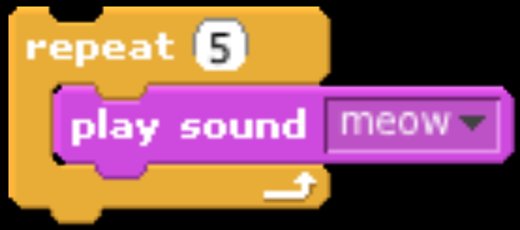
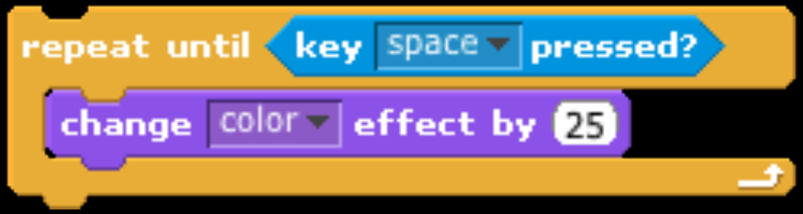
Quiz Topics

- Binary. ASCII. Algorithms. Pseudocode. Source code. Compiler. Object Code. Scratch. Statements. Boolean expressions. Loops. Variables. Functions. Arrays. Threads. Events.
- Linux. C. Compiling. Libraries. Types. Standard output.
- Casting. Imprecision. Switches. Scope. Strings. Arrays. Cryptography.
- Command-line arguments. Searching. Sorting. Bubble, Selection, Insertion sort. O. Ω. θ. Recursion. Merge Sort
- Stack. Debugging. File I/O. Hexadecimal. Strings. Pointers. Dynamic memory allocation
- Heap. Buffer overflow. Linked lists

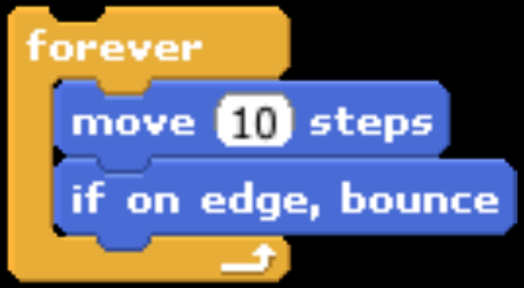
Loops



```
repeat (5)
    play sound [meow ▾]
```



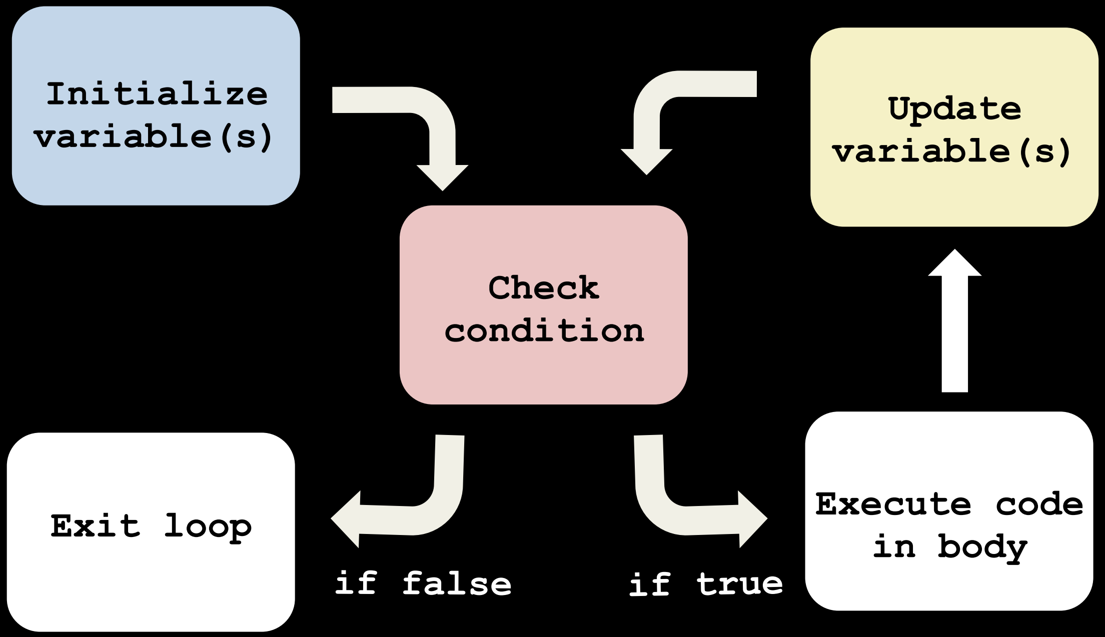
```
repeat until <key [space ▾] pressed?>
    change [color ▾] effect by (25)
```



```
forever
    move (10) steps
    if on edge, bounce
```

For Loops

```
for (initialization; condition; update)
{
    execute this code
}
```



Initialize variable(s)

Check condition

Update variable(s)

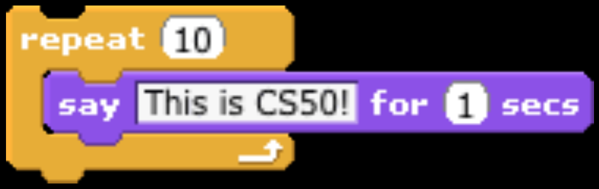
Exit loop

if false

if true

Execute code in body

Example #1
Prints “This is CS50!” ten times



```
repeat 10
    say This is CS50! for 1 secs
```

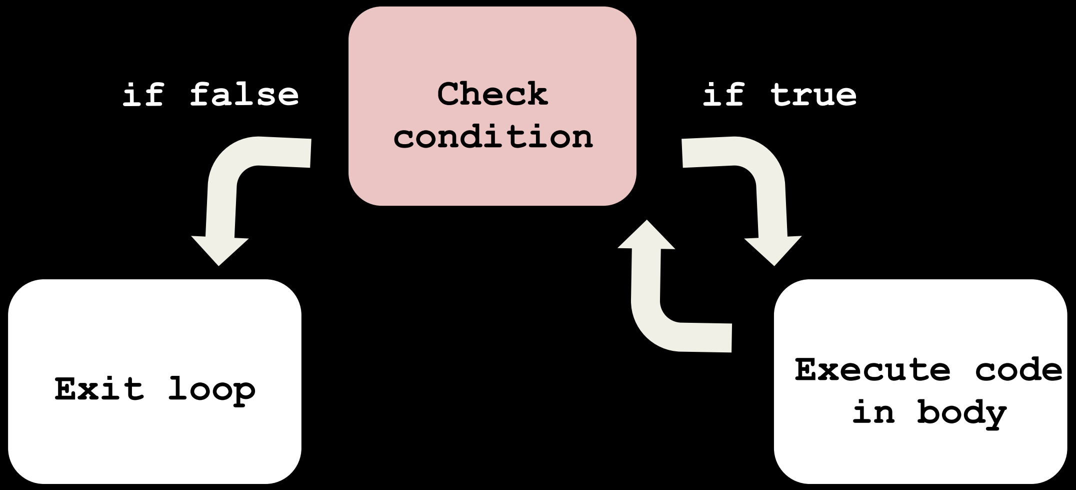
```
for (int i = 0; i < 10; i++)
{
    printf("This is CS50!\n");
}
```

Example #2
Converts a lowercase string to uppercase

```
char name[] = "milo";
for (int i = 0, j = strlen(name); i < j; i++)
{
    name[i] = toupper(name[i]);
}
```

While Loops

```
while (condition)
{
    execute this code
}
```



Check condition

if false

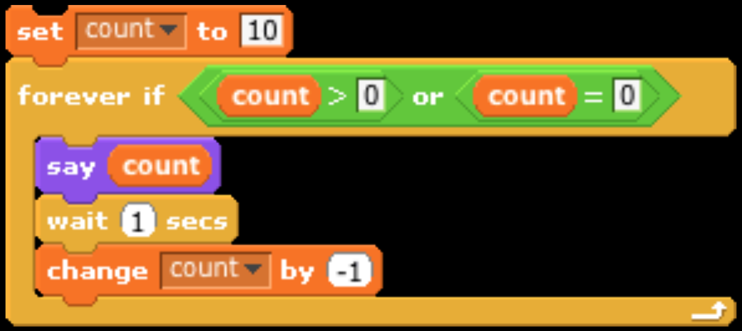
Exit loop

if true

Execute code in body

Example #3

Counts down from 10 to 0



```
set count to 10
forever if count > 0 or count = 0
    say count
    wait 1 secs
    change count by -1
```

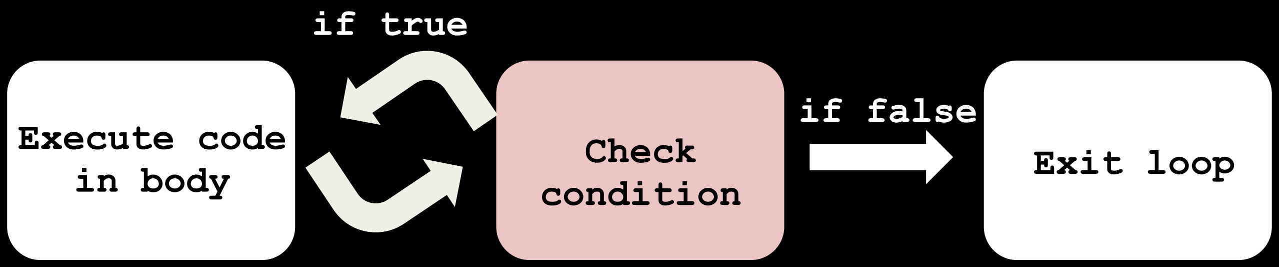
```c
int count = 10;
while (count >= 0)
{
    printf("%i\n", count);
    count--;
}
```

Example #4
Calculates string length

```
string s = GetString();
int length = 0;
while (s[length] != '\0')
    length++;
```

Do While Loops

```
do
{
      execute this code
}
while (condition);
```



Execute code in body

if true

Check condition

if false

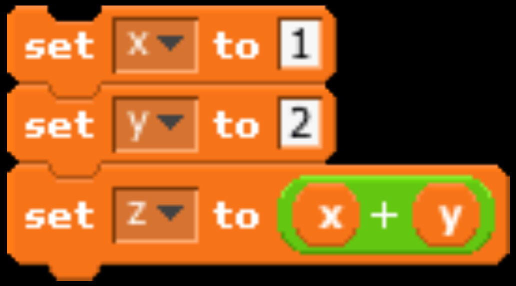
Exit loop

Example #5

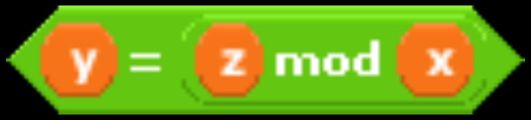
Reprompts until user enters a positive number

```c
int input;
do
{
    printf("Enter a positive number: ");
    input = GetInt();
}
while (input < 1);
```

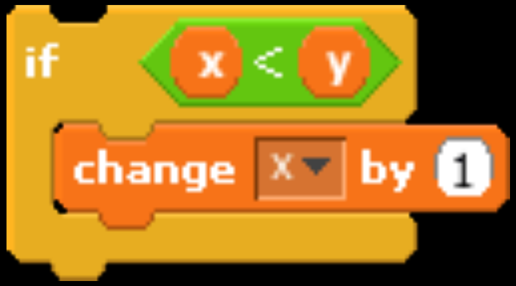
Math in C



```
set x to 1
set y to 2
set z to (x + y)
```



```
<(y) = ((z) mod (x))>
```



```
if <(x) < (y)>
    change x by 1
```

Numerical Variables

```
int
float
double
long long
```

Let’s add some `ints`!

```
// declare x
int x;

// initialize x
x = 2;

// declare and initialize y
int y = x + 1;
```

Division

```
int main(void)
{
    // declare and initialize answer
    float answer = 1 / 10;

    // print answer to two decimal places
    printf("%.2f\n", answer);
}
```

Fixed version: Typecasting

```
int main(void)
{
    // declare and initialize answer
    float answer = (float) 1 / (float) 10;

    // print answer to two decimal places
    printf("%.2f\n", answer);
}
```

Another way

```
int main(void)
{
    // declare and initialize answer
    float answer = 1.0 / 10.0;

    // print answer to two decimal places
    printf("%.2f\n", answer);
}
```

Operator Precedence

What is x?

```
1. int x = 2 * 10 + 10 / 2 + 2;
2. int x = 2 * (10 + 10) / 2 + 2;
3. int x =  2 * (10 + 10) / (2 + 2);
```

Modulo

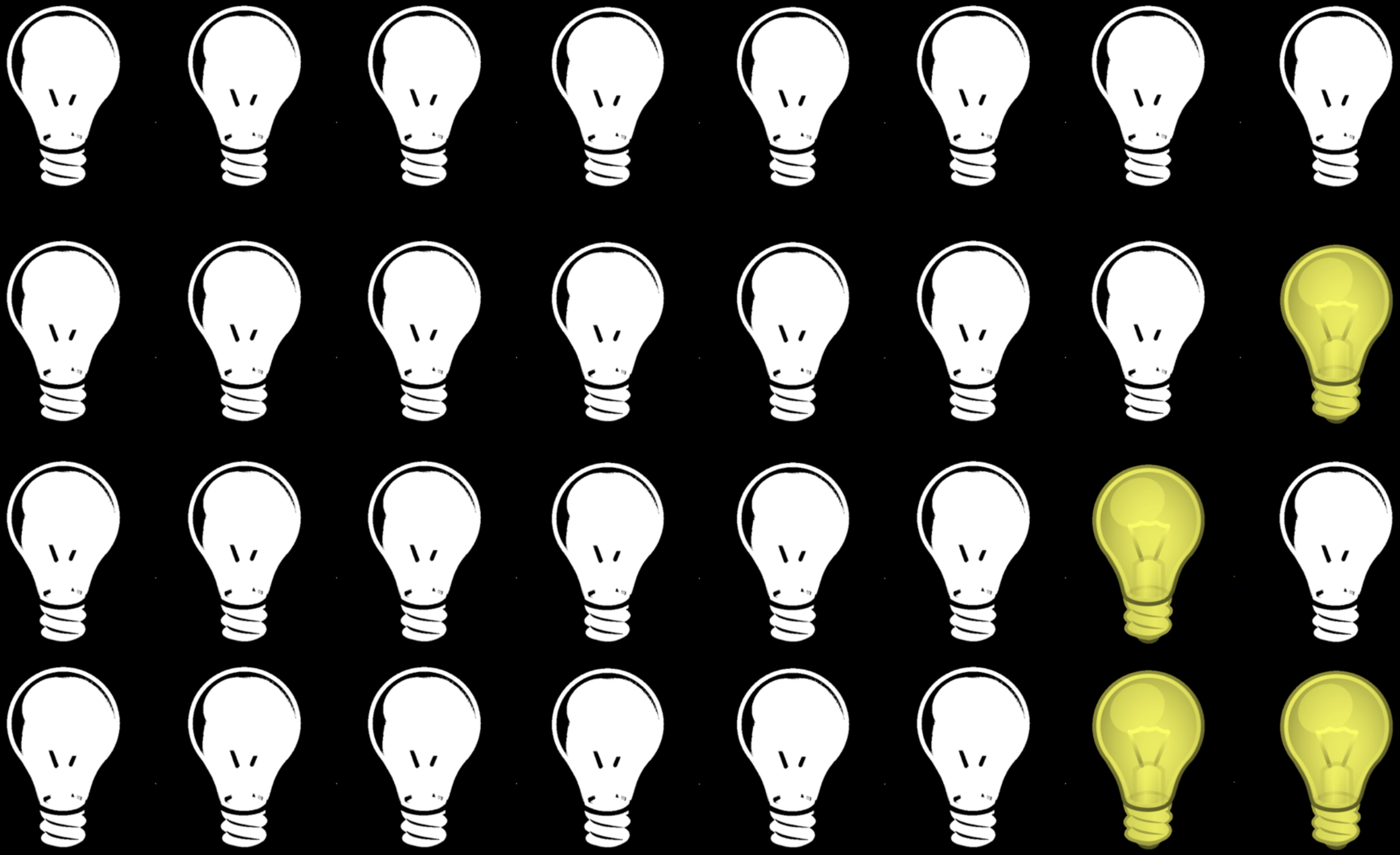
1. 55 % 10
2. 3 % 5
3. 8 % 8
4. 16 % 15

What will print?

```
int main(void)
{
    // declare and initialize x, y, z
    int x = 1;
    int y = 2;
    int z = (x + y) * y % y + y;

    // print z
    printf("%i\n", z);
}
```

Floating Point Imprecision

```
int main(void)
{
    // initialize x and y
    float answer = 1.0 / 10.0;

    // print answer to two decimal places
    printf("%.20f\n", answer);
}
```



We are used to decimal notation:

$$\underset{10^2}{\underline{1}}\quad\underset{10^1}{\underline{6}}\quad\underset{10^0}{\underline{3}}$$

$$1*10^2 + 6*10^1 + 3*10^0 = 163$$

Computers store and process data via binary notation:

$$\begin{array}{cccccccc} \underline{1} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{1} \\ 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

$$1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 163$$

Converting Binary to Decimal (and vice versa)

$1 = 1*2^0 = 1$

$10 = 1*2^1 + 0*2^0 = 2$

$11 = 1*2^1 + 1*2^0 = 3$

$100 = 1*2^2 + 0*2^1 + 0*2^0 = 4$

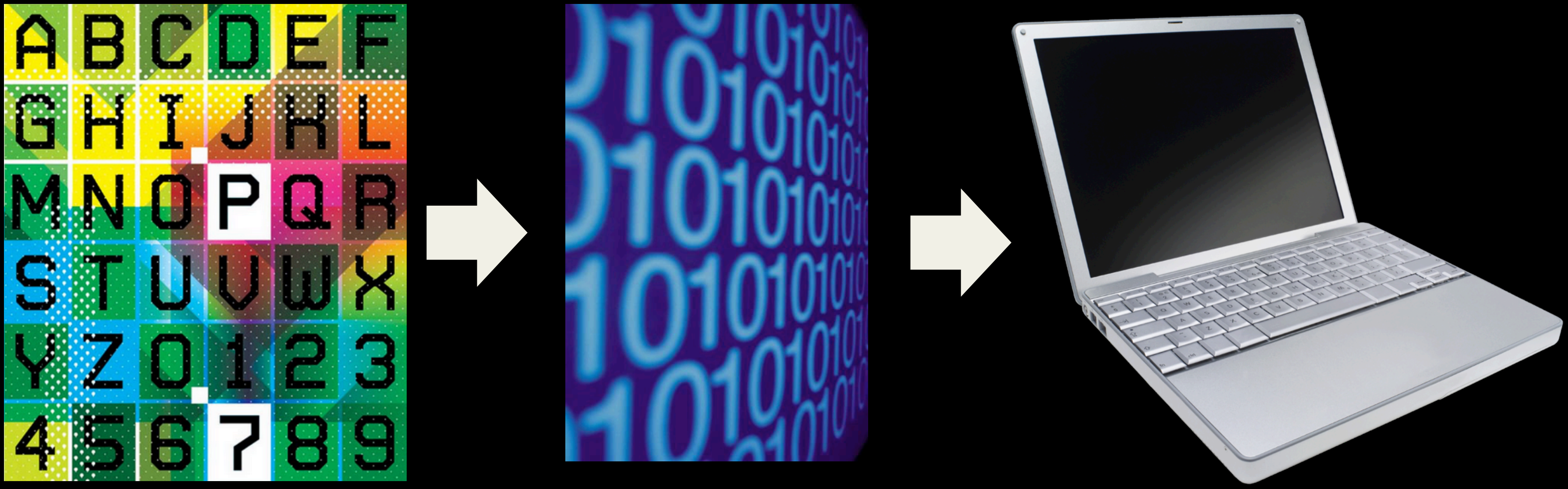
$101 = 1*2^2 + 0*2^1 + 1*2^0 = 5$

Addition and Subtraction
(Don’t forget to carry your 1s)

