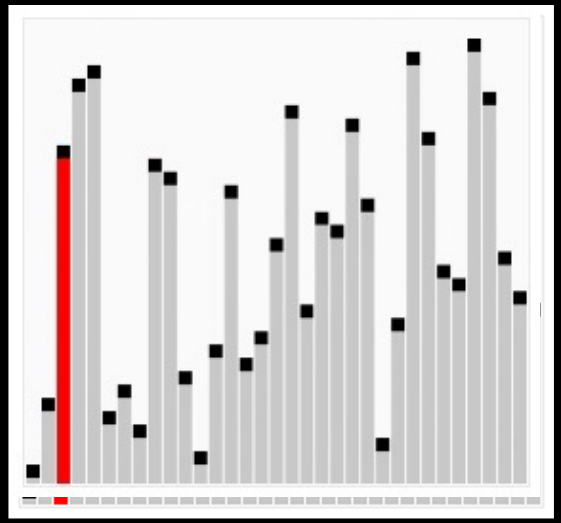
Which sorting algorithm is this?



START AT 3:06

In a file on your IDE, create a ”person” struct with attributes name and age. Then, in int main(void), create an array of 2 persons. Iterate through the array and print each name and age.

```
typedef struct
{
    string name;
    int age;
}
person
```

```
int main (void)
{
  person people[2];
  people[0].name = "Luca";
  people[0].age = 19;
}
```

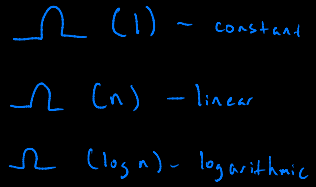
```
int scores[5];

for (int i=0; i<2; i++)
{
  print ("%s is %i years old.",
      people[i].name, people[i].age
}
people.name = ["Luca", "Grace"]
```

CS50 Section 3

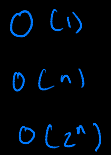
- Searching and Running Times
- Structs
- Sorting
- Recursion

Runtime

- Best Case?

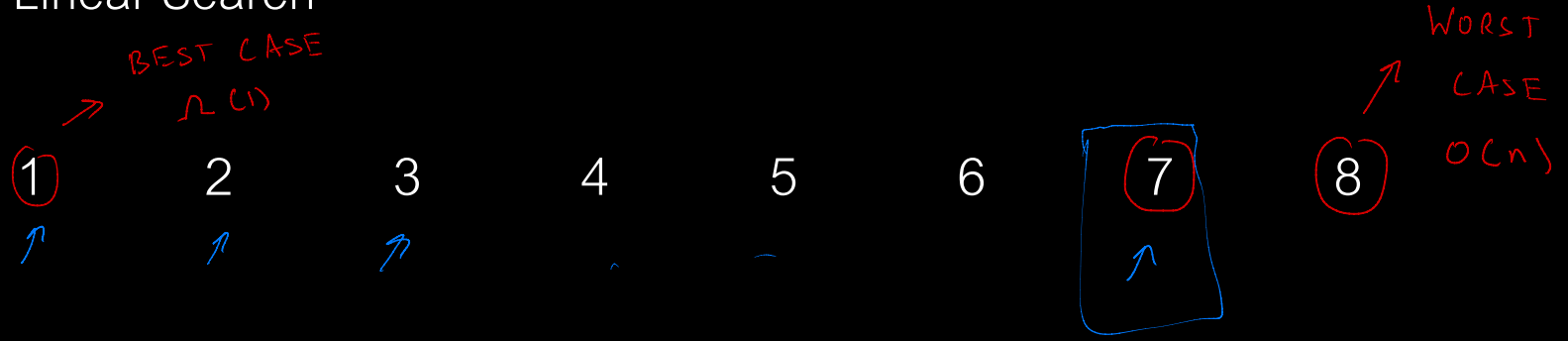


- Worst Case?



