print from beginning, and from a point in the middle
   5. Sample code: sum of digits up to n
      1. int sumDigits(int n) {
         1. if (n < 10) return n;
         2. return ((n % 10) + sumdigits(n/10)); //Takes off units digit; rest of digits are n/10
      2. }
   6. Sample code: convert bases
      1. void convert(int n, int b) {
         1. if (n > 0) {
            1. convert(n/b, b);
            2. printf(“%d”, n % b);
         2. }
   7. General code frameworks
      1. function(parameters) {
         1. if(**base case**: easy problem)
            1. return (easy answer);
         2. return (make recursive call and use easy answer);
         3. }
      2. function(parameters){
         1. if(base case)
            1. //do nothing;
         2. <other cases>
         3. }
2. Comparison to other methods
   1. Most things can be done without recursion.
   2. Some recursion is very efficient; other recursion is not.
   3. Recursion generally uses more parameters.
3. Why/How recursion works
   1. **Call stack**: limited set of “orders”.
      1. mypow(3,4) becomes superseded by mypow(3,3)
      2. mypow(3,3) becomes superseded by mypow(3,2)
      3. …
      4. mypow(3,1) becomes superseded by mypow(3,0)
      5. mypow(3,0) returns 1 and pops off the stack
      6. mypow(3,1) returns (3 \* 1) and pops off the stack
      7. …
      8. mypow(3,4) returns (3 \* 3 \* 3 \* 3)
4. Recursion & the binary search
   1. Code
      1. binsearch(array, int low, int high, intsearchval) {
         1. if (low > high)
            1. return 0;
         2. int mid = (low+high)/2;
         3. if (searchval < array[mid])
            1. return binsearch(low, mid-1, searchval, array);
         4. else if (searchval > array[mid])
            1. return binsearch(mid+1, high, searchval, array);
         5. else
            1. return 1;
      2. }