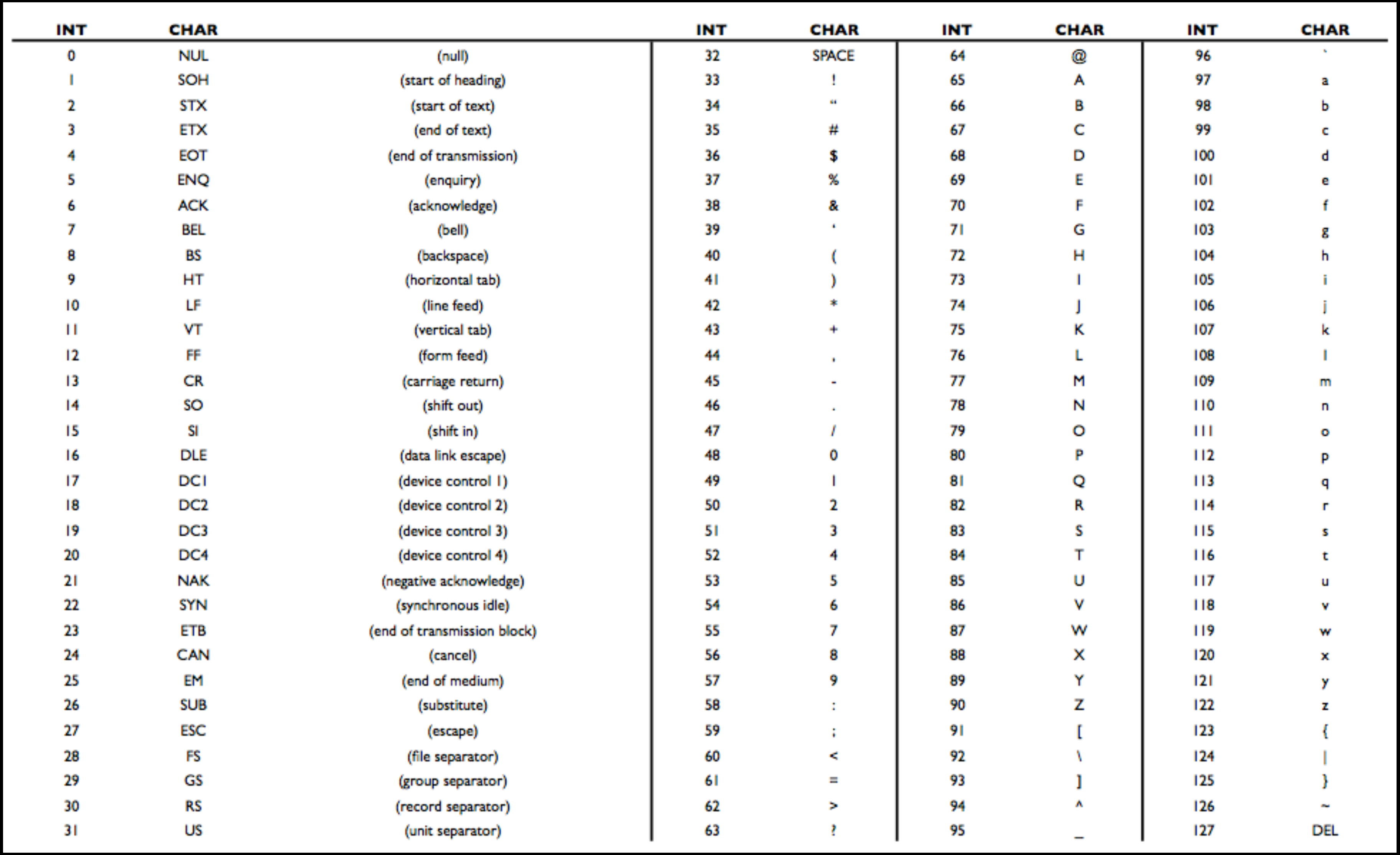
$$\begin{array}{r} 1\;0\;1\;0^{1}\;1^{1}\;1 \\ +\;\;0\;1\;0\;0\;0\;1 \\ \hline 1\;1\;1\;1\;0\;0 \end{array}$$

$$\begin{array}{r} 1\;1\;\cancel{1}^{0}\;\cancel{0}^{10}\;0 \\ -\;\;0\;0\;0\;1\;0 \\ \hline 1\;1\;0\;1\;0 \end{array}$$

Characters must also be encoded in binary



ASCII maps characters to numbers



INT	CHAR		INT	CHAR	INT	CHAR	INT	CHAR
0	NUL	(null)	32	SPACE	64	@	96	`
1	SOH	(start of heading)	33	!	65	A	97	a
2	STX	(start of text)	34	“	66	B	98	b
3	ETX	(end of text)	35	#	67	C	99	c
4	EOT	(end of transmission)	36	$	68	D	100	d
5	ENQ	(enquiry)	37	%	69	E	101	e
6	ACK	(acknowledge)	38	&	70	F	102	f
7	BEL	(bell)	39	'	71	G	103	g
8	BS	(backspace)	40	(	72	H	104	h
9	HT	(horizontal tab)	41	)	73	I	105	i
10	LF	(line feed)	42	*	74	J	106	j
11	VT	(vertical tab)	43	+	75	K	107	k
12	FF	(form feed)	44	,	76	L	108	l
13	CR	(carriage return)	45	-	77	M	109	m
14	SO	(shift out)	46	.	78	N	110	n
15	SI	(shift in)	47	/	79	O	111	o
16	DLE	(data link escape)	48	0	80	P	112	p
17	DC1	(device control 1)	49	1	81	Q	113	q
18	DC2	(device control 2)	50	2	82	R	114	r
19	DC3	(device control 3)	51	3	83	S	115	s
20	DC4	(device control 4)	52	4	84	T	116	t
21	NAK	(negative acknowledge)	53	5	85	U	117	u
22	SYN	(synchronous idle)	54	6	86	V	118	v
23	ETB	(end of transmission block)	55	7	87	W	119	w
24	CAN	(cancel)	56	8	88	X	120	x
25	EM	(end of medium)	57	9	89	Y	121	y
26	SUB	(substitute)	58	:	90	Z	122	z
27	ESC	(escape)	59	;	91	[	123	{
28	FS	(file separator)	60	<	92	\	124	|
29	GS	(group separator)	61	=	93	]	125	}
30	RS	(record separator)	62	>	94	^	126	~
31	US	(unit separator)	63	?	95	_	127	DEL

ASCII Math

What will print?

```
printf("%d\n", 'a' - 'A');
printf("%c\n", 'B' + ('a' - 'A'));
printf("%c\n", 'b' - ('a' - 'A'));
printf("%c\n", 'B' + 1);
printf("%c\n", ('z' - 'a' + 1) % 26 + 'a');
```

Example #1
Prints Z through A

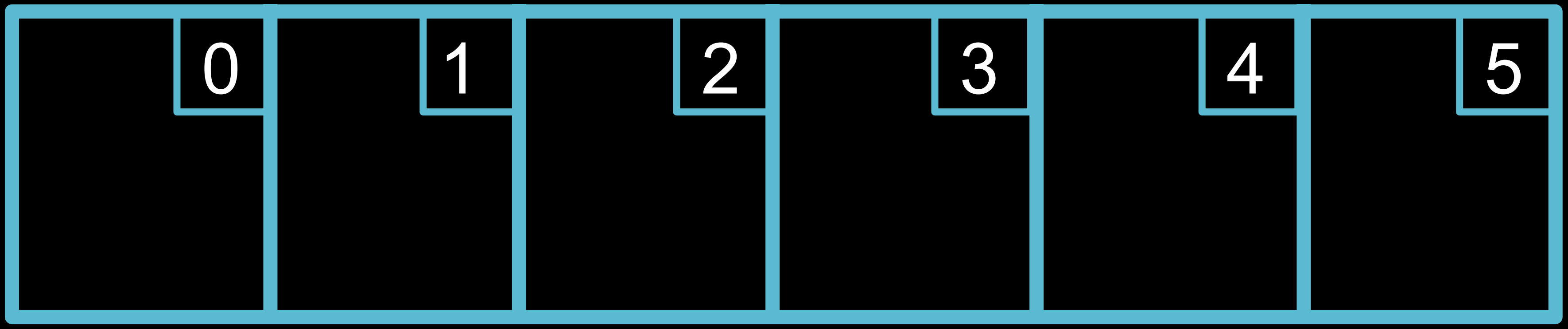
```
for (int i = 'Z'; i >= 'A'; i--)
    printf("%c\n", i);
```

Example #2

Converts a lowercase string to uppercase

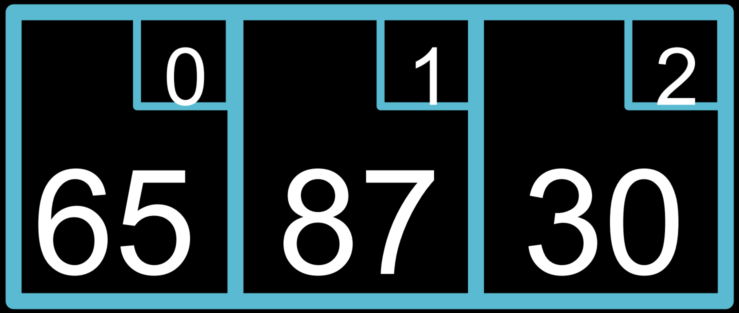
```
char name[] = "milo";
for (int i = 0, j = strlen(name); i < j; i++)
    name[i] = name[i] + ('A' - 'a');
```

Arrays



Creating an Array

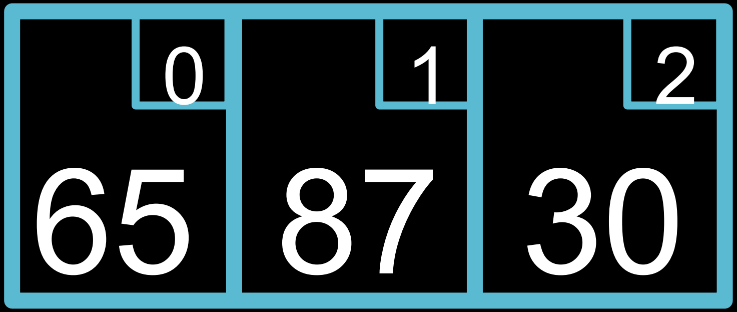
```
<data type> name[<size>];

Example:
int temperature[3];
temperature[0] = 65;
temperature[1] = 87;
temperature[2] = 30;

OR

int temperature[] = { 65, 87, 30 };
```



0 1 2

65 87 30

Accessing Array Elements



0	1	2
65	87	30

```
for (int i = 0; i < 3; i++)
{
      printf("%d\n", temperature[i]);
}
```

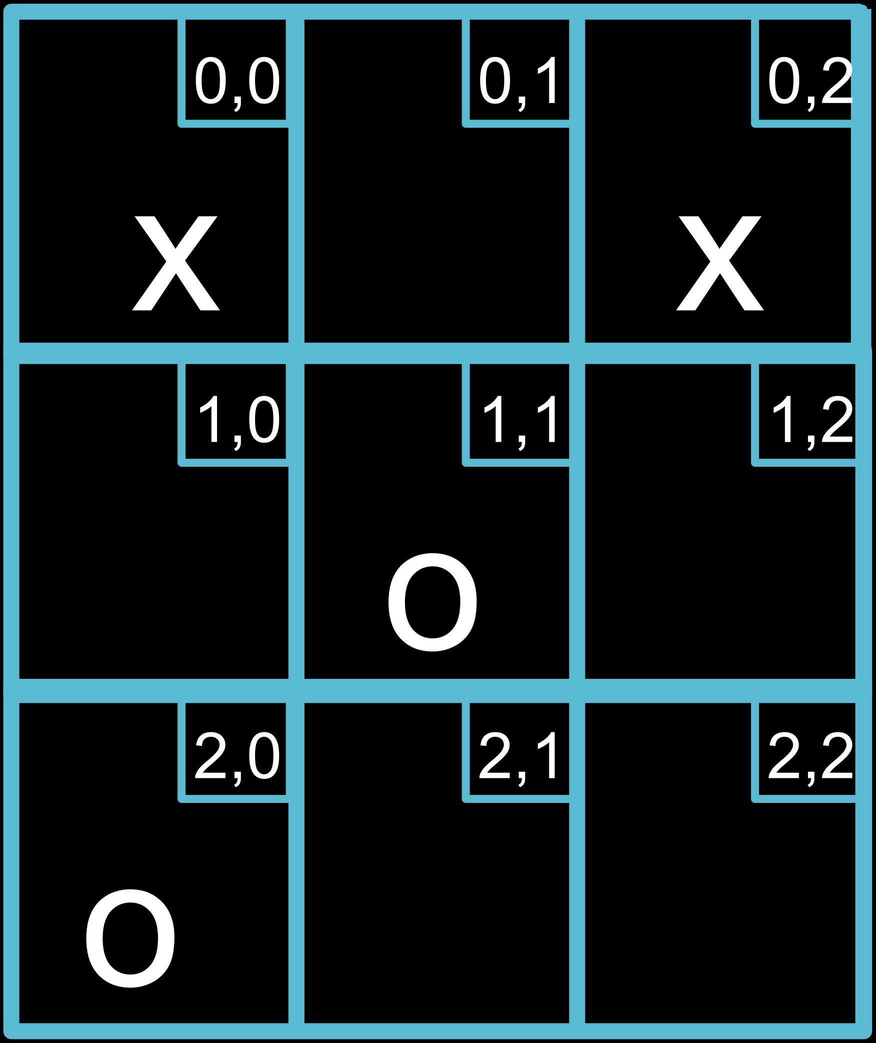
```
#include <stdio.h>
#include <cs50.h>

#define CLASS_SIZE 30

int main(void)
{
     // declare array
     int scores_array[CLASS_SIZE];

     // populate array
     for (int i = 0; i < CLASS_SIZE; i++)
     {
     printf("Enter score for student %d: ", i);
     scores_array[i] = GetInt();
   }
}
```

Where's the bug?

```
string class[3] = { "Sam", "Jess", "Kim" };

for (int i = 0; i <= 3; i++)
{
     printf("%s\n", class[i]);
}
```

Multidimensional Arrays

```
char board[3][3];
board[1][1] = 'o';
board[0][0] = 'x';
board[2][0] = 'o';
board[0][2] = 'x';
```

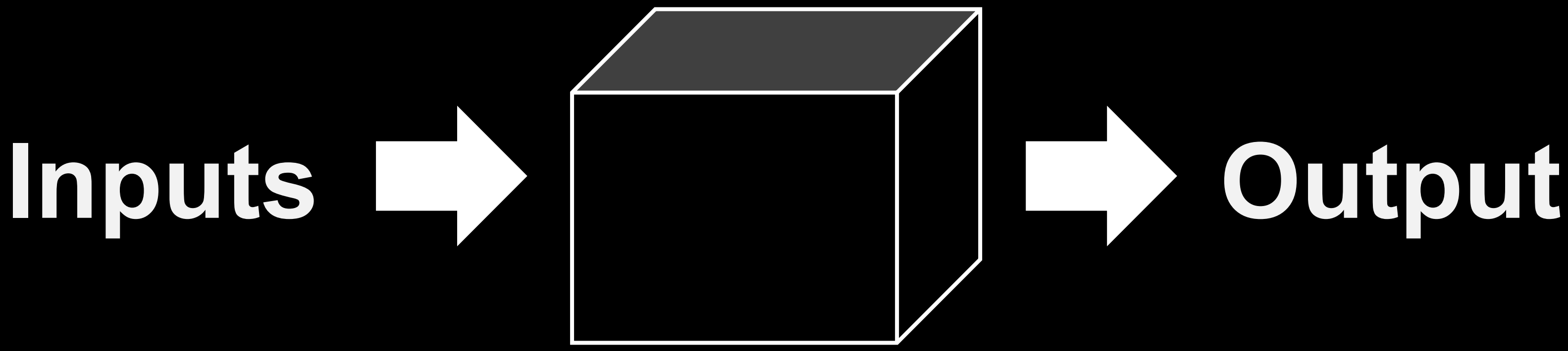


0,0 X	0,1	0,2 X
1,0	1,1 O	1,2
2,0 O	2,1	2,2

Accessing Multidimensional Array Elements

```
// print out all elements
for (int i = 0; i < 3; i++)
{
   for (int j = 0; j < 3; j++)
           printf("%c", board[i][j]);
     printf("\n");
}
```

Functions



Why Functions?

- Organization
- Simplification
- Reusability

A Function Definition

```
int cube(int input)
{
    int output = input * input * input;
    return output;
}
```

Header

function name

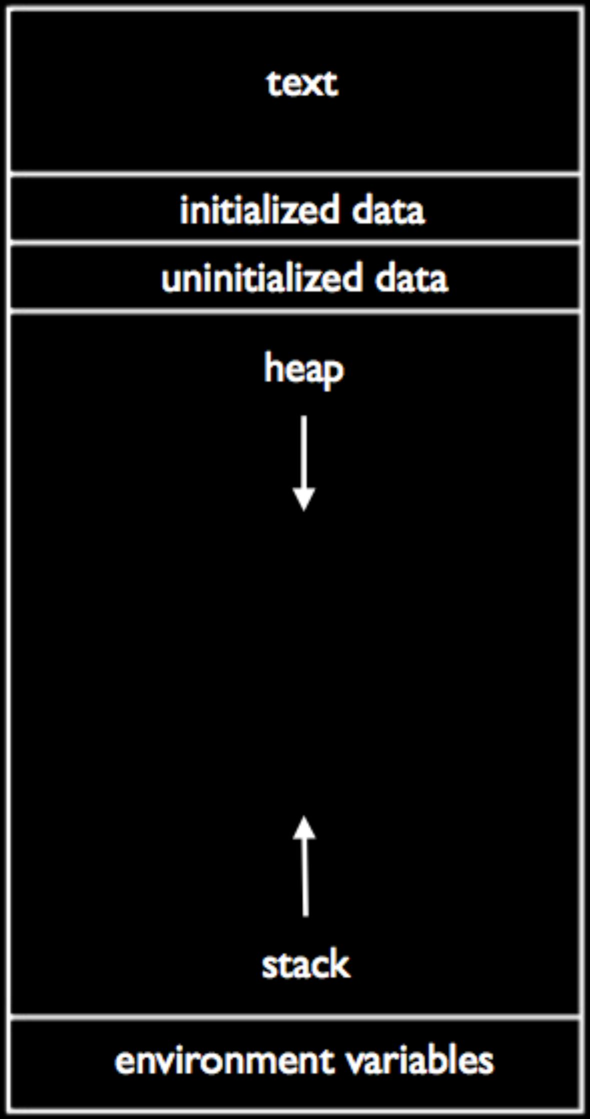
```
int cube(int input)
{
    int output = input * input * input;
    return output;

}
```

return type

parameter list

Body

```c
#include <stdio.h>

int cube(int input);

int main(void)
{
    int x = 2;
    printf("x is %i\n", x);
    x = cube(x);
    printf("x is %i\n", x);
}

int cube(int input)
{
    int output = input * input * input;
    return output;
}
```



text

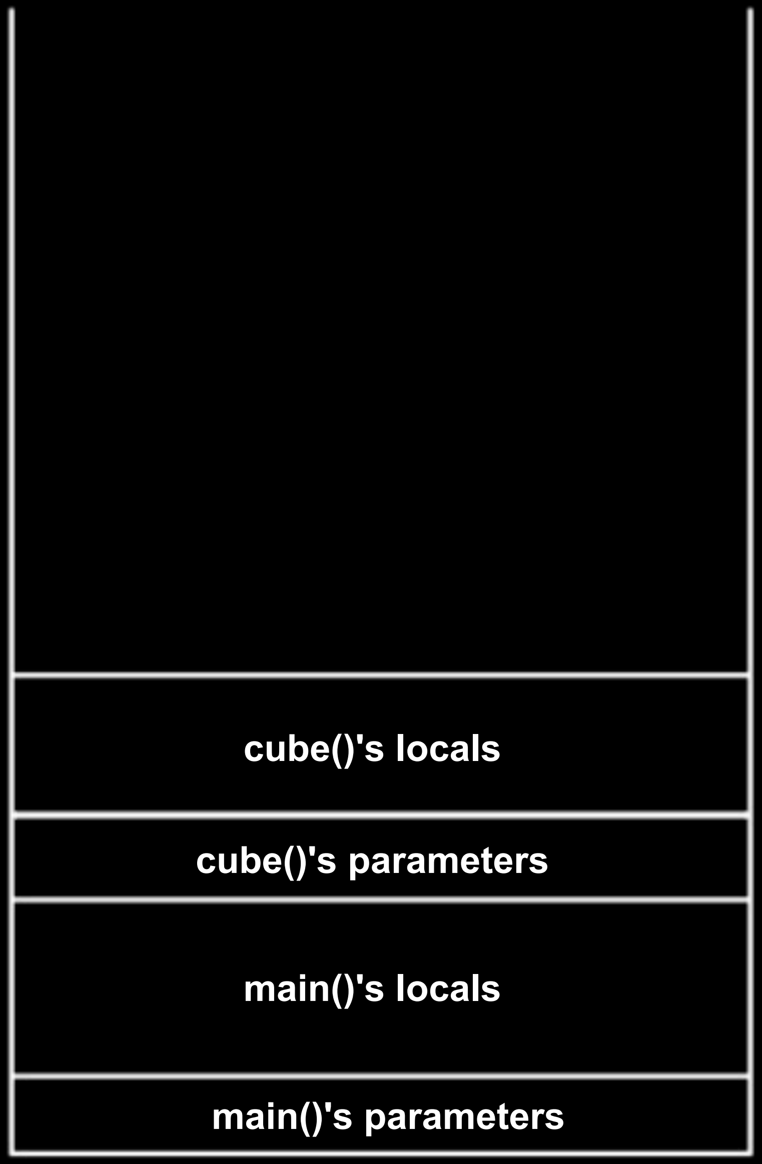
initialized data

uninitialized data

heap

stack

environment variables



cube()'s locals

cube()'s parameters

main()'s locals

main()'s parameters

```c
#include <stdio.h>
void swap(int a, int b);

int main(void)
{
    int x = 1;
    int y = 2;
    swap(x, y);
    printf("x is %i\n", x);
    printf("y is %i\n", y);
}

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

Command-line Arguments