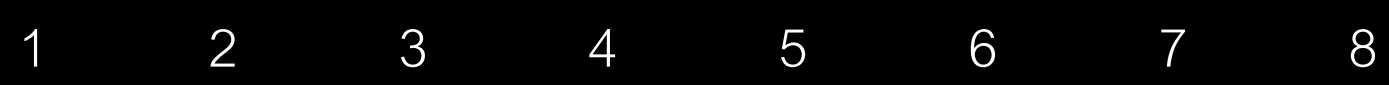
Searching

- Linear Search



Searching and Run Times

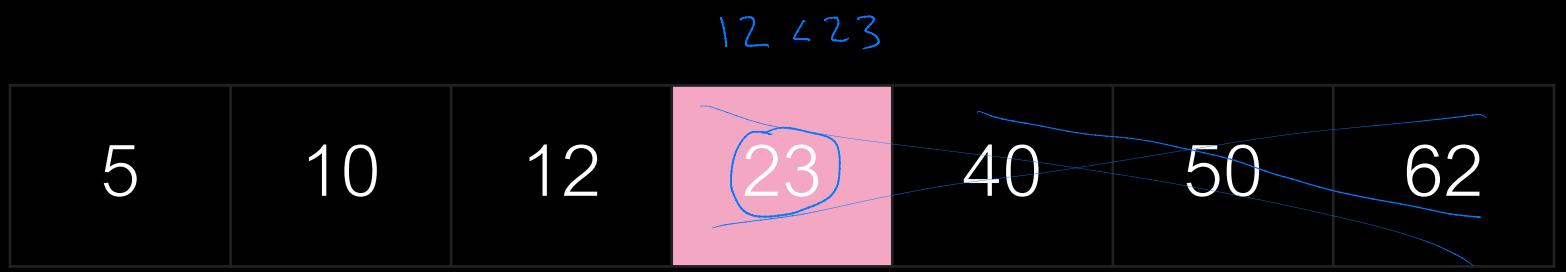
- Linear Search



- Best Case? $\Omega(1)$
- Worst Case? $O(n)$

Searching and Run Times

- Binary Search – Let's look for the number 12.



Searching and Run Times

- Binary Search – Let's look for the number 12.

Searching and Run Times

- Binary Search – Let's look for the number 12.

3 operations

$O(\log n)$

$\log_2 n$

$\log_2 7 \approx 3$

32
16
8
4
2
1

Searching and Run Times

- Binary Search – Let's look for the number 12.

- Best Case? $\Omega(1)$
- Worst Case? $O(\log n)$

- Searching and Running Times
- Structs
- Sorting
- Recursion

Structs

```
typedef struct
{
    string name;
    int age;
}
person;
```

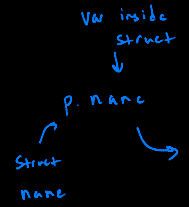
variables that your struct contains

name of the struct

```
struct person {
    string name;
    int age;
};

struct person
```

Structs

```
typedef struct
{
    string name;
    int age;
}
person;
```




```
person p;
p.name = “Phyllis”;
p.age = 19;
```

Why a struct?

group variables into a common entity

3:30

Exercise 1: Struct Practice

```
typedef struct
{
    string name;
    int age;
}
person;
```

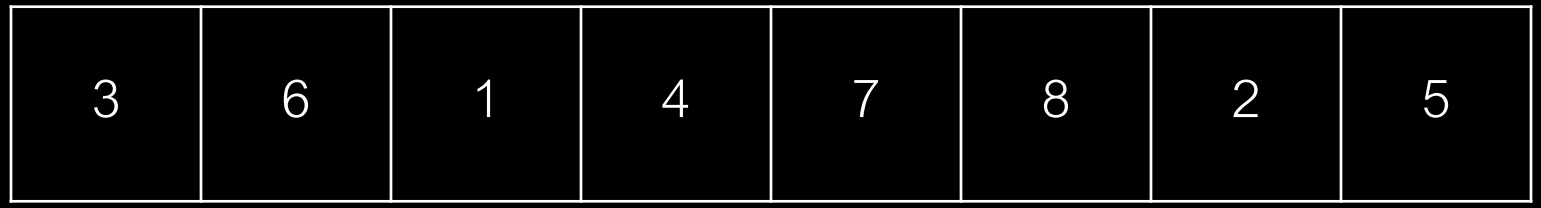
typedef struct
{
string name;
int month;
int day;
}
birthday;

Construct a struct called “birthday” that contains a variable *name* as a *string*, the variable *month* as an *int*, and the variable *day* as an *int*.

