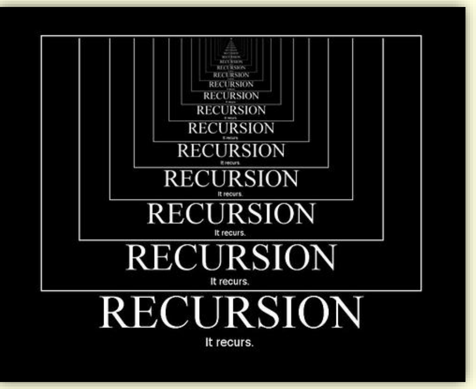
CS 241
Data Organization using C
Section 4.10: Recursion

Instructor: **Joel Castellanos**
e-mail: joel@unm.edu
Web: http://cs.unm.edu/~joel/
Office: Farris Engineering Center
Room 2110



9/25/2019

1

Quiz: Section 4.10

```c
1)   #include <stdio.h>
2)
3)   void intToStr(int n)
4)   { if (n / 10)
5)     { intToStr(n);
6)     }
7)     putchar(n % 10 + '0');
8)   }
9)
10)  void main(void)
11)  { intToStr(342);
12)  }
```

This program will cause a segmentation fault because:

a) Line 5 is a function call to the function that line 5 is inside.

b) Line 5 uses recursion.

c) Line 5 uses recursion and **`intToStr`**, is not declared recursive.

d) Line 5 is a recersive call to a function that returns **`void`**.

e) The function **`intToStr(n)`**, can call **`intToStr(n)`** without changing the value of n.

2

2

Recursion Example: intToStr

```
1)  #include <stdio.h>
2)
3)  void intToStr(int n)
4)  { if (n / 10)
5)    {
6)      intToStr(n / 10);
7)    }
8)    putchar(n%10 + '0');
9)  }
10)
11) void main(void)
12) { intToStr(342);
13) }
```

Line 12: **intToStr(342)**
    Line 3: n = 342
    Line 6: **intToStr(34)**
        Line 3: n = 34
        Line 6: **intToStr(3)**
            Line 3: n = 3
            Line 8: **put '3'**
        Return: end of line 6
        Line 8: **put '4'**
    Return: end of line 6
    Line 8: **put '2'**
Return: end of line 12

3

3

Recursion

4

4

Fibonacci Numbers: Recursive Definition

Recursive Definition:

Recurrence relation: $f_{n+2} = f_{n+1} + f_n$
Base case: $f_1 = 1$, $f_0 = 1$

Example: $f_7 = f_6 + f_5$

$f_6 = f_5 + f_4$

$f_5 = f_4 + f_3$

$f_4 = f_3 + f_2$

$f_3 = f_2 + f_1$

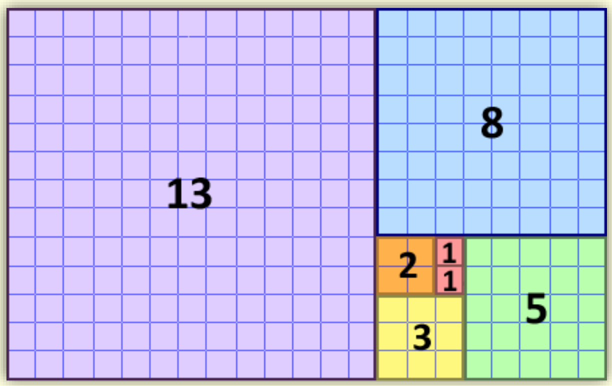
$f_2 = \boxed{f_1 + f_0}$ ← Base Case

Fibonacci Numbers and Tiling

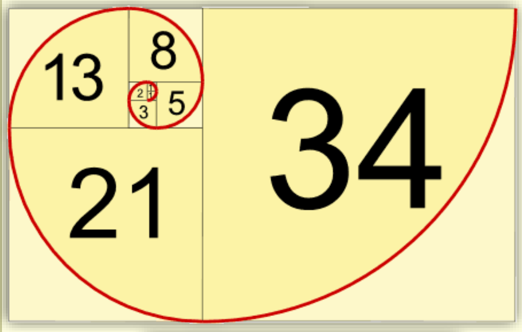
Fibonacci Number Sequence:

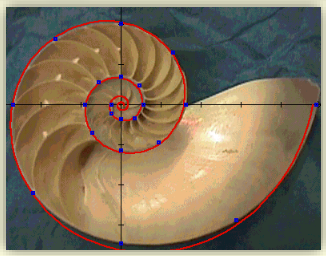
1 2 3 5 8 13 21 34 55 89 144
233 377 610 987 1597 2584 4181 ......

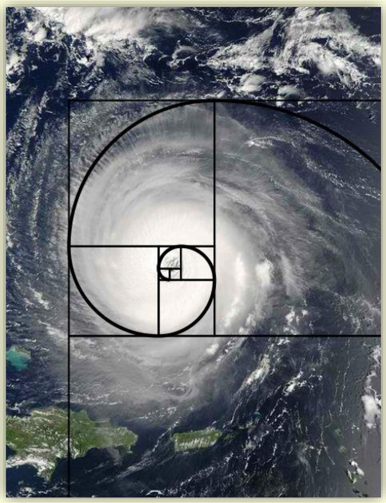
A *Fibonacci Tiling* is a tiling of the plane with squares whose sides have length equal to successive Fibonacci numbers.



Fibonacci Spiral



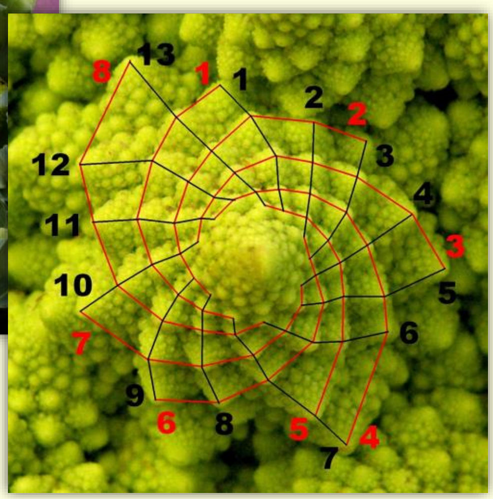




7

7

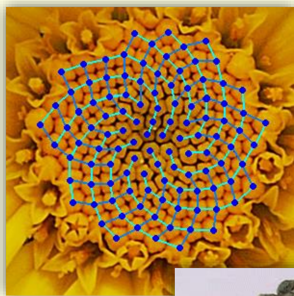
Fibonacci Spiral in Romanesco Broccoli



8

8

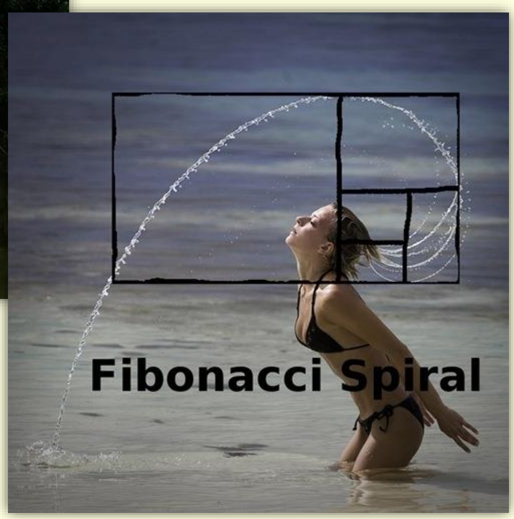
Fibonacci Spiral in Nature



9

9

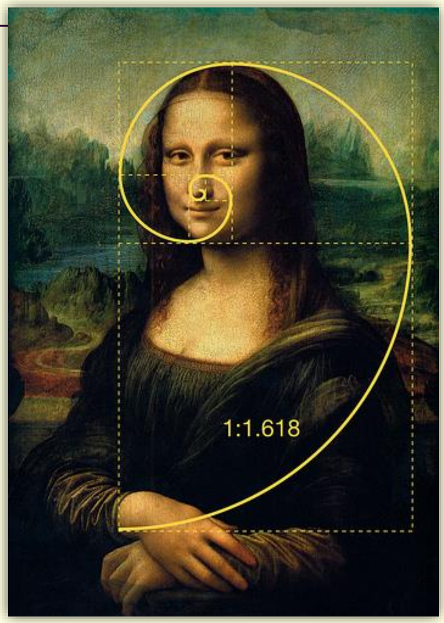
Fibonacci Spiral in Physics



10

10

Fibonacci Spiral in Art



11

11

Fibonacci Series and the Golden Ratio

```c
void main(void)
{
  long f0 = 1;
  long f1 = 1;

  int i;
  for (i=2; i<26; i++)
  {
    double ratio = (double)f1 / (double)f0;