```
int main(void)

int main(int argc, string argv[])
```

Test Yourself

```
jharvard@appliance (~): ./copy infile outfile
```

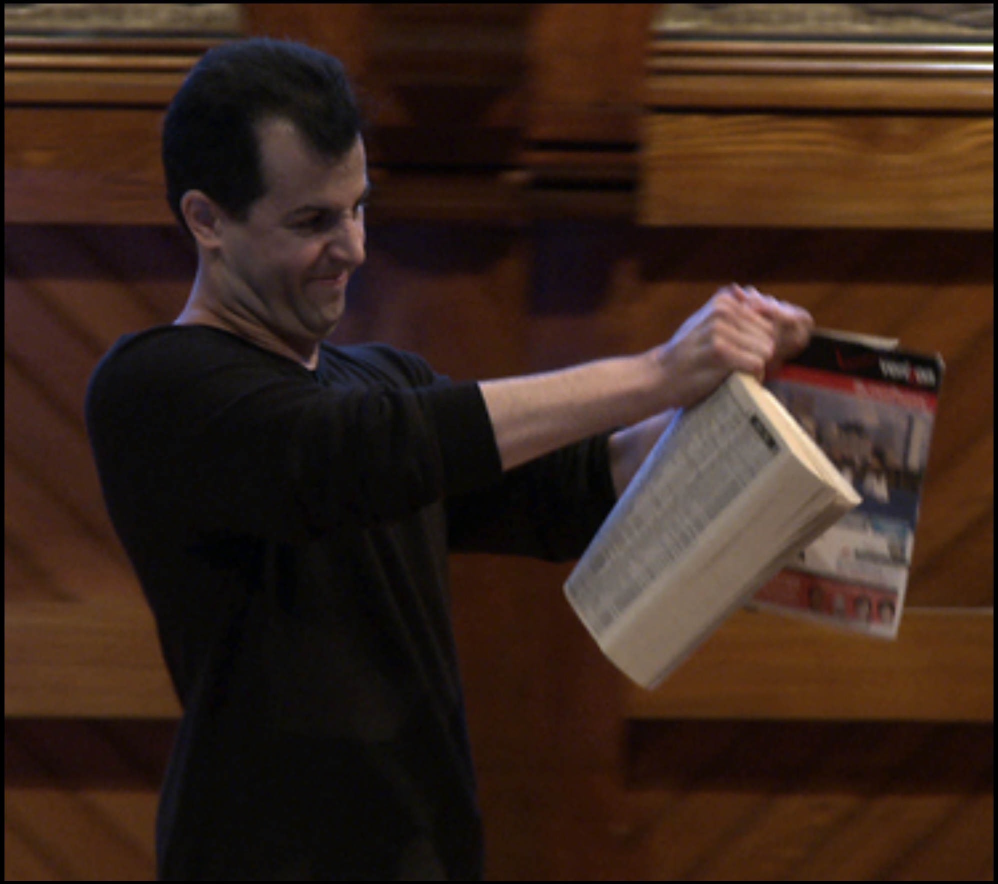
1. What is argc?
2. What is argv[0]?
3. What is argv[1]?
4. What is argv[2]?
5. What is argv[3]?
6. What is argv[4]?

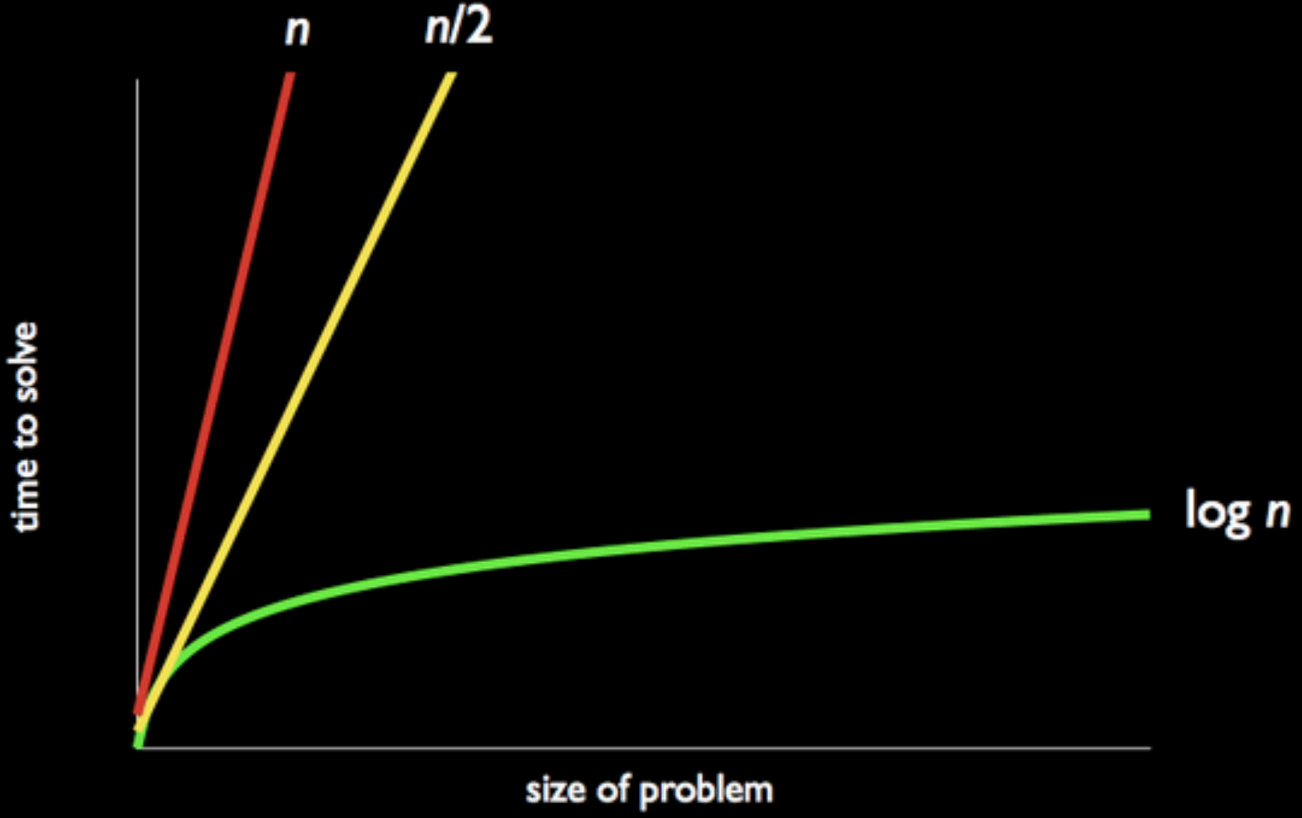
Mario Revisited

```
jharvard@appliance (~): ./mario 10
```

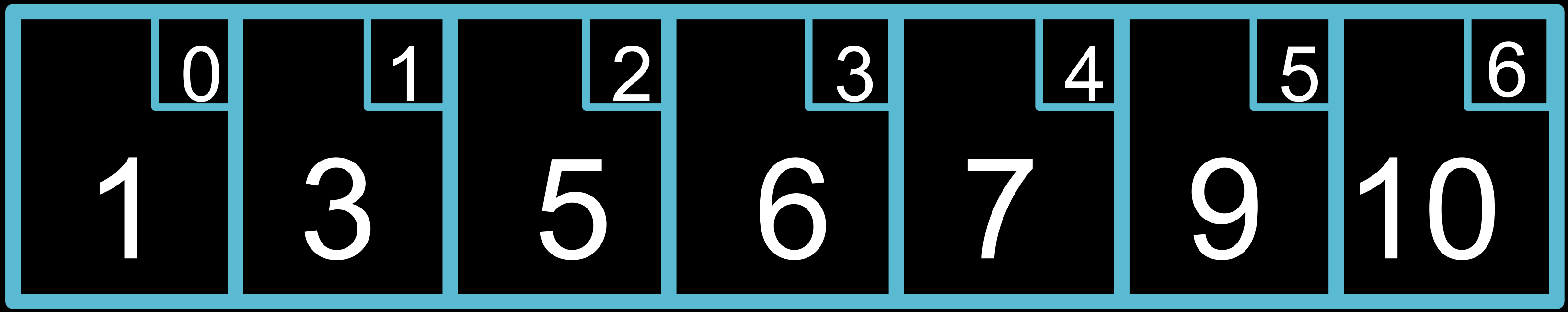
```c
int main(int argc, string argv[])
{
    if (argc != 2)
    {
        printf("Usage: mario height");
        return 1;
    }

        int height = atoi(argv[1]);

    // etc
    . . .
}
```

Binary Search

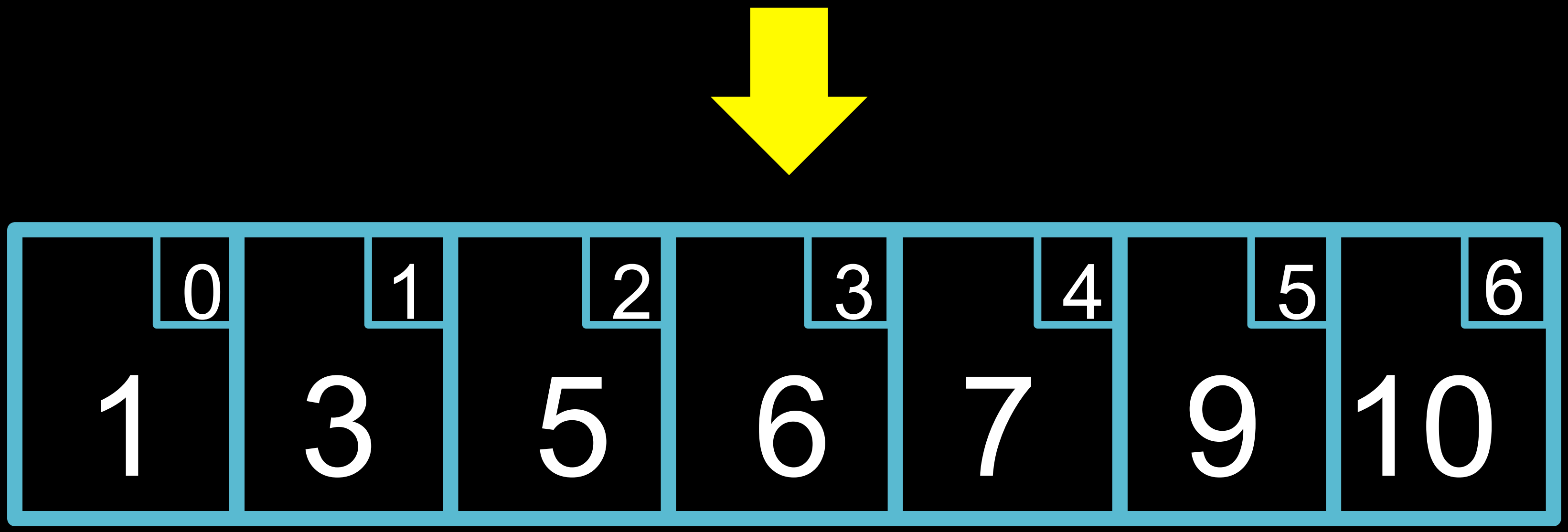



n
n/2
log n
time to solve
size of problem

Does the array contain 7?

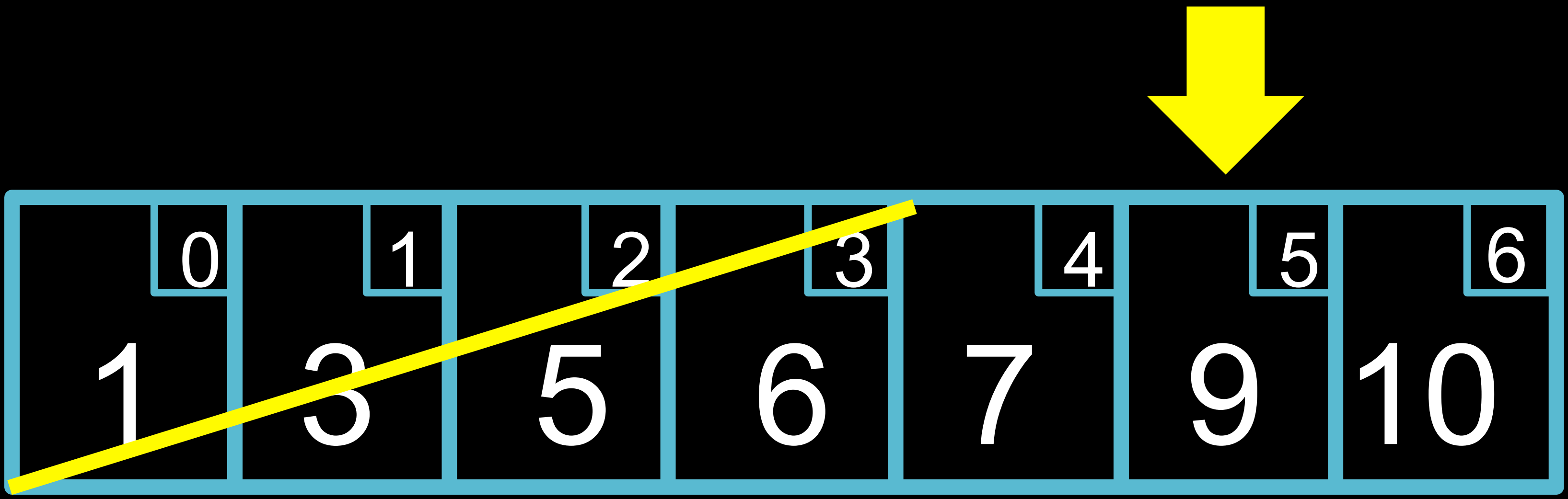


0	1	2	3	4	5	6
1	3	5	6	7	9	10



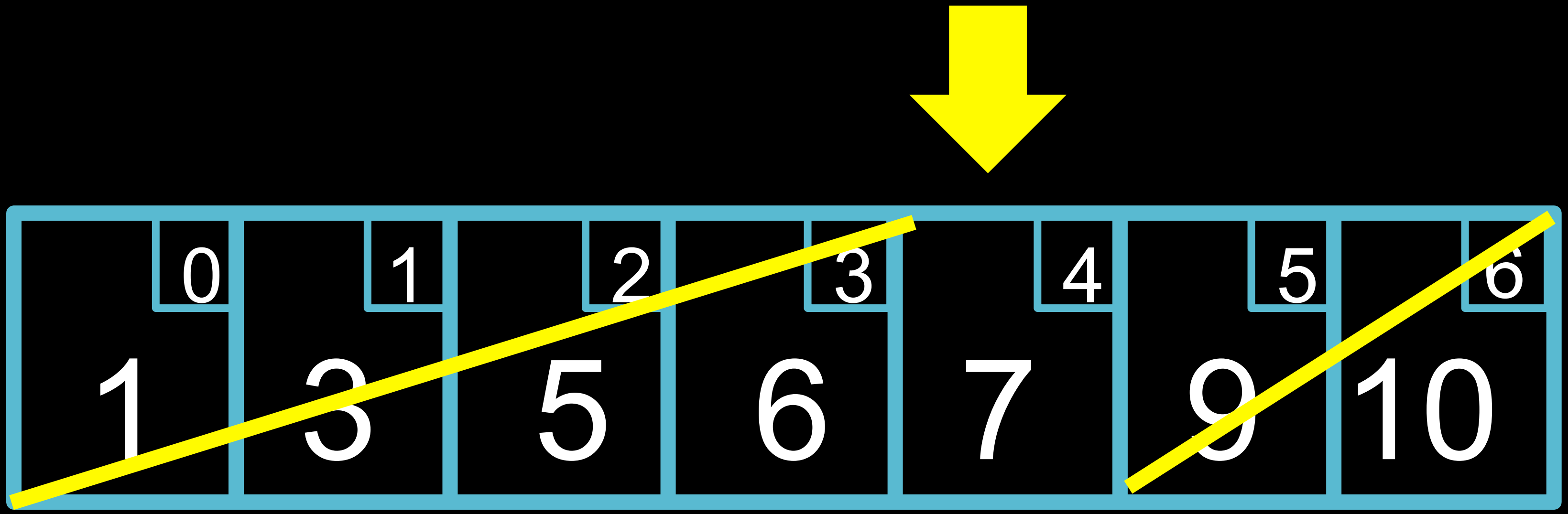
0	1	2	3	4	5	6
1	3	5	6	7	9	10

Is array[3] == 7?
Is array[3] < 7?
Is array[3] > 7?



0	1	2	3	4	5	6
1	3	5	6	7	9	10

Is array[5] == 7?
Is array[5] < 7?
Is array[5] > 7?



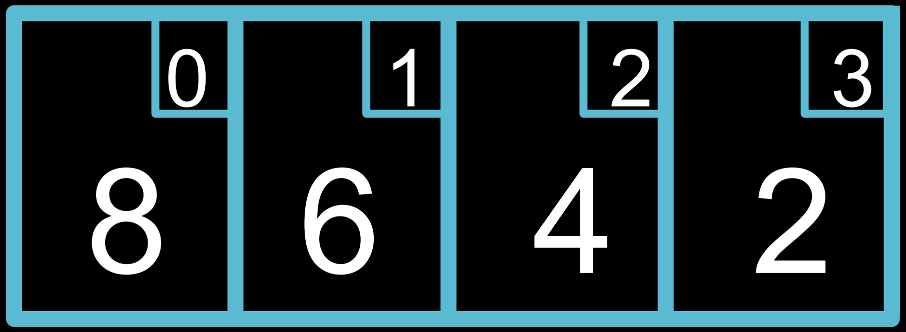
Is array[4] == 7?
Is array[4] < 7?
Is array[4] > 7?

Bubble Sort

Algorithm

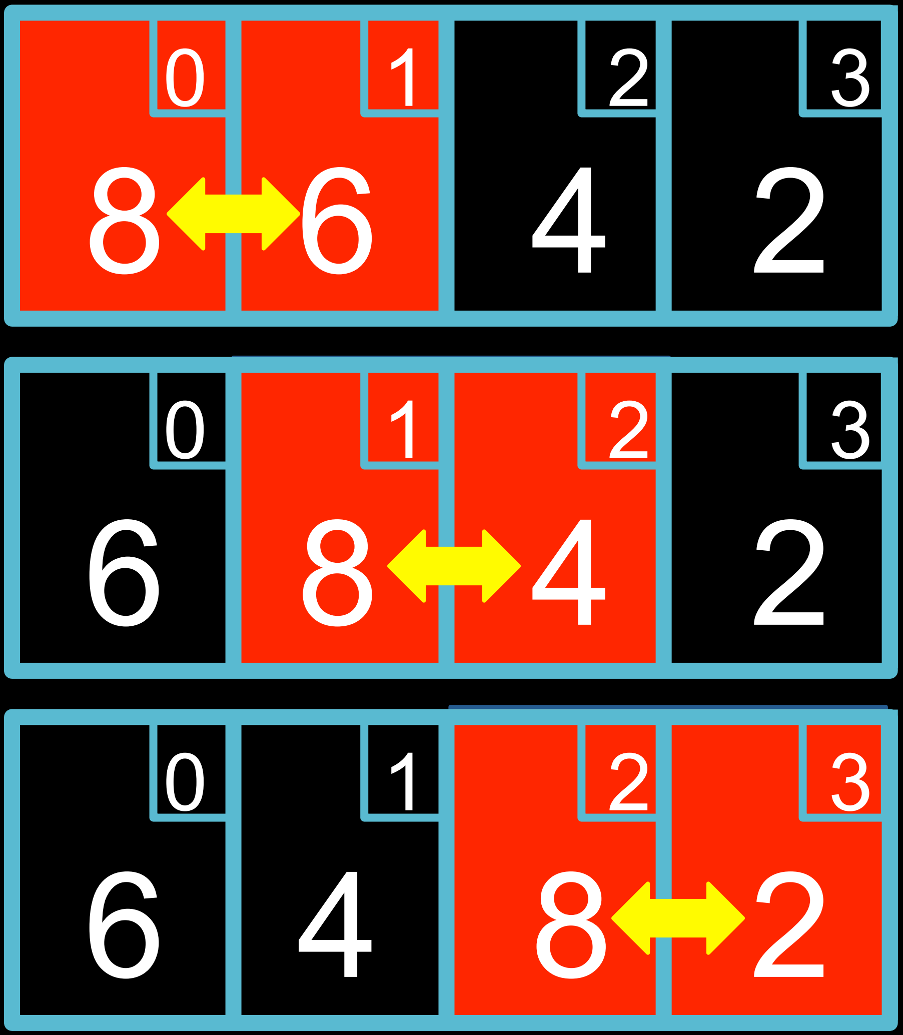
1. Step through entire list, swapping adjacent values if not in order

2. Repeat from step 1 if any swaps have been made

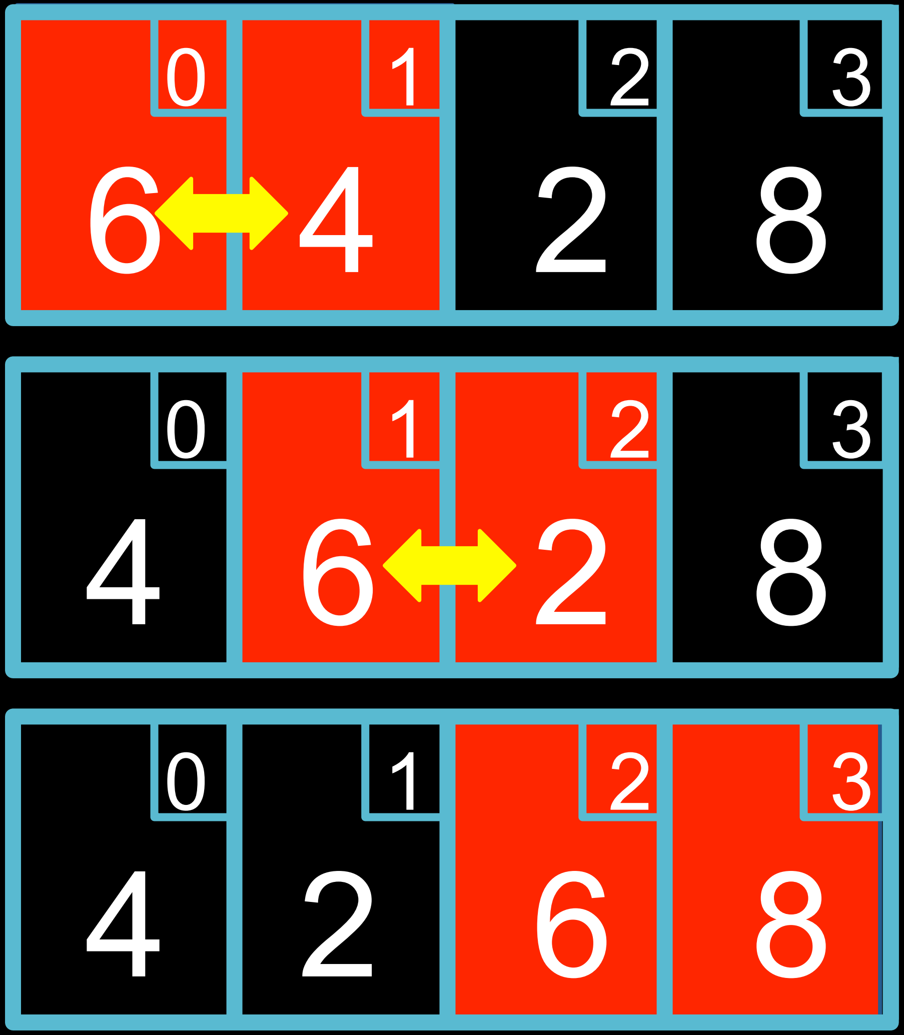


0	1	2	3
8	6	4	2

First pass: 3 swaps



Second pass: 2 swaps

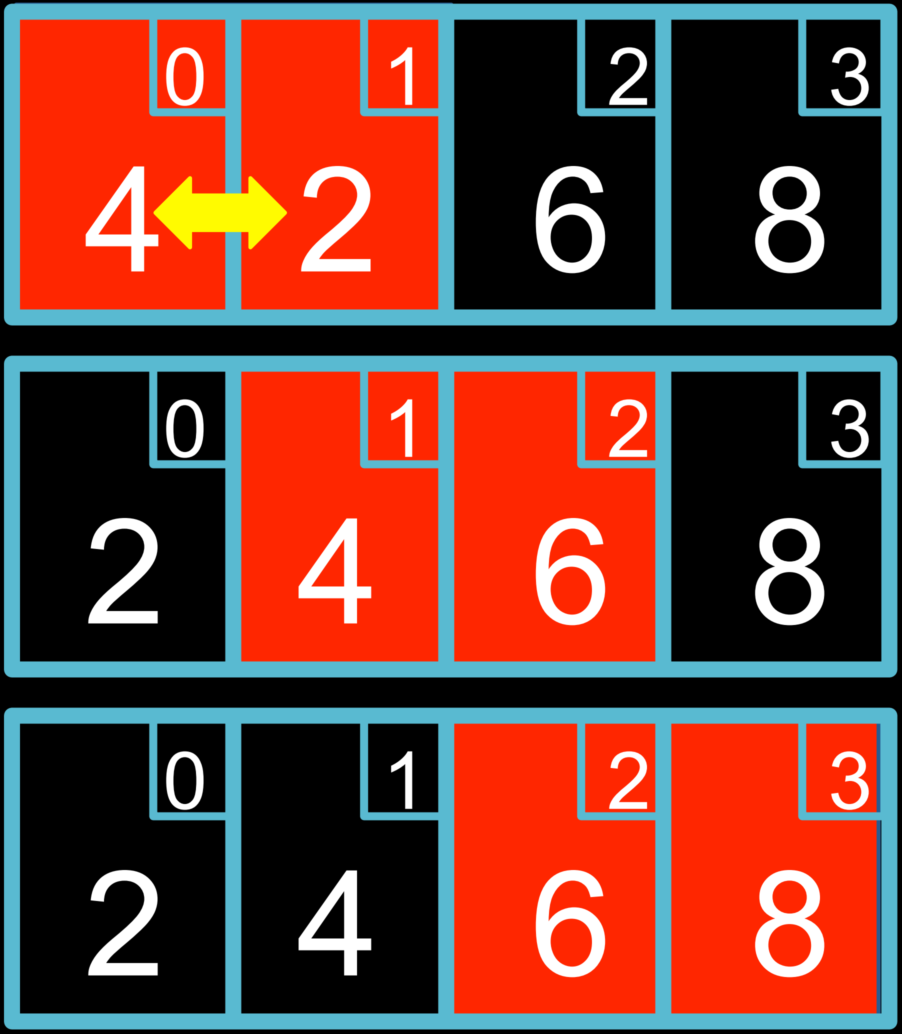


0	1	2	3
6 ↔	4	2	8

0	1	2	3
4	6 ↔	2	8

0	1	2	3
4	2	6	8

Third pass: 1 swap

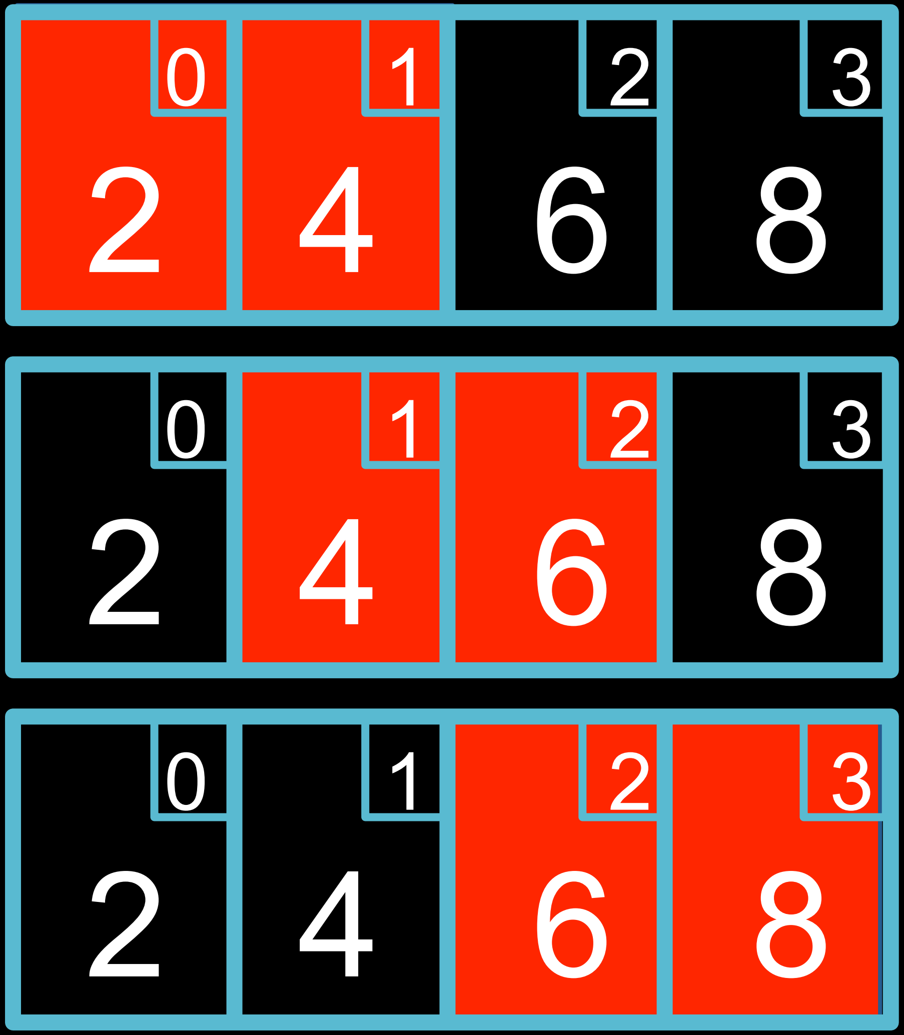


0	1	2	3
4	2	6	8

0	1	2	3
2	4	6	8

0	1	2	3
2	4	6	8

Fourth pass: 0 swaps



0	1	2	3
2	4	6	8

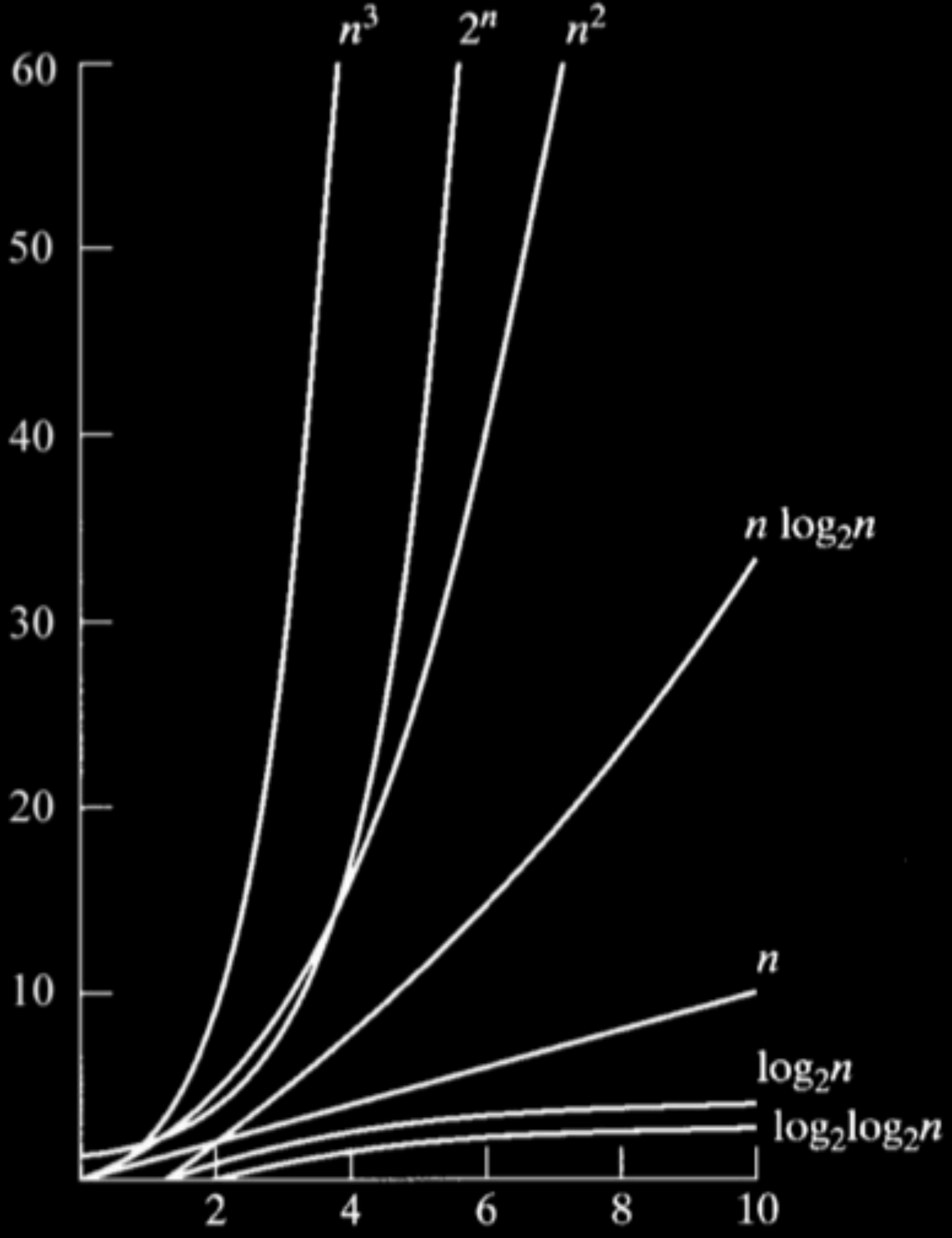
0	1	2	3
2	4	6	8

0	1	2	3
2	4	6	8