Create an array storing birthdays, initialize your contents, and print the contents of your array.

- Searching and Running Times
- Structs
- Sorting
- Recursion

Bubble Sort



3 6 1 4 7 8 2 5

3 6 1 4 7 8 2 5

3 1 6 4 7 8 2 5

3 1 4 6 7 8 2 5

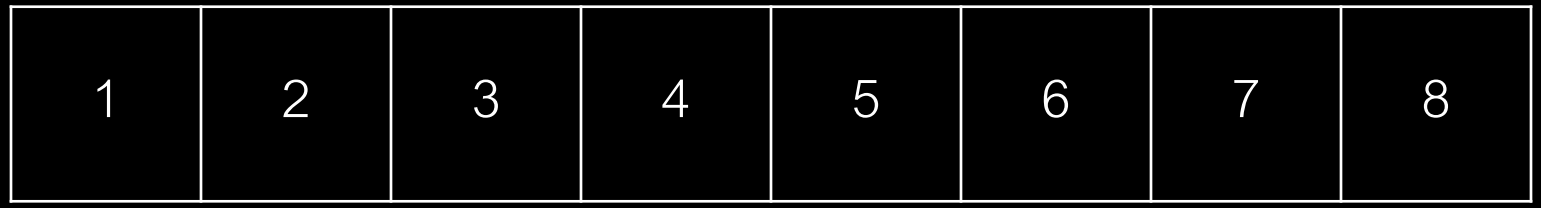
3 1 4 6 7 8 2 5

3 1 4 6 7 8 2 5

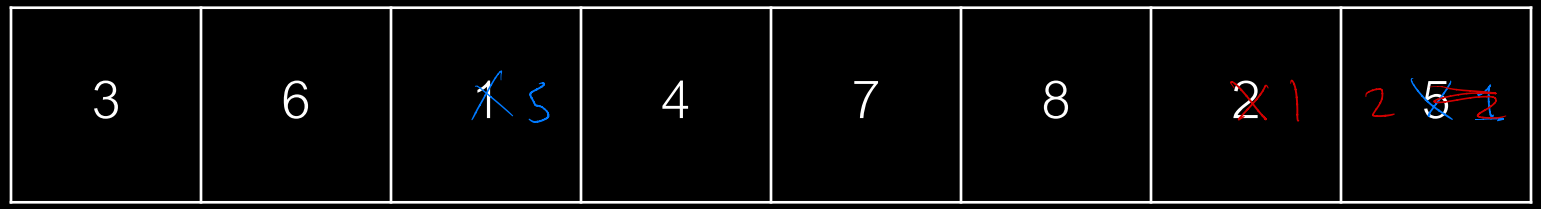
3 1 4 6 7 2 8 5

3 1 4 6 7 2 5 8

Bubble Sort



Bubble Sort

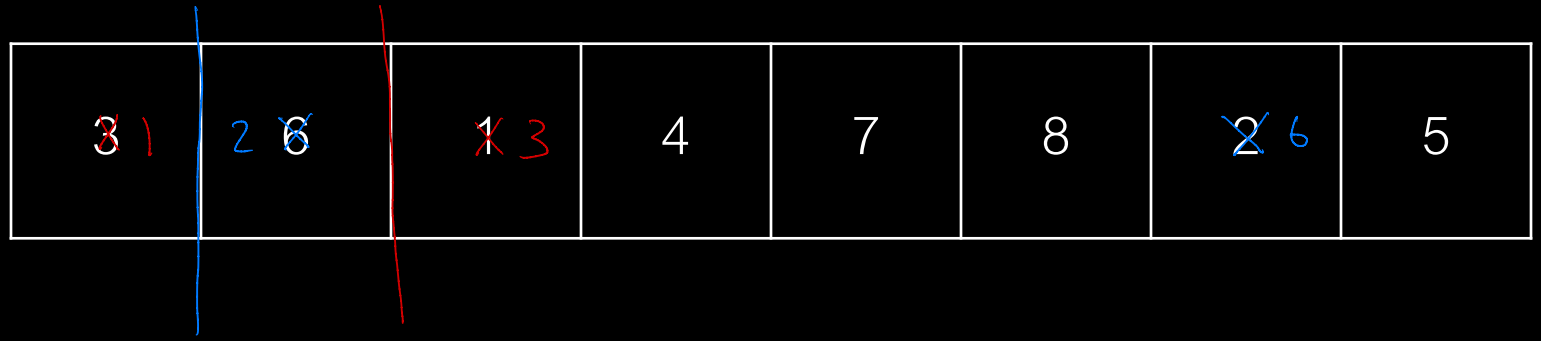


- Best Case? $\Omega(n) = \Omega(n-1) = \Omega(10^{30}n)$
- Worst Case? $O(n^2)$

n passes, n comparisons per pass $\Rightarrow$ n^2 operations

WHEN SMALLEST ELEMENT IS LAST IN ORIGINAL ARRAY