    printf("%6ld / %ld\t=%.9f\n", f0,f1,ratio);

    long f2 = f1 + f0;
    f0 = f1;
    f1 = f2;
  }
}
```

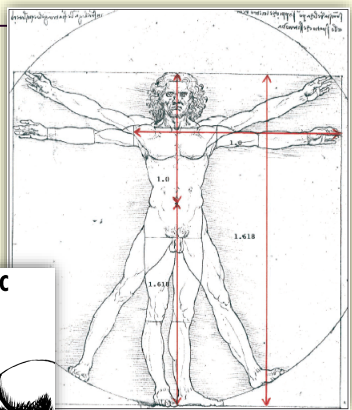
12

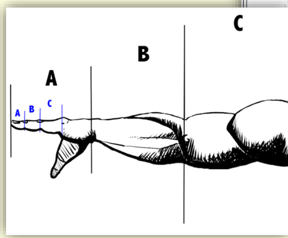
12

Fibonacci Series and the Golden Ratio

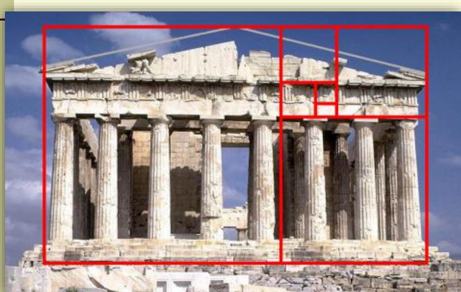
1 / 1	=1.000000000
1 / 2	=2.000000000
2 / 3	=1.500000000
3 / 5	=1.666666667
5 / 8	=1.600000000
8 / 13	=1.625000000
13 / 21	=1.615384615
21 / 34	=1.619047619
34 / 55	=1.617647059
55 / 89	=1.618181818
89 / 144	=1.617977528
144 / 233	=1.618055556
233 / 377	=1.618025751
377 / 610	=1.618037135
610 / 987	=1.618032787
987 / 1597	=1.618034448
1597 / 2584	=1.618033813
2584 / 4181	=1.618034056
4181 / 6765	=1.618033963
6765 / 10946	=1.618033999
10946 / 17711	=1.618033985
17711 / 28657	=1.618033990
28657 / 46368	=1.618033988