```
initialize counter
do
{
    set counter to 0

    iterate through entire array
        if array[n] > array[n+1]
            swap them
            increment counter
}
while (counter > 0)
```

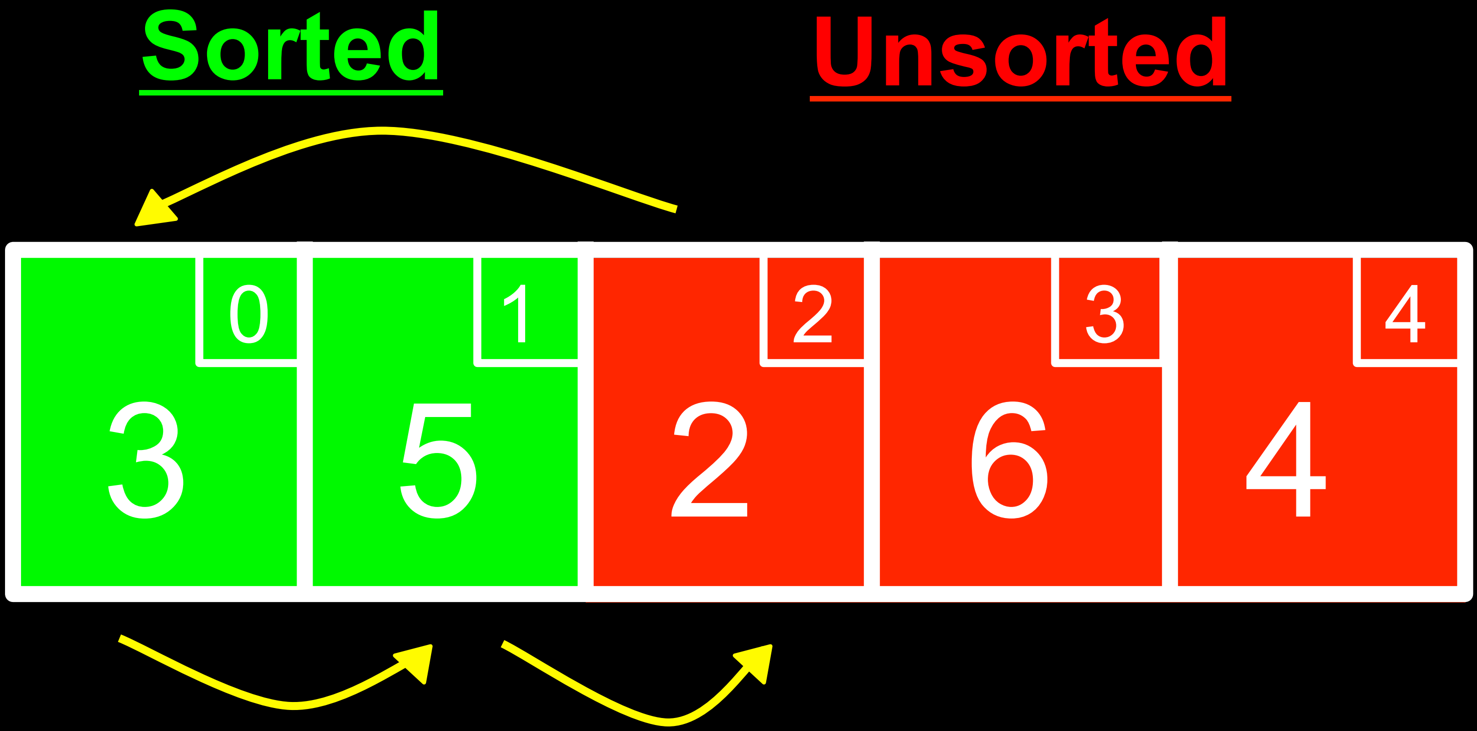
What's the worst case runtime of bubble sort?

What's the best case runtime of bubble sort?


n^3
2^n
n^2
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
60
50
40
30
20
10
2
4
6
8
10

	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$n \log n$
Ω	n	n^2	n	$n \log n$
Θ		n^2		$n \log n$

Insertion Sort



Sorted

Unsorted

0	1	2	3	4
3	5	2	6	4

All values start as Unsorted

Sorted

Unsorted

0	1	2	3	4
3	5	2	6	4

Add first value to Sorted

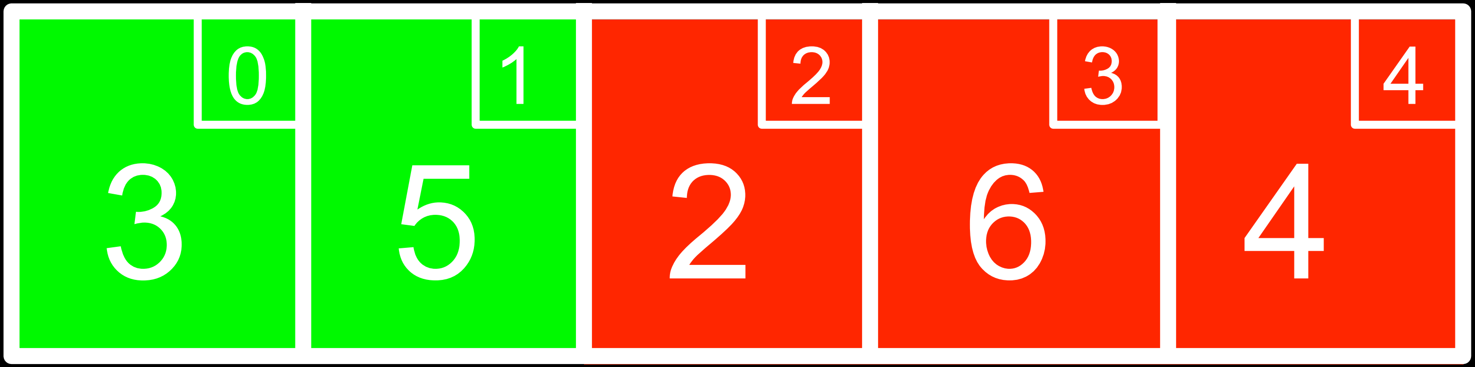
Sorted

Unsorted

0	1	2	3	4
3	5	2	6	4

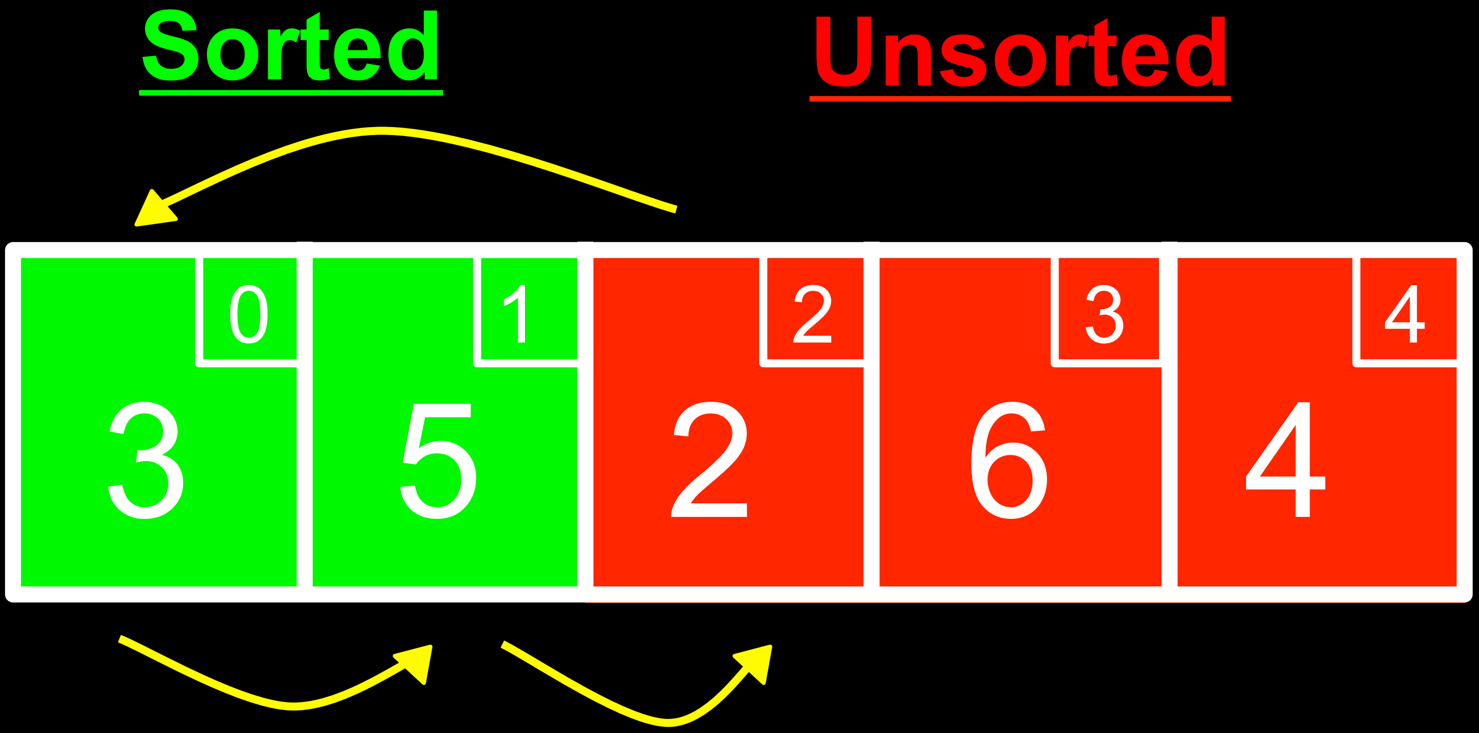
5 > 3
insert 5 to right of 3

Sorted **Unsorted**



0	1	2	3	4
3	5	2	6	4

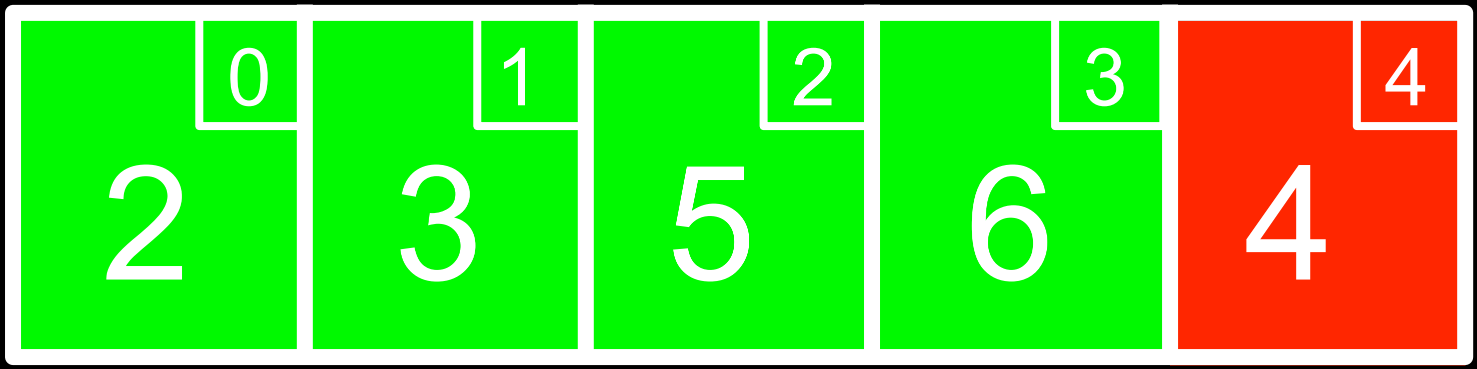
2 < 5 and 2 < 3
shift 3 and 5
insert 2 to left of 3



6 > 5
insert 6 to right of 5

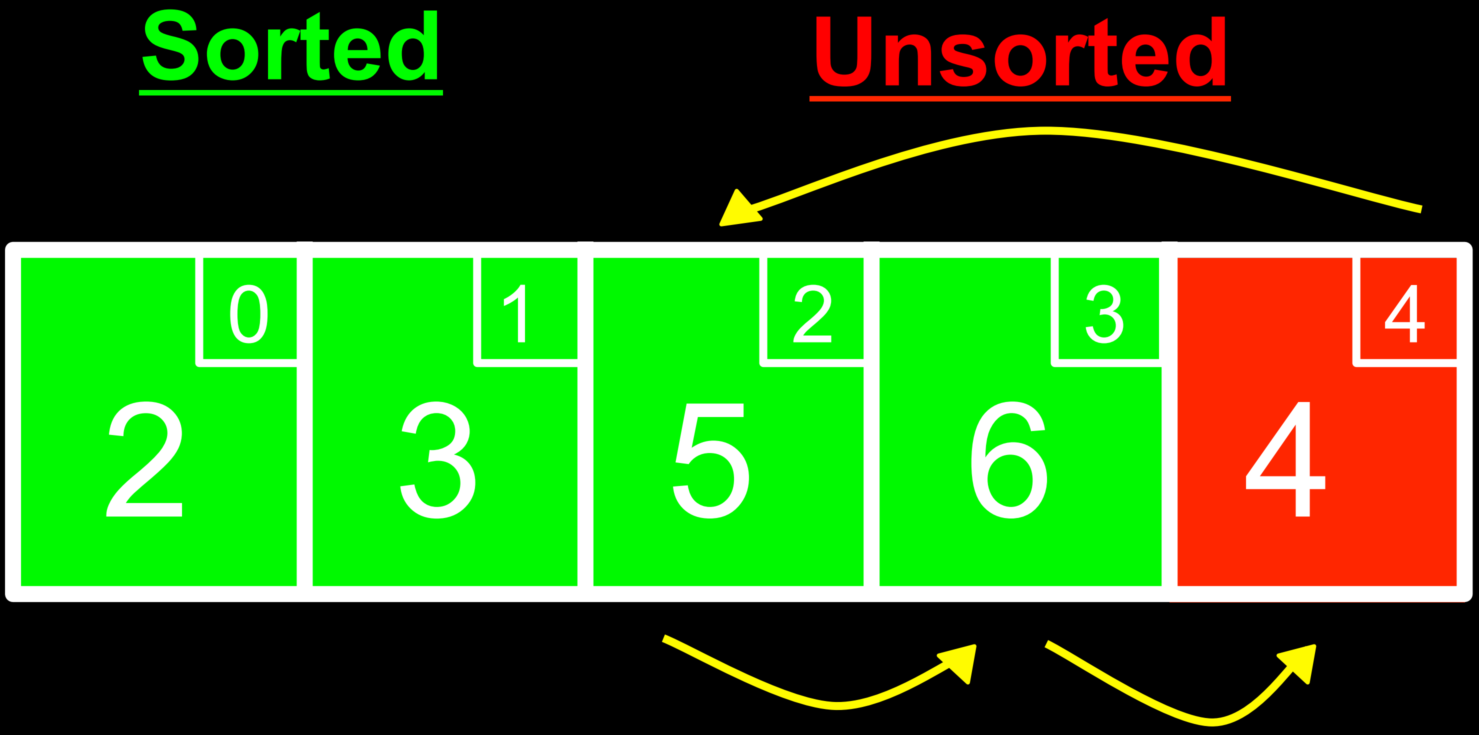
Sorted

Unsorted



0	1	2	3	4
2	3	5	6	4

4 < 6, 4 < 5, and 4 > 3
shift 5 and 6
insert 4 to right of 3



For each unsorted element n:

1. Determine where in sorted portion of the list to insert n

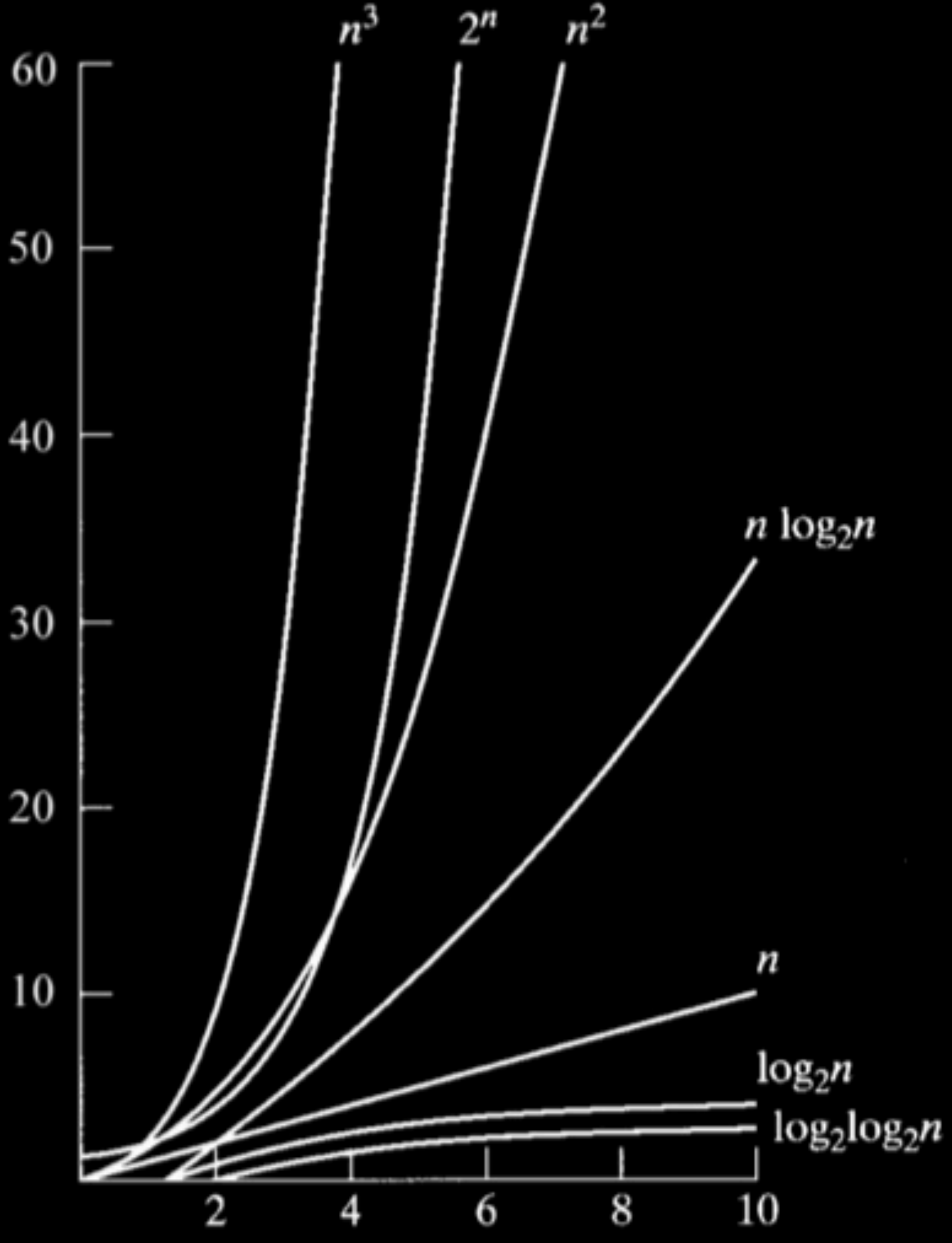
2. Shift sorted elements rightwards as necessary to make room for n

3. Insert n into sorted portion of the list