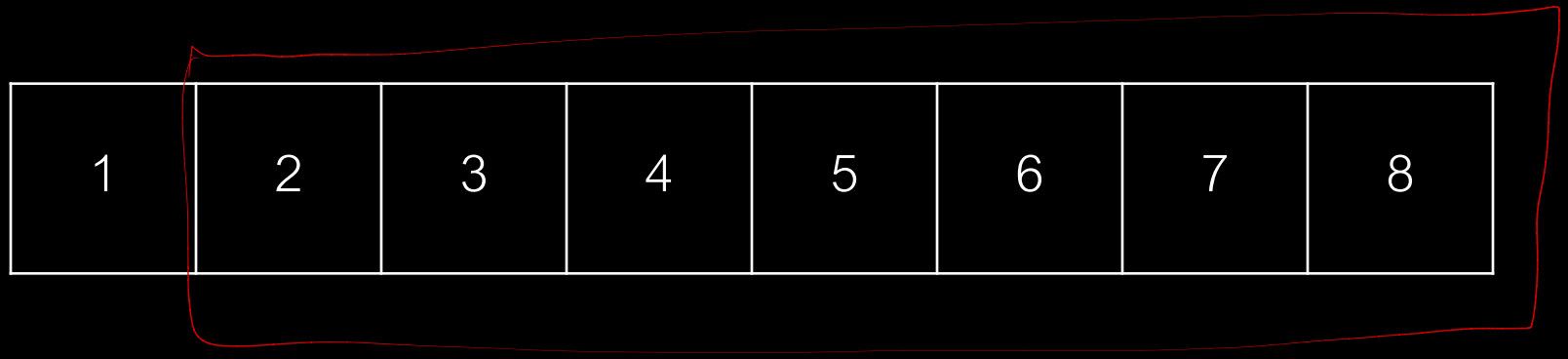
Selection Sort



min

3, 1

Selection Sort

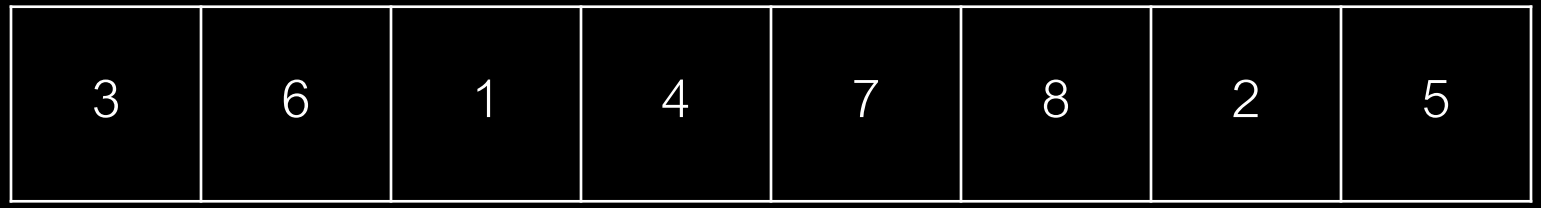


$7 + 6 + 5 + 4 + 3 + 2 + 1$

$(n-1) + (n-2) + \ldots + 1$

$\frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$

Selection Sort



- Best Case? $\Omega(n^2)$
- Worst Case? $O(n^2)$

BEST = WORST

$\Theta(n^2)$

Selection Sort

[1, 5, 2, 6, 3, 7]

```
int arr [6] = [3, 5, 2, 6, 1, 7];

for (int i = 0; i < 6; i++)
{
    min = arr[i];
    idx = i;
    for (int j = 1; j < 6; j++)
    {
        if (min > arr[j])
        {
            min = arr[j];
            idx = j;
        }
    }
    temp = arr[i];
    arr[i] = min;
    arr[idx] = temp;
}
```