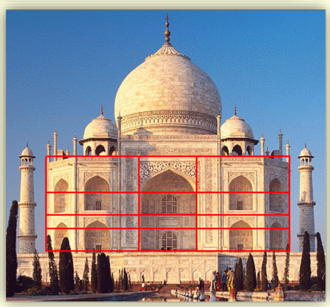
46368 / 75025 =1.618033989

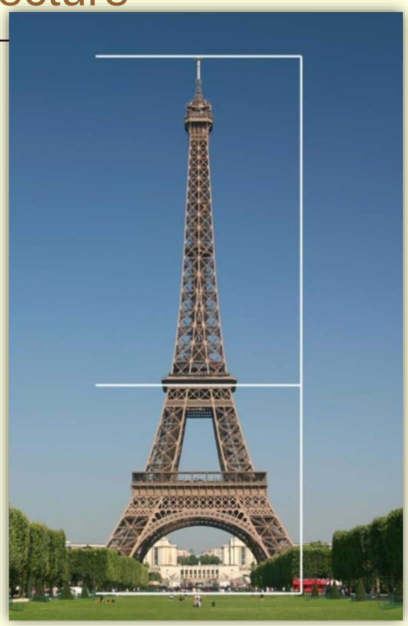




Golden Ratio in Architecture







Fibonacci Sequence by Recursion

```c
int fibonacci(int n)
{
  if (n==1 || n==2) return 1;
  return fibonacci(n-1) + fibonacci(n-2);
}

void main(void)
{
  printf("%d\n", fibonacci(20));
}
```

When a function calls itself recursively, each invocation gets a ***separate copy*** of all automatic variables

15

15

What Some C Coders Find Beautiful

```c
int f(int x){if(x<2)return 1;return f(x-1)+f(x-2);}
```

51 characters including spaces.

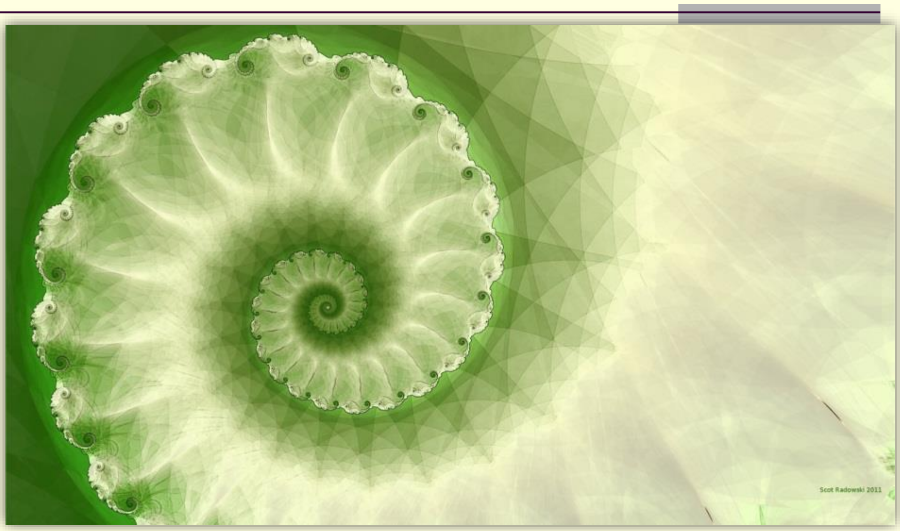
Minimalist yet complex: built from layering many circular ceramic sections within a single form.

-- Matthew Chambers

16

16

What C Program Reproduces this?



17 Extra Credit: Write, demo and explain a C program to draw this.

17

Quicksort Algorithm

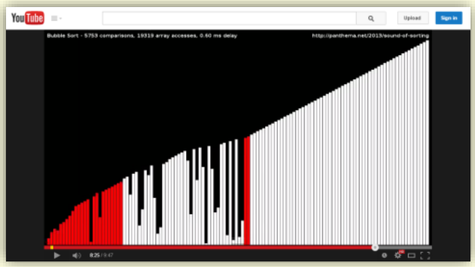
- Quicksort is a ***divide and conquer*** algorithm for sorting the elements of an array.
- Performance: Average Case: $O(n \log n)$, Worst Case: $O(n^2)$.
- Given an array, one element is chosen and the others are partitioned into two subsets:
    1) Those less than the partition element and
    2) Those greater than or equal to it.
- The same process is then recursively applied to each of the two subsets.
- When a subset has fewer than two elements, it doesn't need any sorting: this stops the recursion.