```
for i = 0 to n - 1
    element = array[i]
    j = i
    while (j > 0 and array[j - 1] > element
        array[j] = array[j - 1]
        j = j - 1
    array[j] = element
```

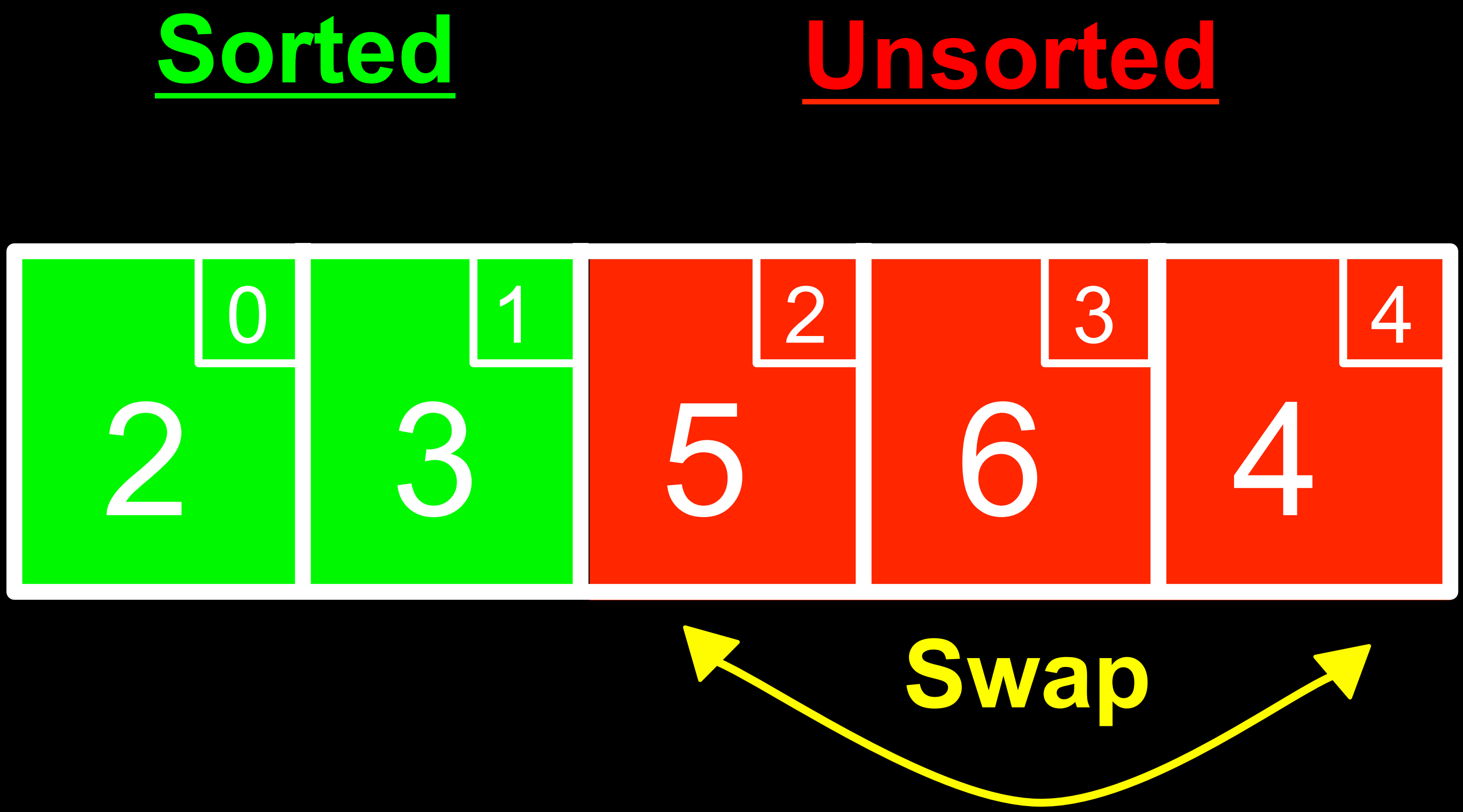
What's the worst case runtime of insertion sort?

What's the best case runtime of insertion sort?


n^3
2^n
n^2
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
60
50
40
30
20
10
2
4
6
8
10

	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$nlogn$
Ω	n	n^2	n	$nlogn$
Θ		n^2		$nlogn$

Selection Sort



Sorted

Unsorted

0	1	2	3	4
2	3	5	6	4

Swap

Algorithm

1. Find the smallest unsorted value

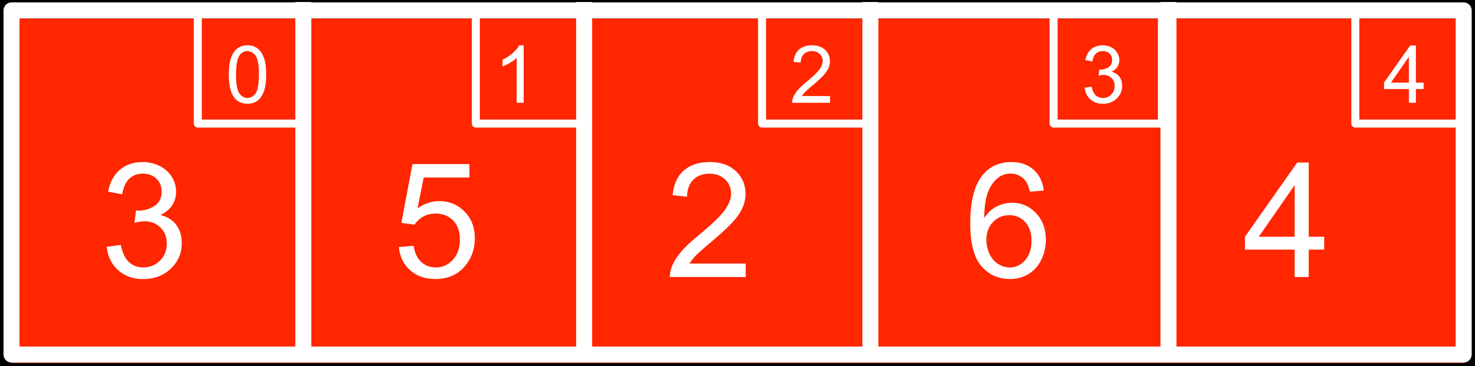
2. Swap that value with the first unsorted value

3. Repeat from Step 1 if there are still unsorted items

All values start as Unsorted

Sorted

Unsorted

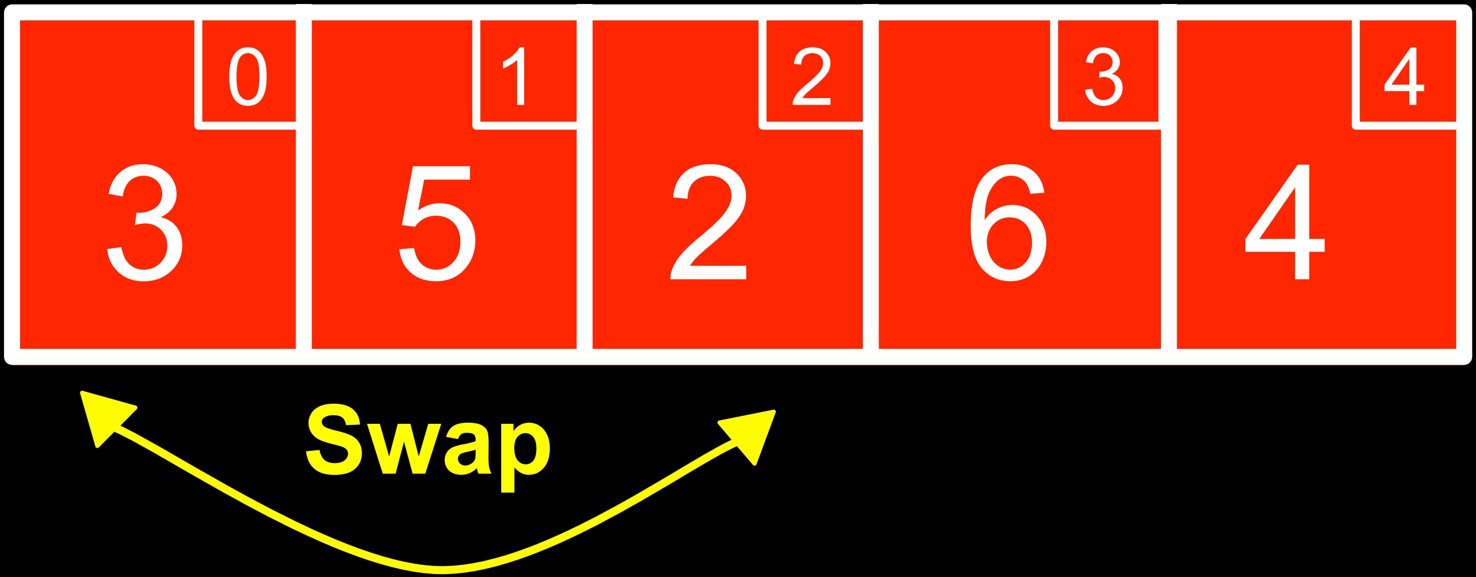


0	1	2	3	4
3	5	2	6	4

First pass:
2 is smallest, swap with 3

Sorted

Unsorted



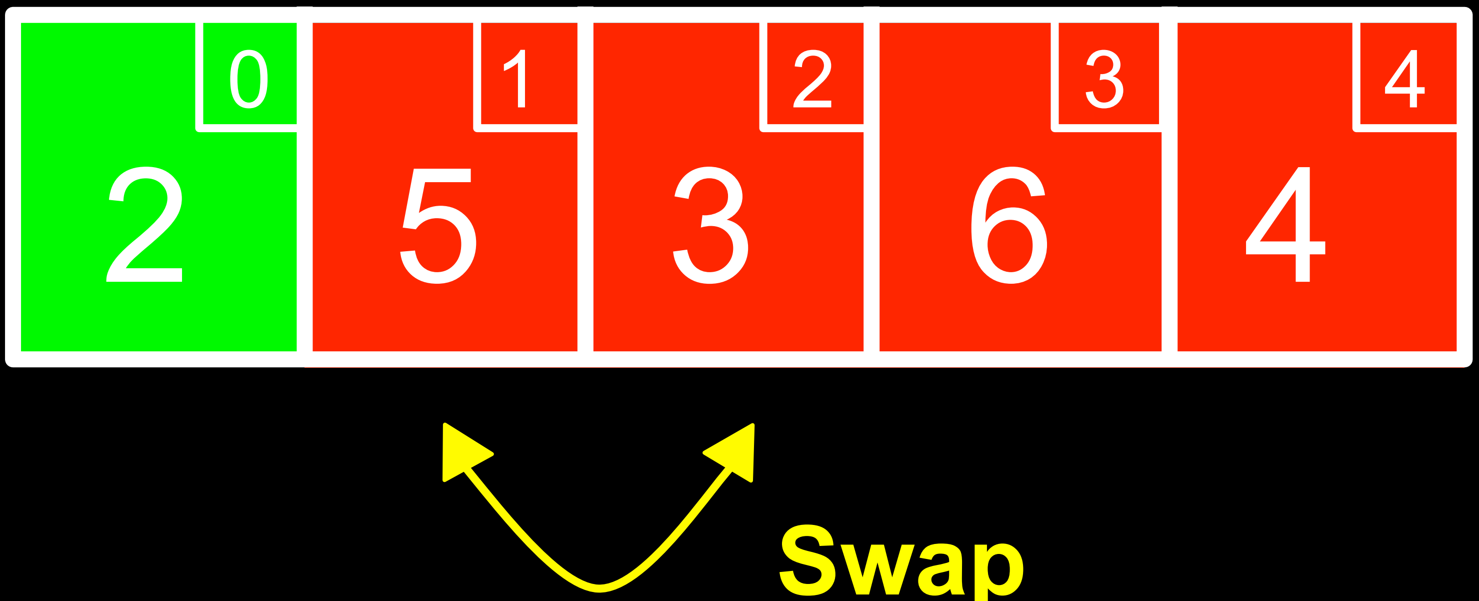
0	1	2	3	4
3	5	2	6	4

Swap

Second pass:
3 is smallest, swap with 5

Sorted

Unsorted



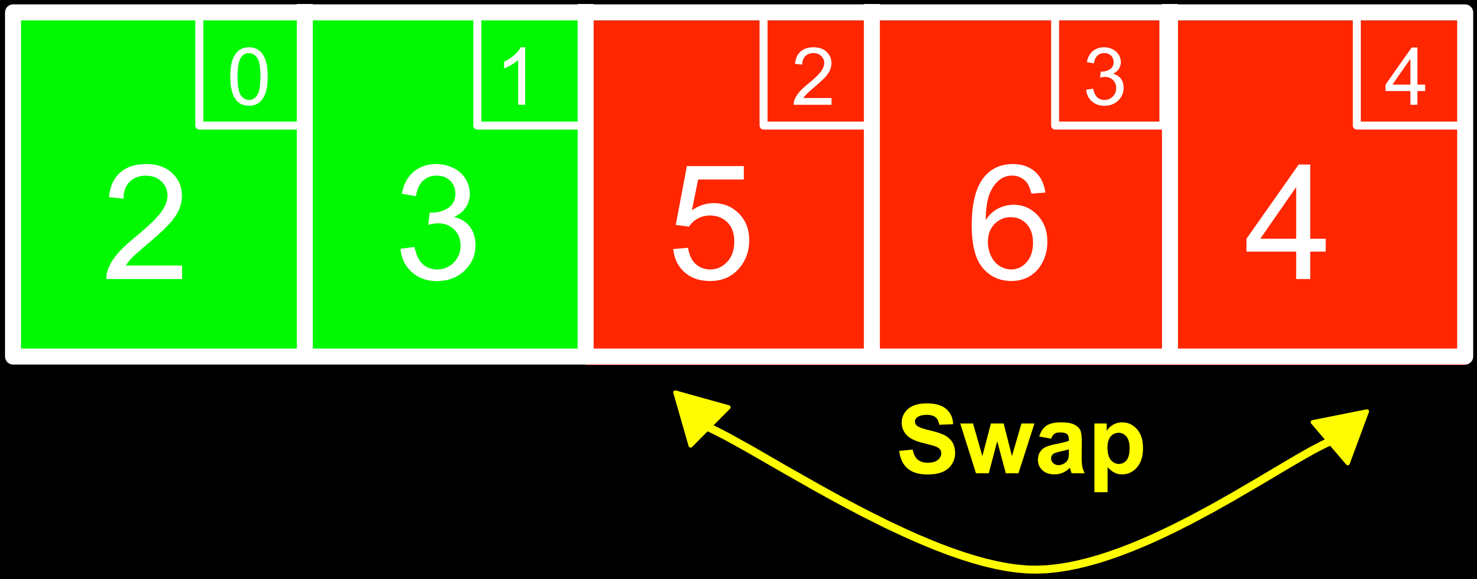
0	1	2	3	4
2	5	3	6	4

Swap

Third pass:
4 is smallest, swap with 5

Sorted

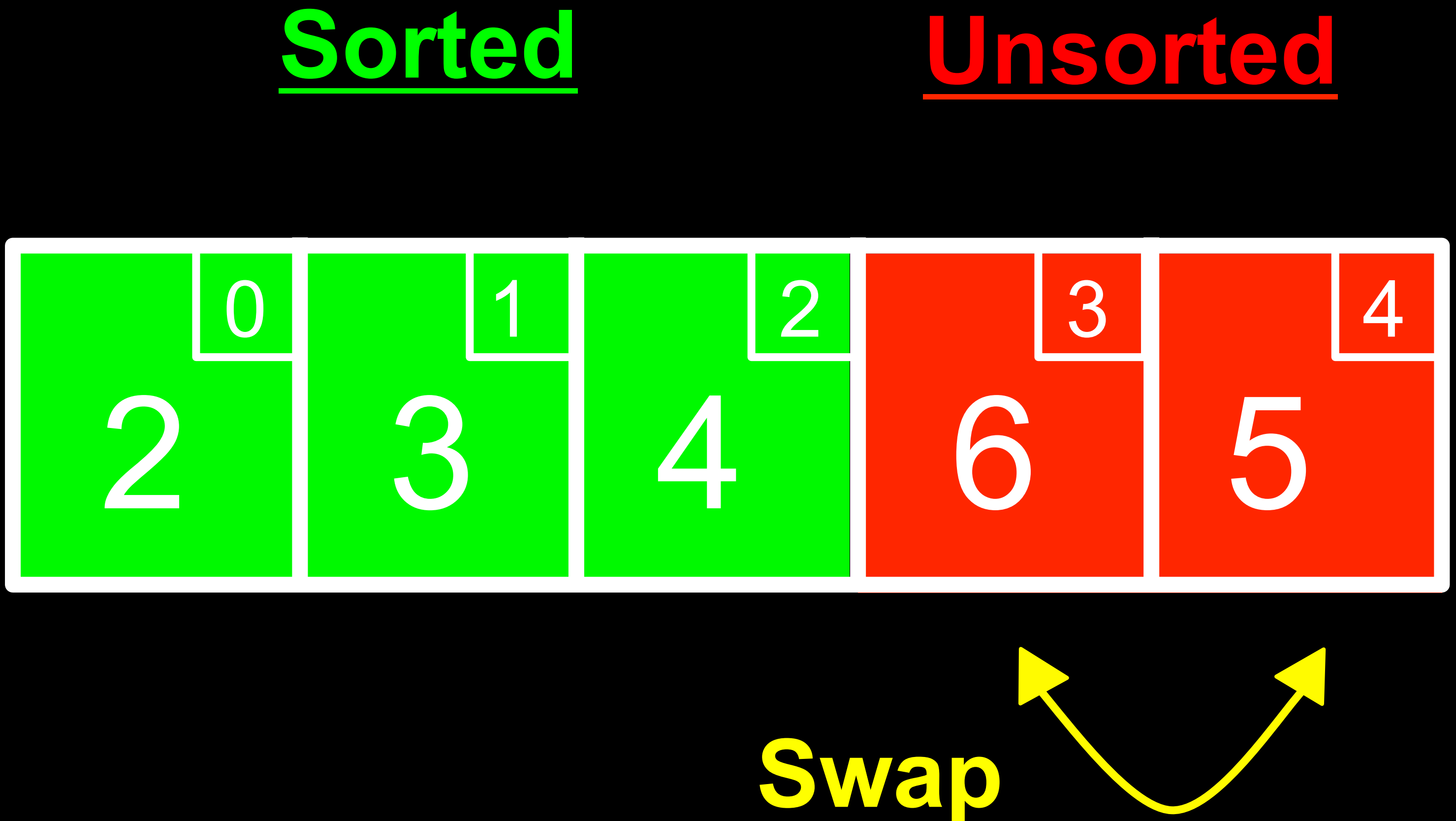
Unsorted



0	1	2	3	4
2	3	5	6	4

Swap

Fourth pass:
5 is smallest, swap with 6



Sorted

Unsorted

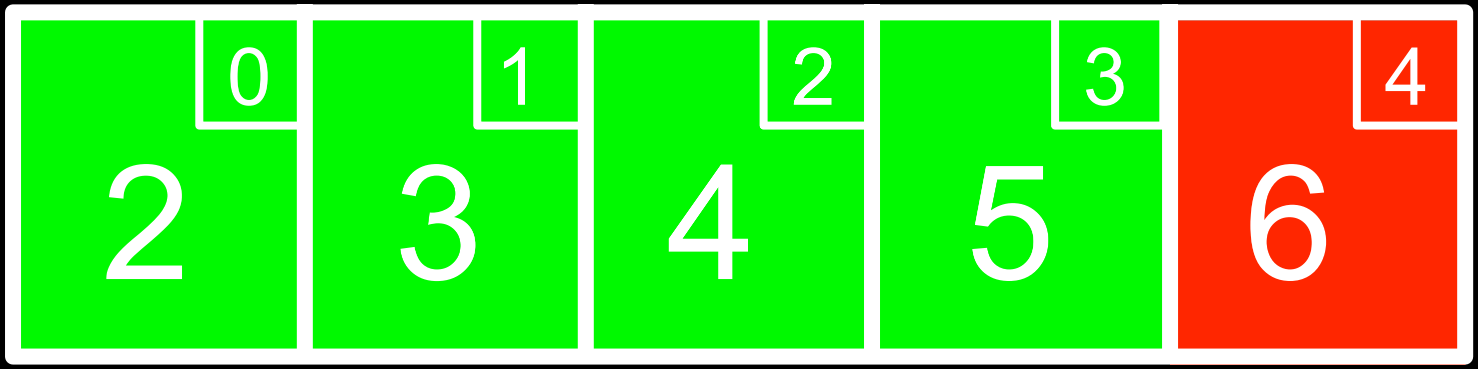
0	1	2	3	4
2	3	4	6	5

Swap

Fifth pass:
6 is the only value left, done!

Sorted

Unsorted



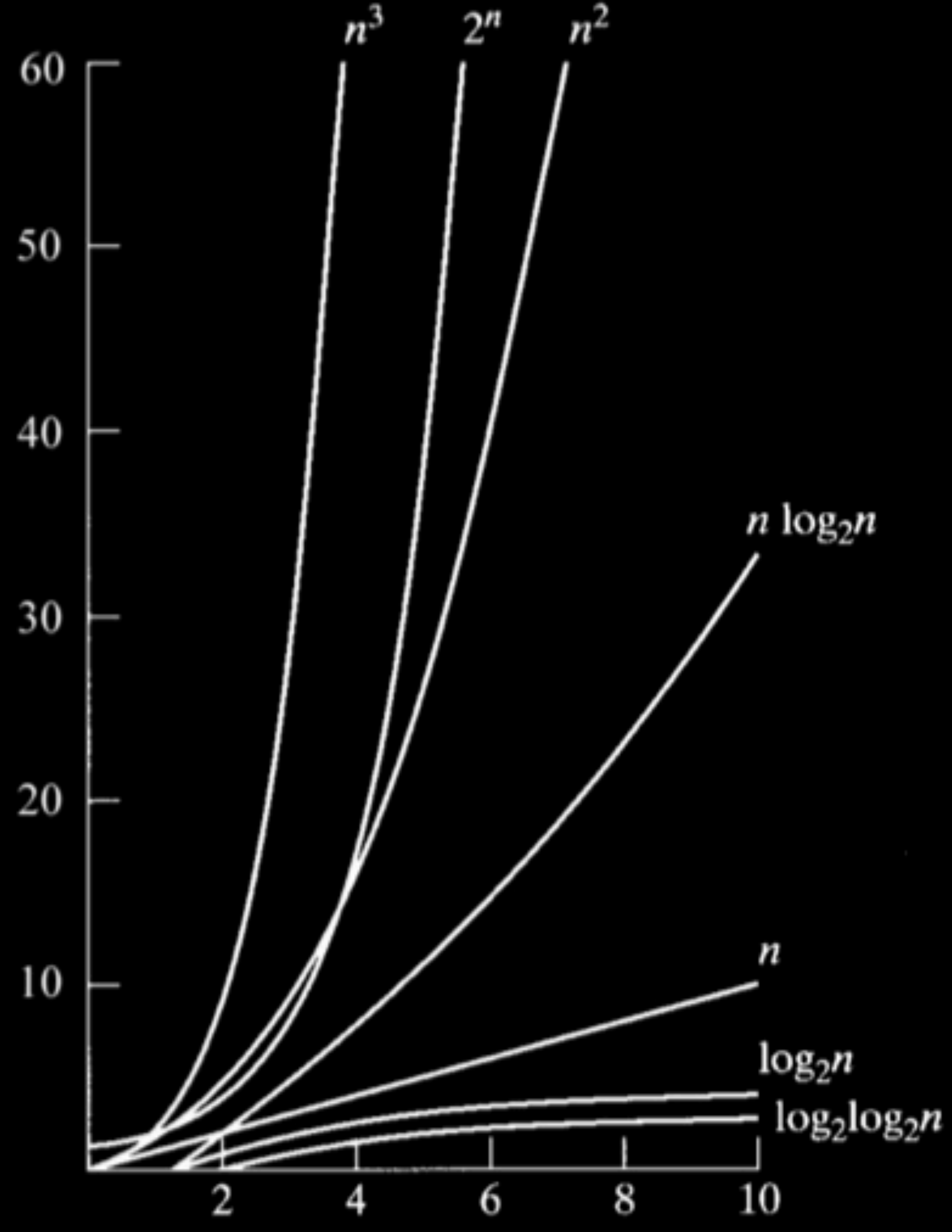
0	1	2	3	4
2	3	4	5	6