temp = arr[0]
arr[0] = 1
arr[4] = 3

i = 0
min = arr[i] = 3;
idx = i;

j = 0
min = 3;
idx = 0;

j = 1
min = 3;
idx = 0;

j = 2
min = 2;
idx = 2;

j = 3
min = 2;
idx = 2;

j = 4
min = 1;
idx = 4;

j = 5
min = 1;
idx = 4

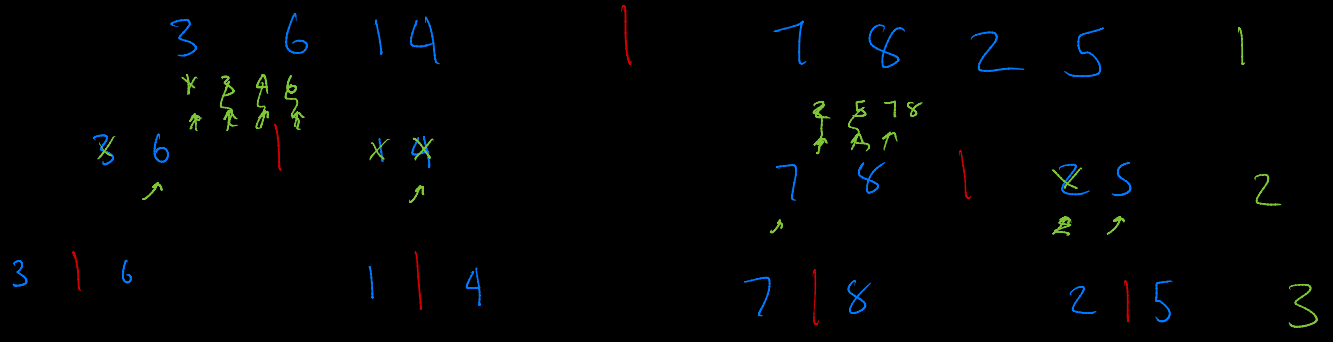
RESTART AT 4

Merge Sort

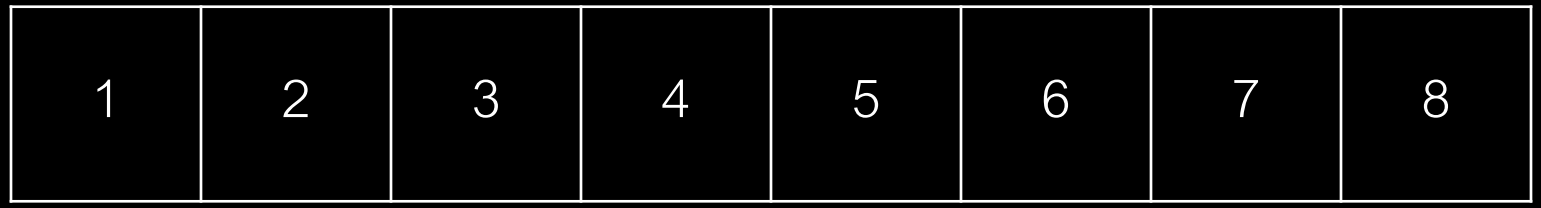
1 3 5 7 2 4 6 8 n comparisons when merging