18

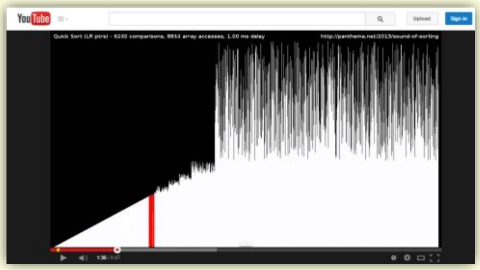
18

The Sound Sorting Video (*n*=500)

Bubble Sort



Quicksort



19

19

Quicksort: main()

```
#include <stdio.h>
int arraySize;
int level = 0;

void main(void)
{
  int v[] = {23, 13, 82, 33, 51, 17, 45, 75, 11, 27};

  int arraySize = sizeof(v)/sizeof(int);
  printf("arraySize=%d\n", arraySize);

  quicksort(v, 0, arraySize-1);
}
```

20

20

Quicksort: Helper Function swap

```
void swap(int v[], int i, int j)
{
  int c = v[i];
  v[i] = v[j];
  v[j] = c;
}


void swap(int* v, int i, int j)
{
  int c = v[i];
  v[i] = v[j];
  v[j] = c;
}
```

21

21

Quicksort: Helper Function printArray

```
void printArray(int code, int v[], int left, int right)
{
  int i=0;
  if (code < 0) printf("  Done%2d [", -level);
  else printf("Level=%2d [",level);

  for(i=0; i<arraySize; i++)
  {
    if (i<left || i>right)
    { printf("   ");
    }
    else
    { printf("%2d ", v[i]);
    }
  }
  printf("]\n");
}
```

22

22

```
void quicksort(int v[], int left, int right)
{ level++;
  printArray(level, v, left, right);

  int i, last;
  if (left < right)
  { swap(v, left, (left+right)/2);
    last = left;
    int num2 = v[left];
    for (i=left+1; i <= right; i++)
    { if (v[i] < v[left])
      { last++;
        swap(v, last, i);
      }
    }

    swap(v, left, last);
    quicksort(v, left, last-1);
    quicksort(v, last+1, right);
    printArray(-level, v, left, right);
  }
  level--;
}
```

Nothing to sort if segment is fewer than two elements.

Move the partition item out of the partition range

Restore partition.

23

23

Quicksort Output Trace

```
Level= 1 [23 13 82 33 51 17 45 75 11 27 ]
Level= 2 [27 13 33 23 17 45 11          ]
Level= 3 [11 13 17                      ]
Level= 4 [11                            ]
Level= 4 [      17                      ]
  Done 3 [11 13 17                      ]
Level= 3 [            33 45 27          ]
Level= 4 [            27 33             ]
Level= 5 [                              ]
Level= 5 [               33             ]
  Done 4 [            27 33             ]
Level= 4 [                              ]
  Done 3 [            27 33 45          ]
  Done 2 [11 13 17 23 27 33 45          ]
Level= 2 [                        82 75 ]
Level= 3 [                        75    ]
Level= 3 [                              ]
  Done 2 [                        75 82 ]
  Done 1 [11 13 17 23 27 33 45 51 75 82 ]
```

24

24

Quicksort: Quiz

```
1)  void quicksort(int v[], int left, int right)
2)  { int i, last;
3)    printArray(v, left, right);
4)    if (left >= right) return;
5)
6)    swap(v, left, (left+right)/2);
7)    last = left;
8)    for (i=left+1; i <= right; i++)
9)    {
10)     if (v[i] < v[left])
11)     { last++;
12)       swap(v, last, i);
13)       printArray(v, left, right);
14)     }
15)   }
16)
17)   swap(v, left, last);
18)   quicksort(v, left, last-1);
19)   quicksort(v, last+1, right);
20) }
```

If the output from line 3 is:

`[75 62 33 41 24]`

Then what would be the output the next time line 13 is reached?

a) [33 62 75 24 41]

b) [33 24 75 41 62]

c) [33 62 24 41 75]

d) [33 41 24 63 75]

e) [33 24 41 62 75]

Quicksort Runtime Performance

- Average Case: $O(n \log n)$
- Worst Case: $O(n^2)$ (same as bubble sort).
- What starting arrangement results in the worst case?
- How likely is it that uniformly distributed random data will hit or come near to hitting the worst case?
- Can the algorithm be improved by avoiding the worst case?