```
for i = 0 to n - 2
        min = i
        for j = i + 1 to n - 1
            if array[j] < array[min]
                    min = j;
        if min != i
                swap array[min] and array[i]
```

What's the best case runtime of selection sort?

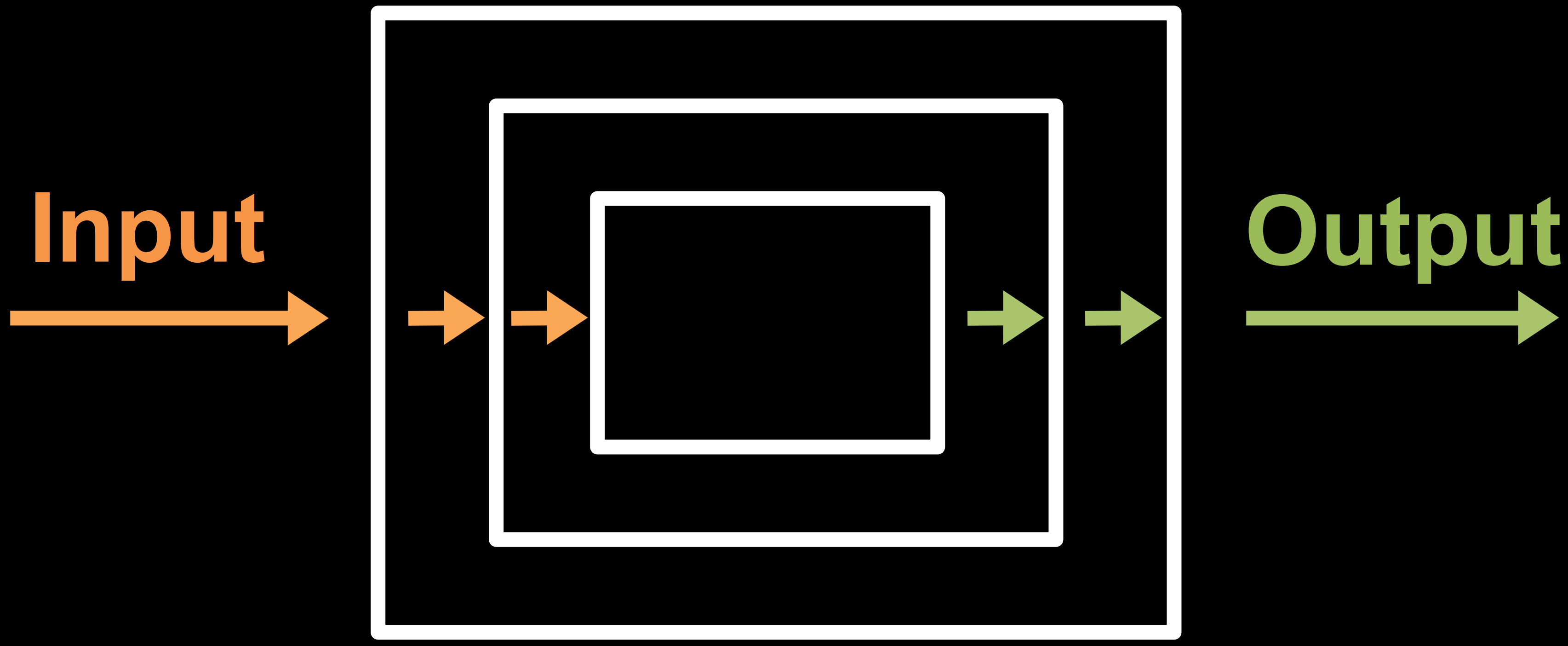
What's the worst case runtime of selection sort?

What's the expected runtime of selection sort?


n^3
2^n
n^2
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
60
50
40
30
20
10
2
4
6
8
10

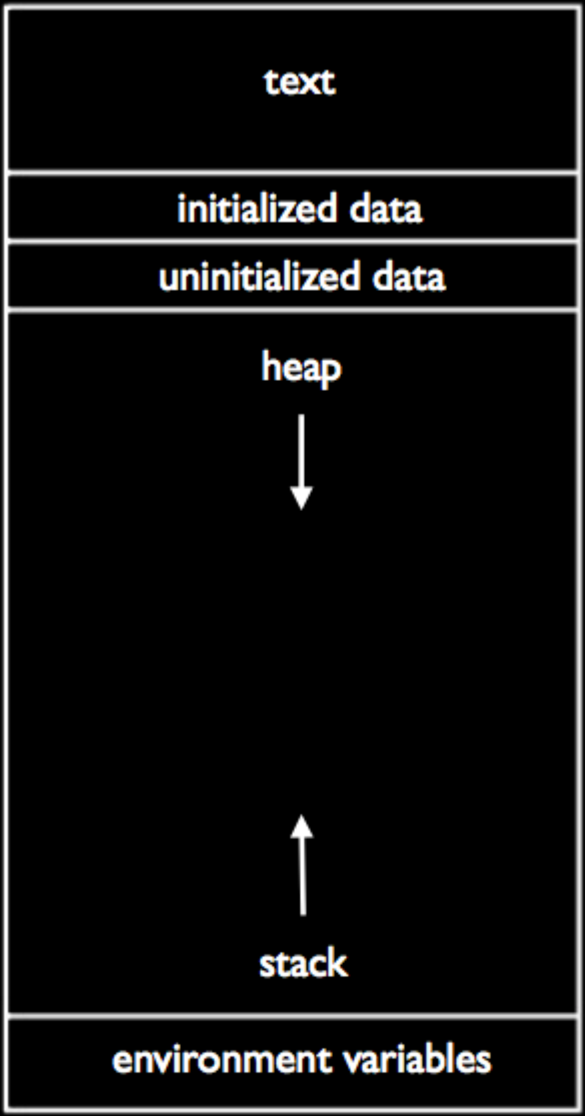
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$nlogn$
Ω	n	n^2	n	$nlogn$
Θ		n^2		$nlogn$

Recursion



Recursion w/out a Base Case

```
void foo(string str)
{
    printf(“%s\n”, str);
    foo(str);
}
```

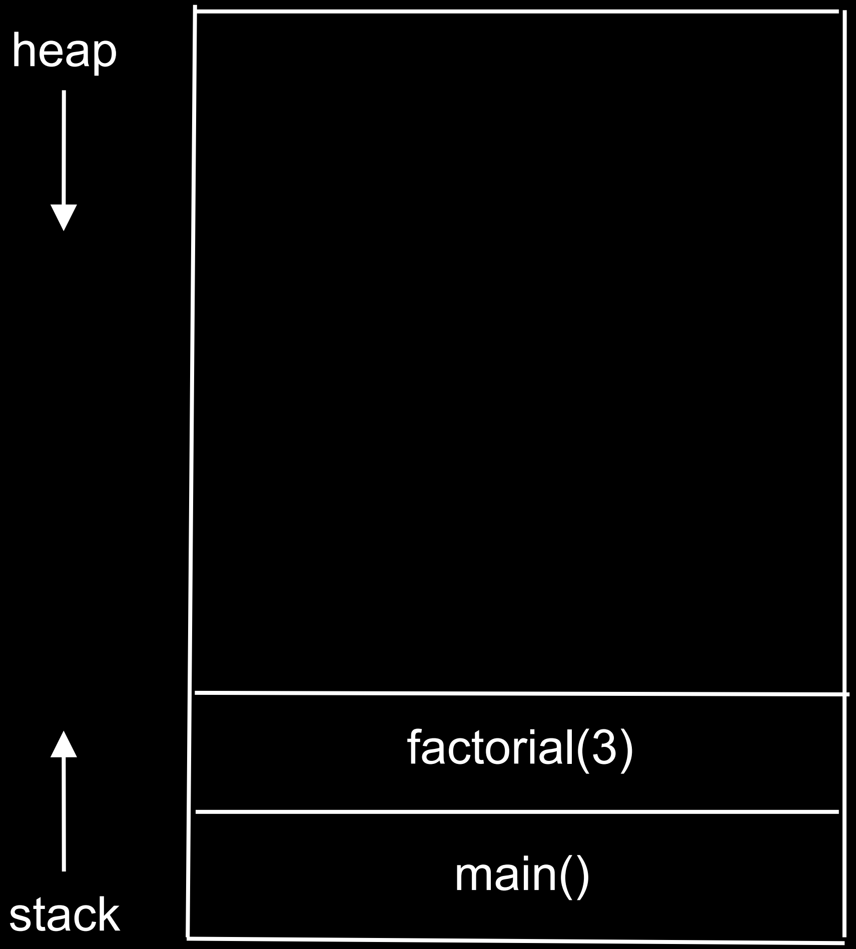


Factorial

$$n! = n * (n - 1) * (n - 2) * \ldots * 1$$