3	6	1	4	7	8	2	5

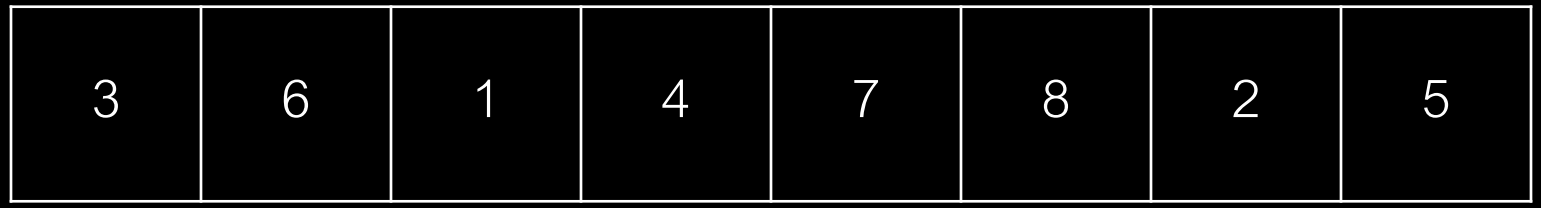
1 2 3 4 5 6 7 8



Merge Sort



Merge Sort

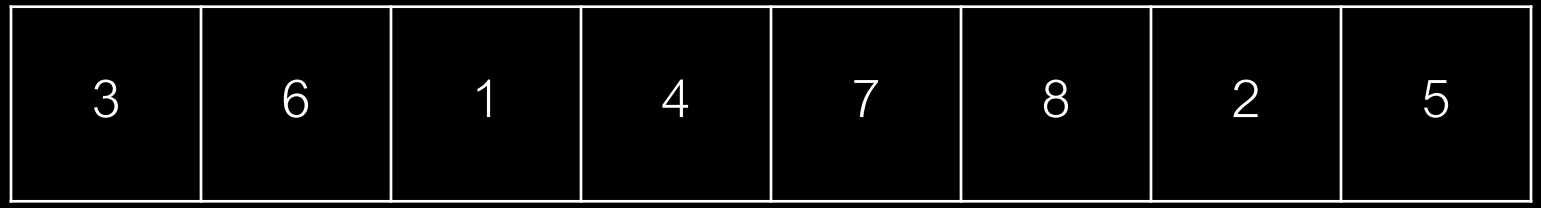


n comparisons when merging
log n merges
] => n log n runtime

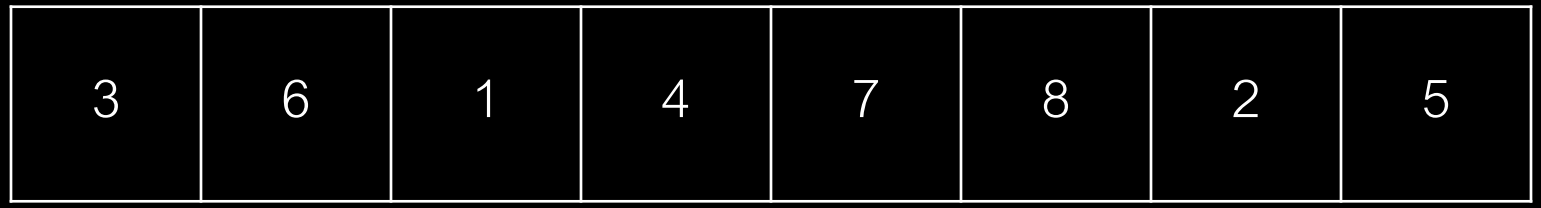
- Best Case? $\Omega(n \log n)$
- Worst Case? $O(n \log n)$

] => $\Theta(n \log n)$

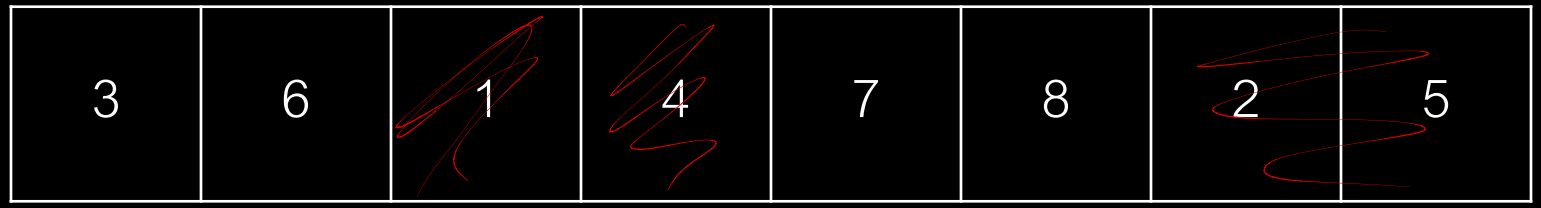
Fun Sorts :D



Stalin Sort (fake sort vibes)



Stalin Sort (fake sort vibes)

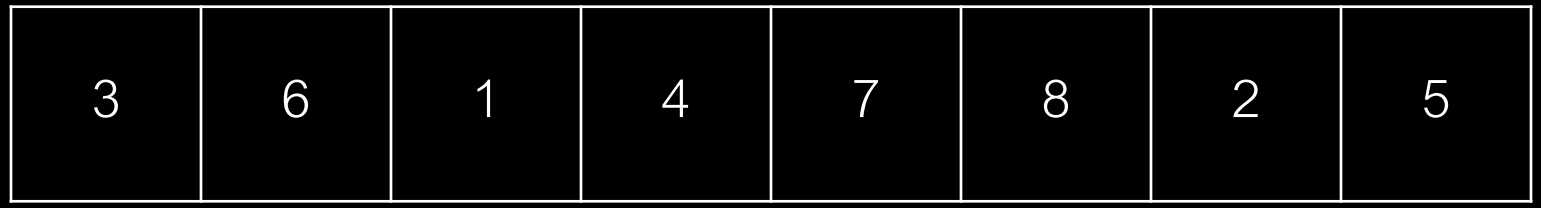


3 6 7 8

- Best Case? $\Omega(n)$
  $\Rightarrow \Theta(n)$
- Worst Case? $O(n)$

Bogosort Sort

8



8! possible permutations

Worst Case: $n \cdot n!$ $O((n+1)!)$

Best Case: $\Omega(n)$

Summary

Selection Sort

BEST : $\Omega(n^2)$

WORST : $O(n^2)$

Bubble Sort

BEST : $\Omega(n)$ → Already sorted

WORST : $O(n^2)$

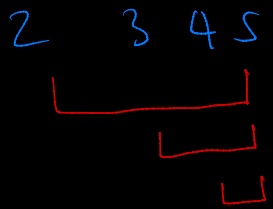
Merge Sort

BEST : $\Omega(n \log n)$

WORST : $O(n \log n)$

Exercise: Reading and Understanding Code!

```
int values[n];
for (int i = 0; i < n - 1; i++)
{
   int min_index = i;
   for (int j = i + 1; j < n; j++)
   {
      if (values[j] < values[min_index])
      {
         min_index = j;
      }
   }
   int temp = values[i];
   values[i] = values[min_index];
   values[min_index] = temp;
}
```



- Searching and Running Times
- Structs
- Sorting
- Recursion

Recursion

- Base Case
- Recursive Call

↑ calling yourself / the same function

from 1 to n

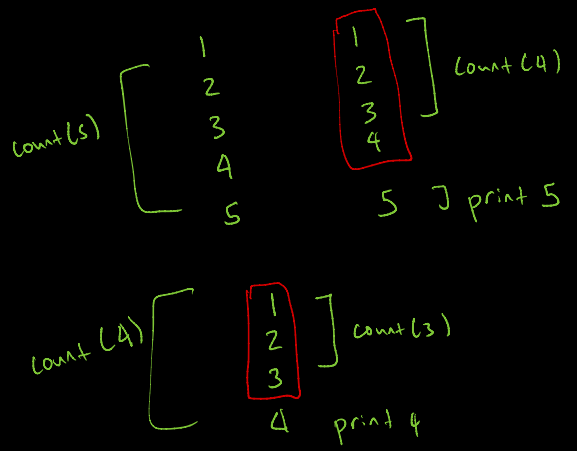
Write a function that prints integers s~~tarting from n~~ line by line, s~~ubtracting~~ adding one for each line until ~~zero~~ n is reached using recursion.

1
2
3
4
5
6
.
.
n

Recursion

```
void count(n)
{
    if (n == 0)
    {
        return;
    }
    count_up_to(n - 1);
    printf("%i\n", n);
}
```

Function



0! = 1
1! = 1

3! = 6

Recursion

- Base Case
- Recursive Call

Write a function that determines the factorial of an integer n using recursion. Prompt the user for an integer and print out its factorial.