```
unsigned int factorial(unsigned int n)
{
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return n * factorial(n - 1);
    }
}
```

$$\texttt{factorial(3) = 3 * }\underbrace{\texttt{factorial(2)}}_{\texttt{2 * }\underbrace{\texttt{factorial(1)}}_{\texttt{1}}}$$

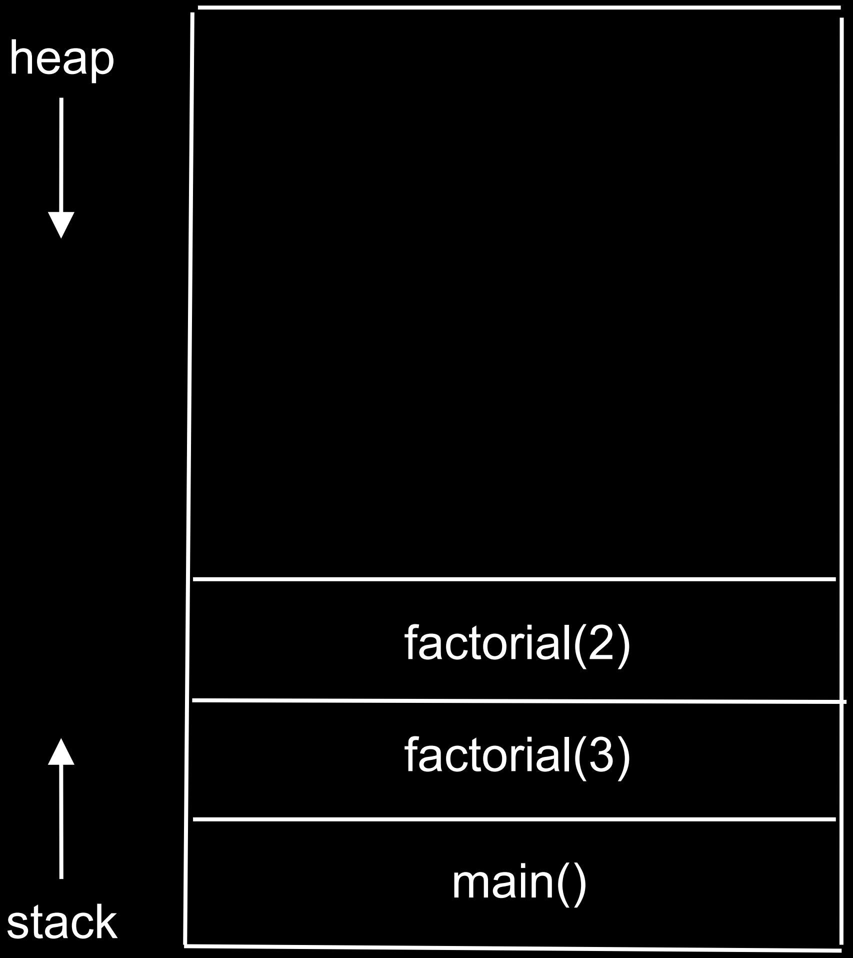


heap

factorial(3)

main()

stack



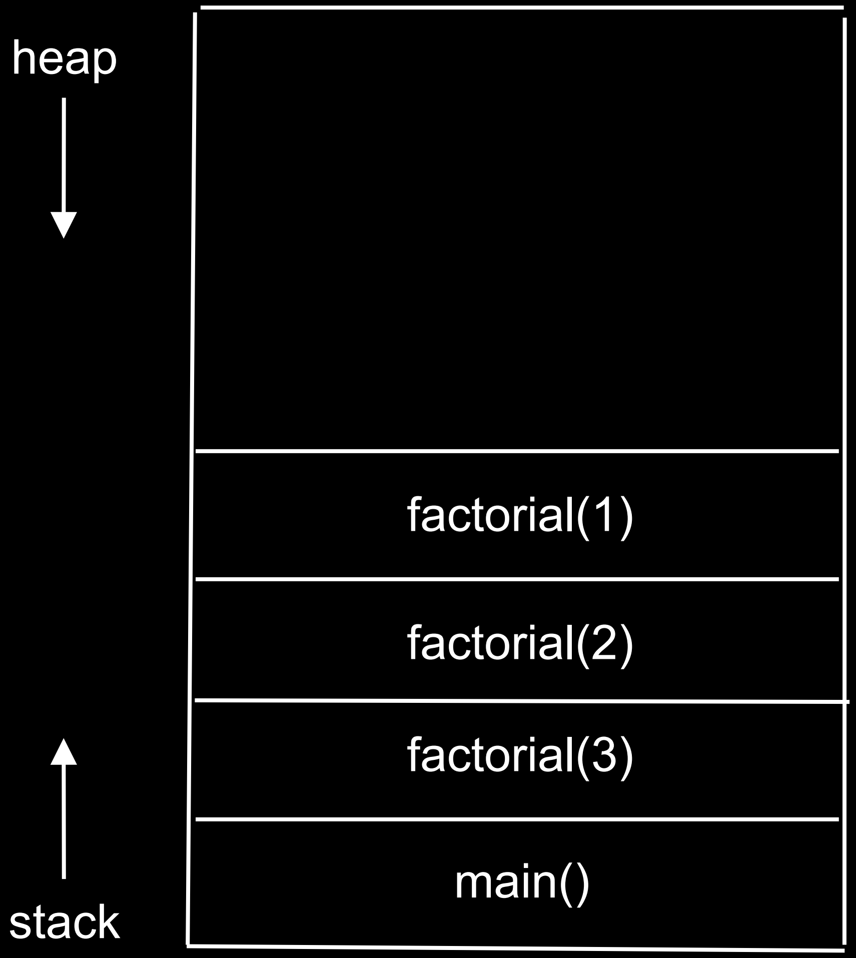
heap

factorial(2)

factorial(3)

main()

stack



heap

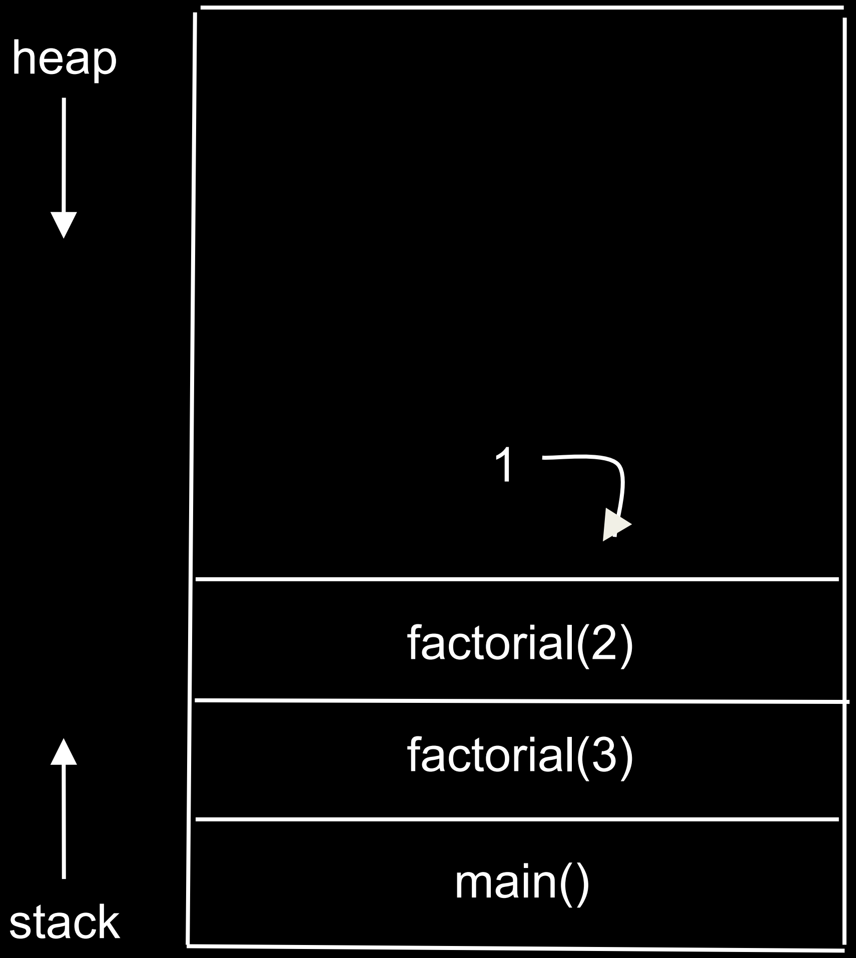
factorial(1)

factorial(2)

factorial(3)

main()

stack



heap

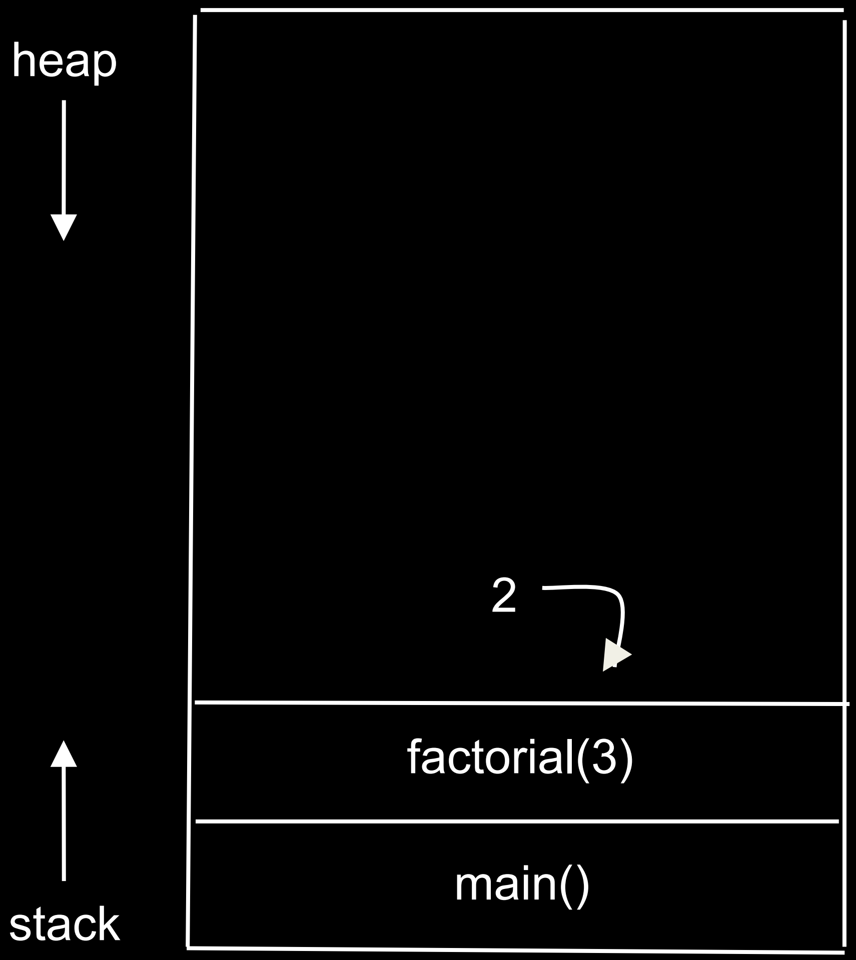
1

factorial(2)

factorial(3)

main()

stack



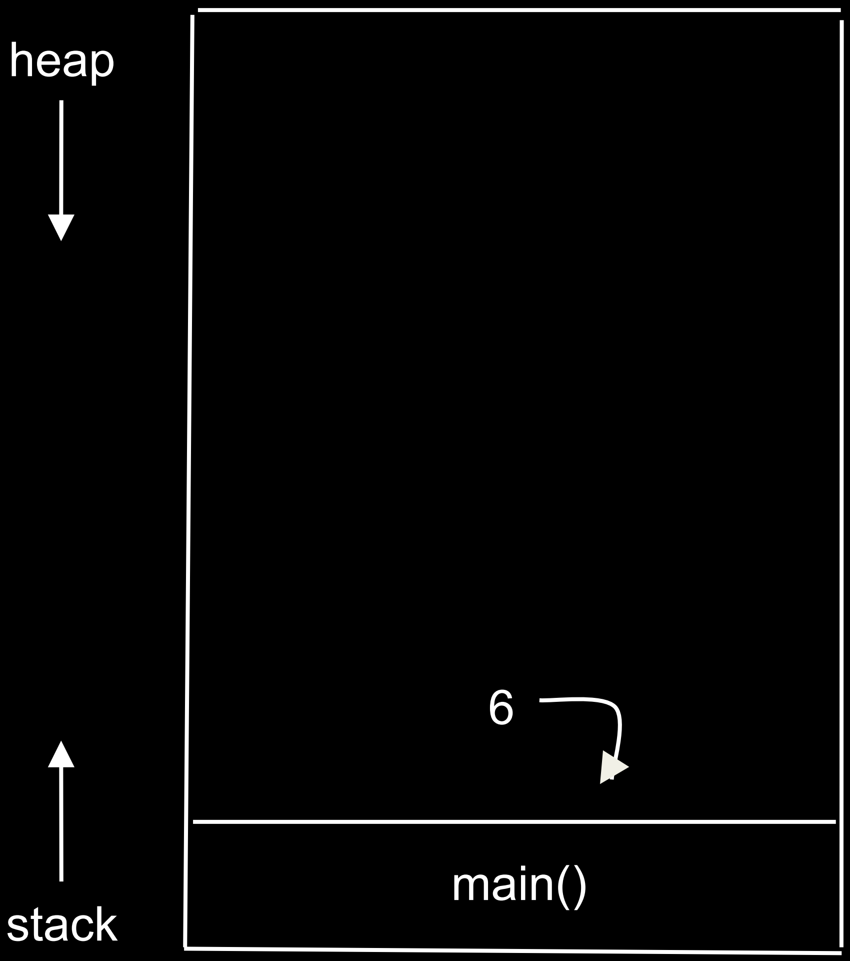
heap

2

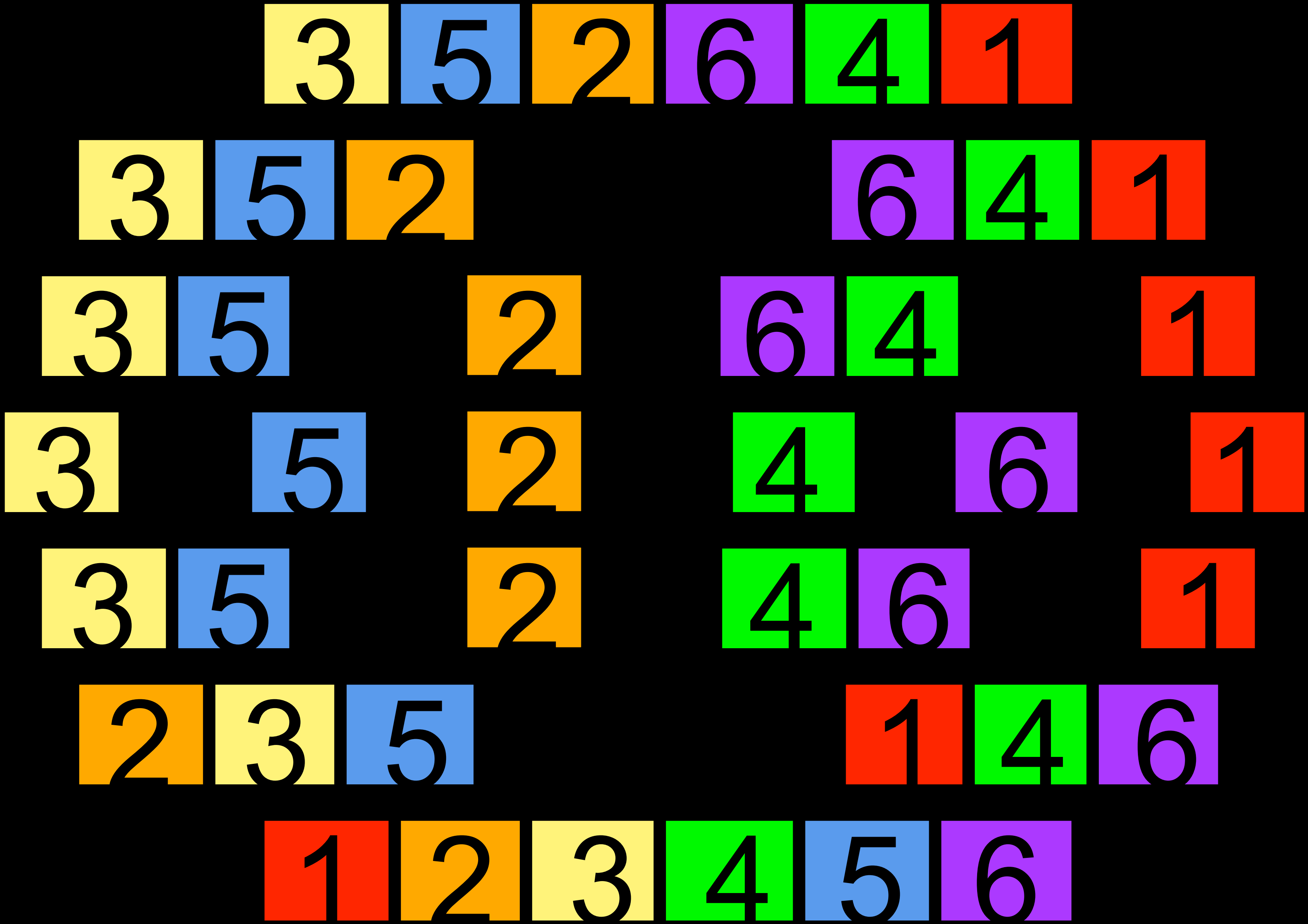
factorial(3)

main()

stack


heap
6
main()
stack

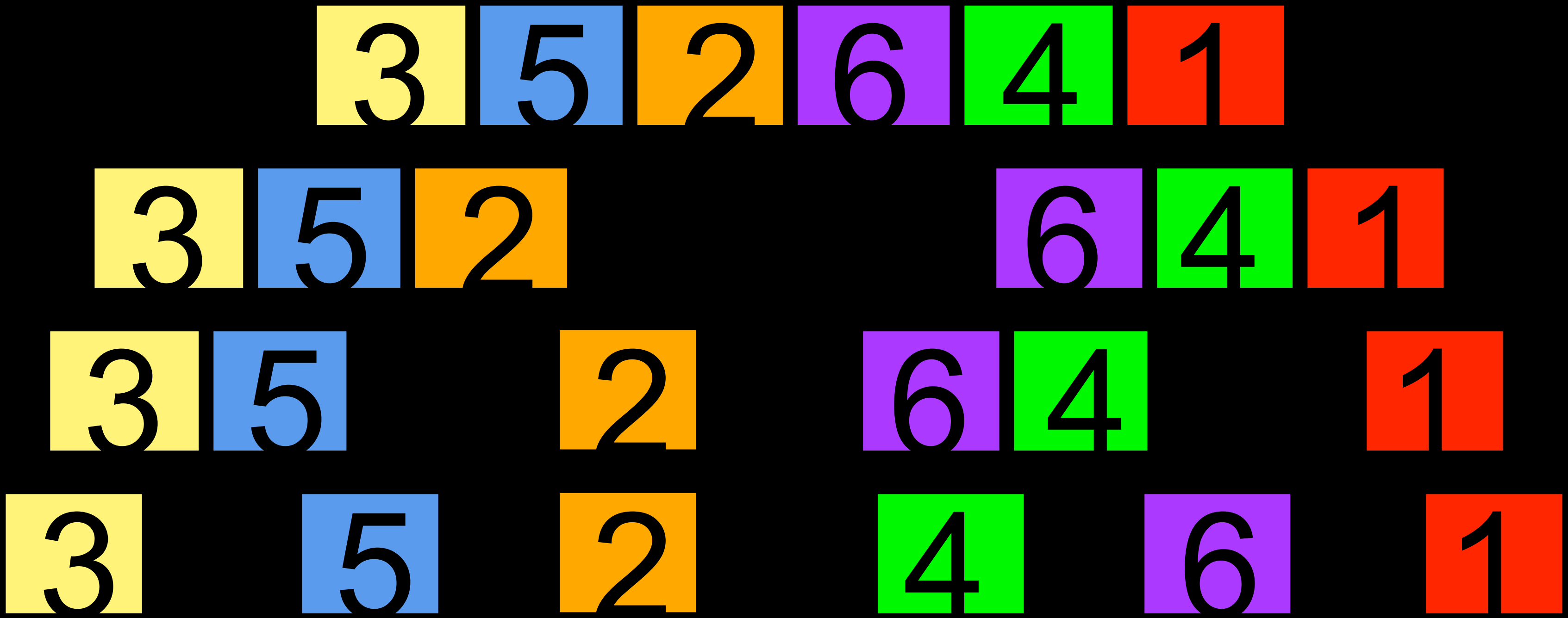
Merge Sort



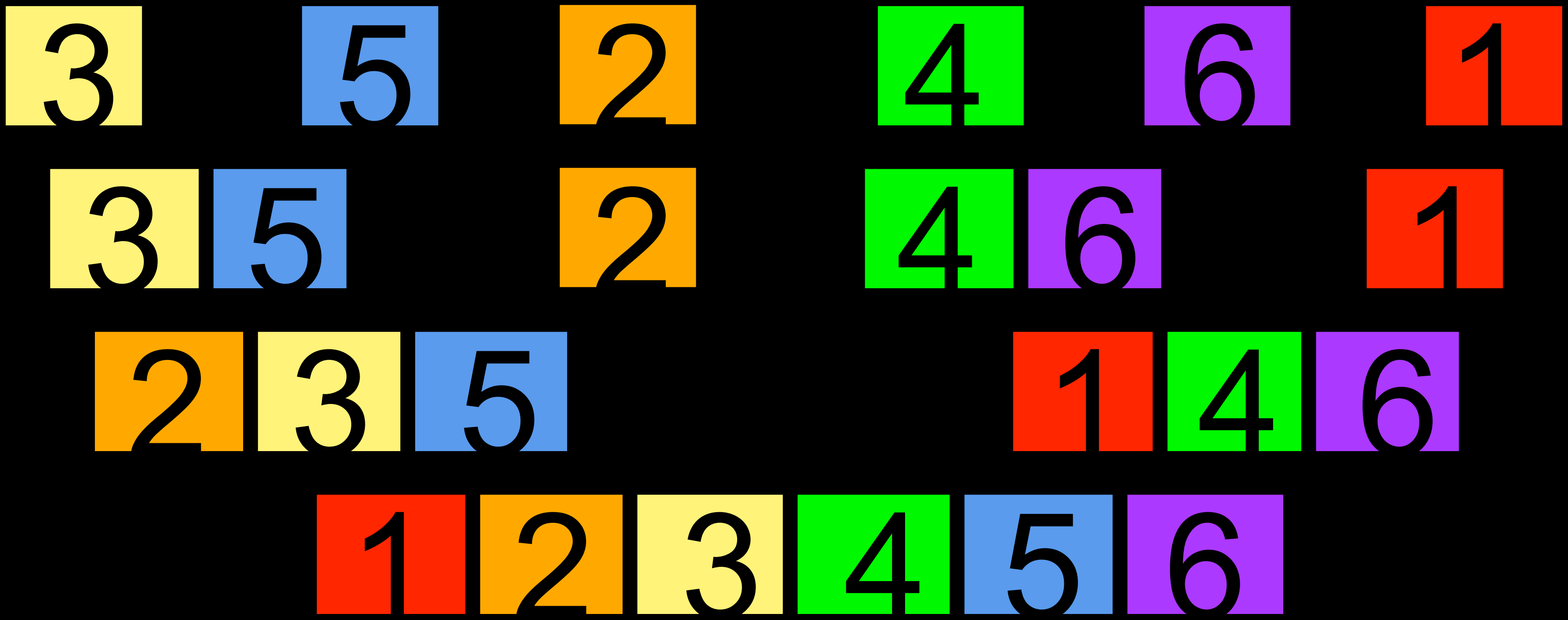
```
On input of n elements:
    If n < 2
        Return.
    Else
        Sort left half of elements.
        Sort right half of elements.
        Merge sorted halves.
```

3	5	2	6	4	1

Halve until each subarray is size 1



Merge Sorted Halves

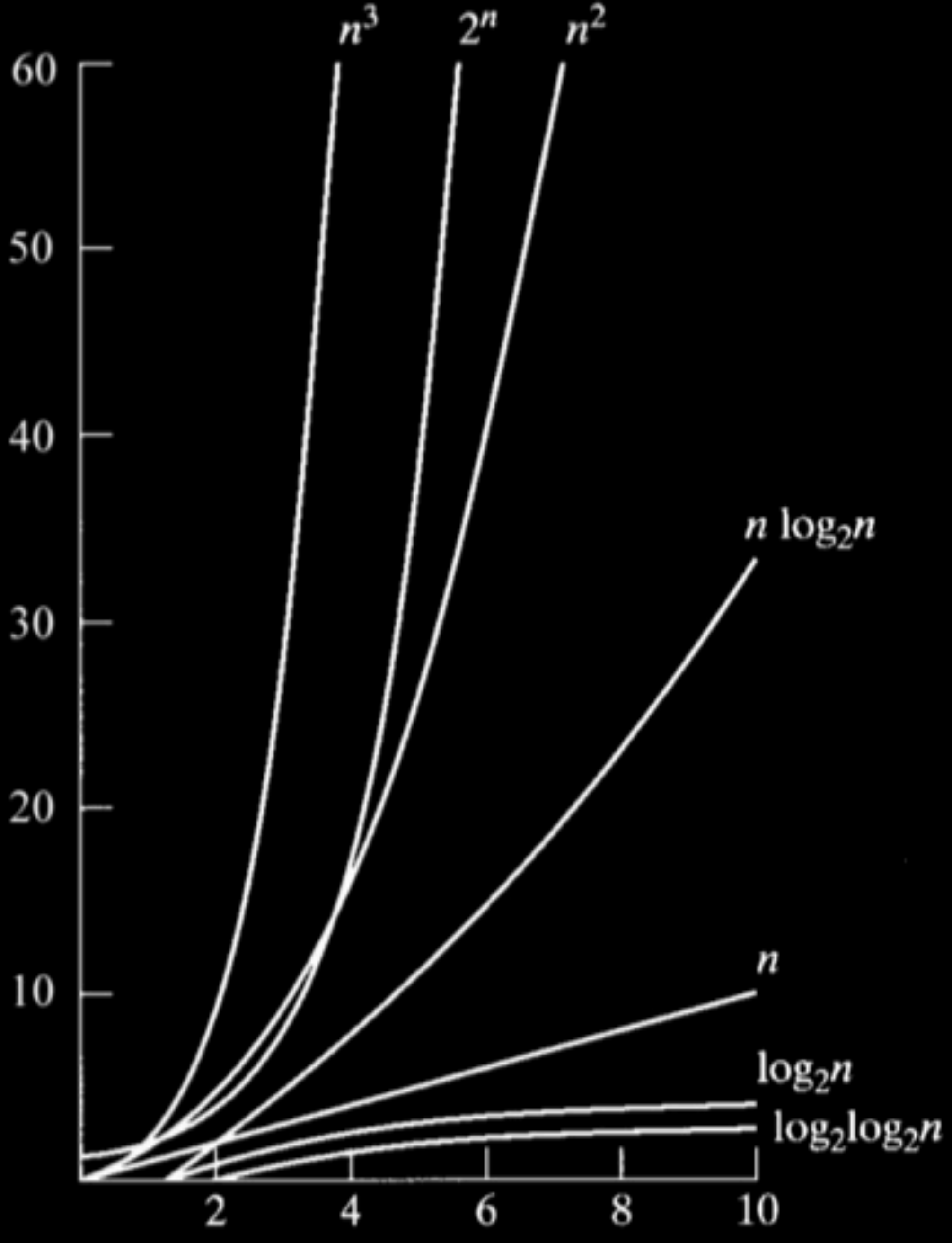