$n! = n(n-1)\cdots 1$

$= n(n-1)!$

```
int factorial (int x)
{
  if (x <= 1) {
    return 1;
  }
  return x * factorial (x-1);
}
```

3

factorial (3)

return 3 * factorial (2)

3 * 2 = 6

factorial (2)

return 2 * factorial (1)

factorial (1) = 1

```
int n = get_int ("give int");
result = factorial (n)
printf("%i", result);
```

factorial (x) = x * factorial (x-1)

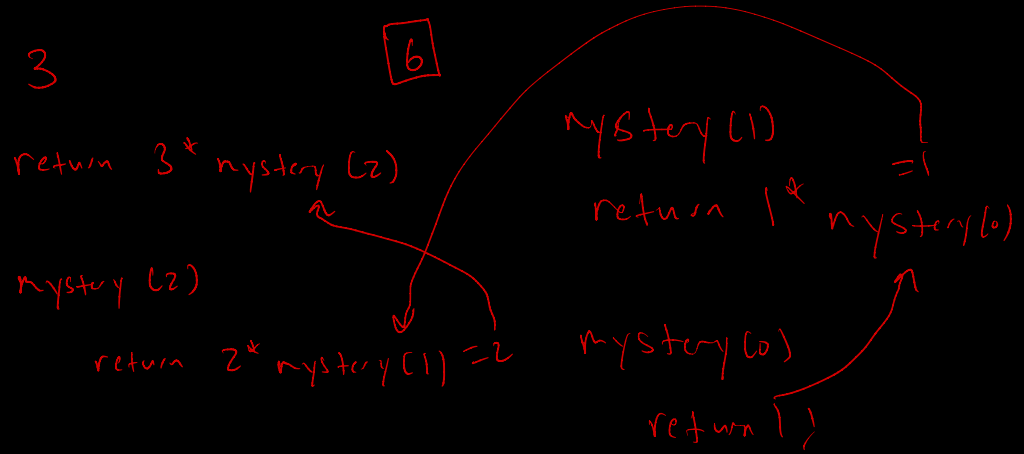
Exercise: Reading and Understanding Code!

```
int mystery(int n);

int main(void)
{
   printf("%i\n", mystery(3));
   printf("%i\n", mystery(5));
}

int mystery(int n)
{
   if (n == 0)
      return 1;

   return n * mystery(n - 1);
}
```

3!

5!

factorial



Exercise 2: Fibonacci Numbers

0 1 1 2 3 5 8 13

0 1 2 3 4

Write a recursive function *fib* that computes the nth Fibonacci number. The 0th Fibonacci number is 0, the 1st Fibonacci number is 1, and every subsequent Fibonacci number is the sum of the two preceding Fibonacci numbers.

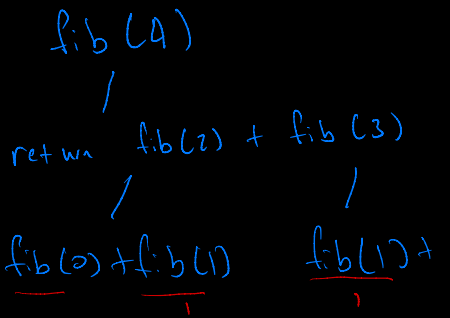
fib(7) returns 13

```
int fib(int n)
{
  if (n == 0) {
    return 0;
  }
  if (n == 1) {
    return 1;
  }
  return fib(n-1) + fib(n-2);
}
```

BASE CASE

use other calls to compute current call

fib(n) = fib(n-1) + fib(n-2)



3

3

Lab

SORT 1

SORT 2

SORT 3

BUBBLE

SELECTION

MERGE

reversed 10000.txt → reversed list of ints

random 10000.txt → random list of ints

sorted _.txt → sorted list of ints

time ./sort 1 reversed10000.txt

(sort # → 1; file → reversed10000.txt)

Run 3 sorts on sorted list

Bubble should do the best

Sorted	Reversed	Random
Bubble	Merge	Merge
Merge	Bubble	Bubble
Selection	Selection	Selection

SORT 1: BUBBLE

2: MERGE

3: SELECTION

CS50 Section 3