```
sort (int array[], int start, int end)
{
  if (end > start)
       {
           int middle = (start + end) / 2;

               sort(array, start, middle);
               sort(array, middle + 1, end);

               merge(array, start, middle, middle + 1, end);
       }
}
```

What's the best case runtime of merge sort?

What's the worst case runtime of merge sort?

What's the expected runtime of merge sort?

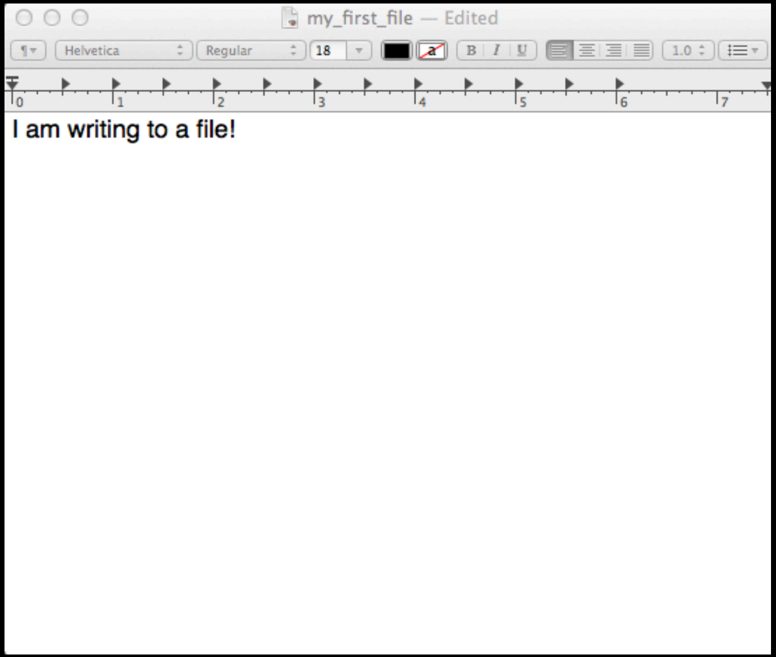

n^3
2^n
n^2
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
60
50
40
30
20
10
2
4
6
8
10

	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$nlogn$
Ω	n	n^2	n	$nlogn$
Θ		n^2		$nlogn$

File I/O

We are used to reading from and writing to the terminal:

- read from `stdin`
- write to `stdout`



I am writing to a file!

But we can also read from and write to files!

Step 1: Create a reference to the file

```
  FILE* file;
```

Step 2: Open the file

```
      file =    fopen(“file.txt”,    “r”);
```

- 1st argument -- path to the file
- 2nd argument -- mode
    - “r” -- read, “w” -- write, “a” -- append

Step 3a: Read from the file

- `fgetc` -- returns the next character
- `fgets` -- returns a line of text
- `fread` -- reads a certain # of bytes and places them into an array
- `fseek` -- moves to a certain position

Step 3b: Write to the file

- `fputc` -- write a character
- `fputs` -- returns a line of text
- `fprintf` -- print a formatted output to a file
- `fwrite` -- write an array of bytes to a file

Step 4: Close the file

```
fclose(file);
```

Remember!

- Always open a file before reading from or writing to it
- Always close a file if you open it

Example #1
Writing to a file

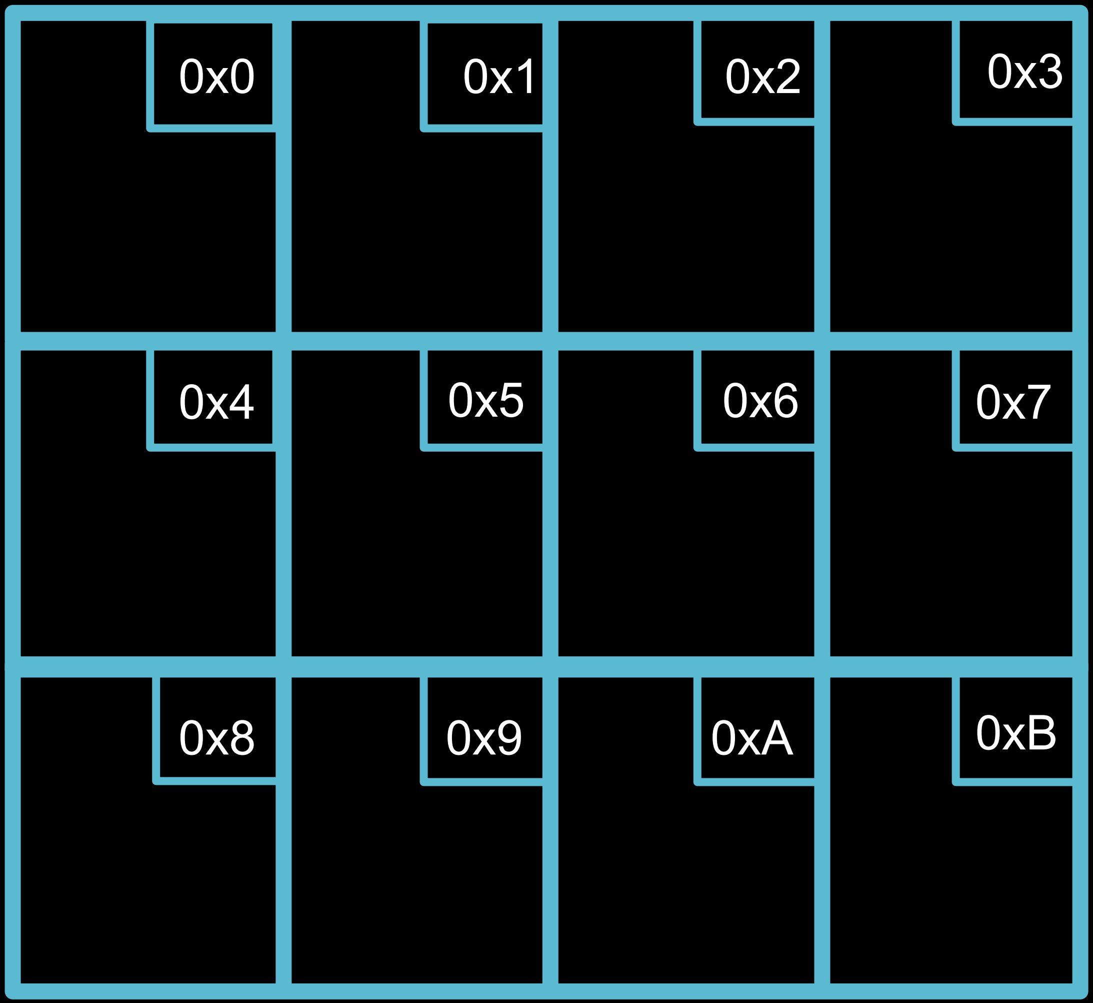
```c
#include <stdio.h>

#define STUDENTS 3

int main(void)
{
    int scores[] = { 96, 90, 83 };
    FILE* file = fopen(“database”, “w”);
    if (file != NULL)
    {
        for (int i = 0; i < STUDENTS; i++)
        {
            fprintf(file, “%i\n”, scores[i]);
        }
        fclose(file);
    }
}
```

Example #2
What does this program do?

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    if (argc < 2)
    {
        printf("Usage: cat file [file ...]\n");
        return 1;
    }
    for (int i = 1; i < argc; i++)
    {
        FILE* file = fopen(argv[i], "r");
        if (file == NULL)
        {
            printf("cat: %s: No such file or directory\n", argv[i]);
            return 1;
        }
        for (int c = fgetc(file); c != EOF; c = fgetc(file))
        {
            putchar(c);
        }
        fclose(file);
    }
    return 0;
}
```

Pointers

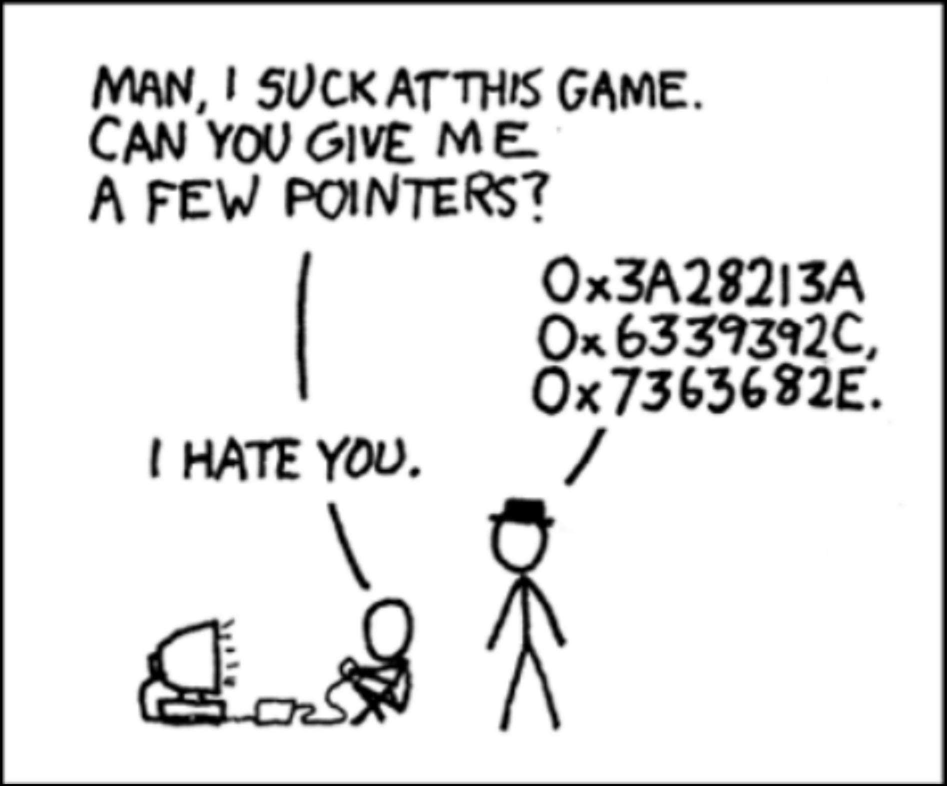
Memory



0x0 0x1 0x2 0x3

0x4 0x5 0x6 0x7

0x8 0x9 0xA 0xB


MAN, I SUCK AT THIS GAME.
CAN YOU GIVE ME
A FEW POINTERS?
0x3A28213A
0x6339392C,
0x7363682E.
I HATE YOU.

Creating Pointers

Declaring pointers:

<type>* <variable name>

Examples:

```
int* x;
char* y;
float* z;
```

Referencing and Dereferencing

Referencing:
&<variable name>

Dereferencing:
*<pointer name>

Under the hood...

```
int x = 5;

int* ptr = &x;

int copy = *ptr;
```

Variable	Address	Value
x	0x04	5
ptr	0x08	0x04
copy	0x0C	5

Track the values

	x	ptr
int x = 5;	5	
int* ptr = &x;	5	&x
*ptr = 35;	35	&x

Test Yourself

```
int a = 3, b = 4, c = 5;
int* pa = &a, *pb = &b, *pc = &c;
```

	a	b	c	pa	pb	pc
a = b * c;						
a *= c;						
b = *pa;						
pc = pa;						
*pb = b * c;						
c = (*pa) * (*pc);						
*pc = a * (*pb);						

Answers

```
int a = 3, b = 4, c = 5;
int* pa = &a, *pb = &b, *pc = &c;
```

	a	b	c	pa	pb	pc
a = b * c;	20	4	5	&a	&b	&c
a *= c;	100	4	5	&a	&b	&c
b = *pa;	100	100	5	&a	&b	&c
pc = pa;	100	100	5	&a	&b	&a
*pb = b * c;	100	500	5	&a	&b	&a
c = (*pa) * (*pc);	100	500	10000	&a	&b	&a
*pc = a * (*pb);	50000	500	10000	&a	&b	&a

Pointer Arithmetic

Adding/subtracting `n` adjusts the pointer by

`n * sizeof(<type of the pointer>)` bytes

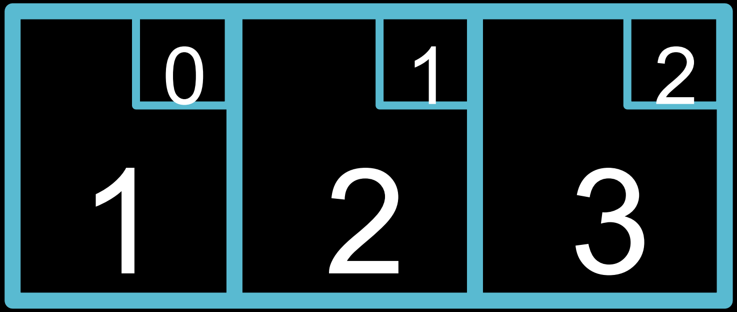
	x	y
int x = 5;	5	
int* y = &x;	5	0x04
y += 1;	5	0x08

What will print?

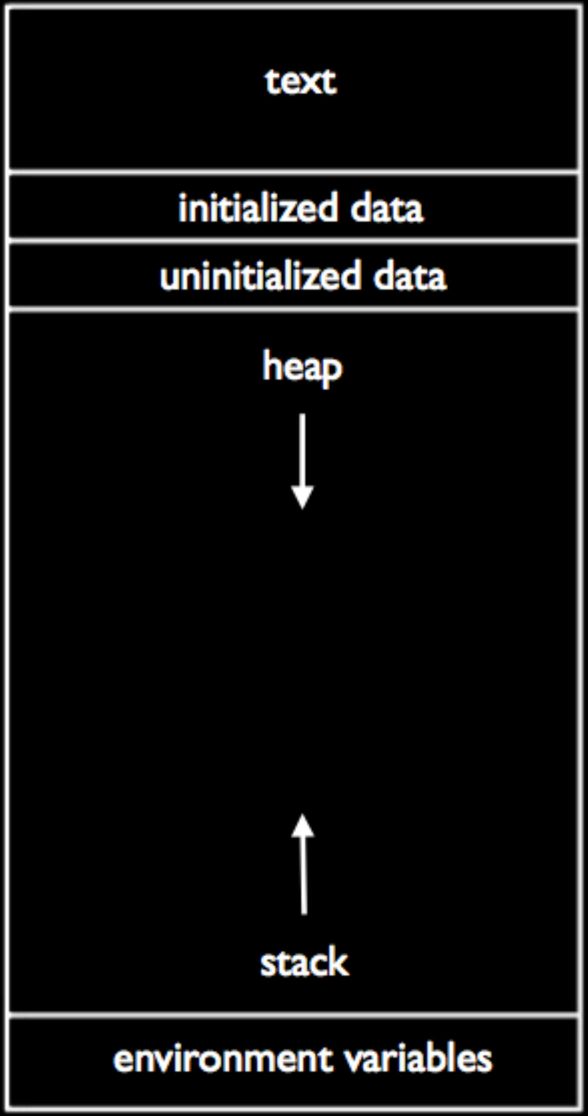
```
int main(void)
{
    char* str = "happy cat";
    int counter = 0;

    for (char* ptr = str; *ptr != '\0'; ptr++)
    {
        counter++;
    }

    printf("%d\n", counter);
}
```

Pointers and Arrays

```
int array[3];

*array = 1;
*(array + 1) = 2;
*(array + 2) = 3;
```



Dynamic Memory Allocation



A call to malloc()

prototype:

```
void* malloc(size in bytes);
```

example:

```
int* ptr = malloc(sizeof(int) * 10);
```

Check for NULL!

```
int* ptr = malloc(sizeof(int) * 10);

if (ptr == NULL)
{
        printf("Error -- out of memory.\n");
        return 1;
}
```

A call to free()

prototype:

```
void free(pointer to heap memory);
```

example:

```
free(ptr);
```

```c
#include <stdio.h>
#include <cs50.h>

int main(void)
{
    int* ptr = malloc(sizeof(int));
    if (ptr == NULL)
    {
        printf("Error -- out of memory.\n");
        return 1;
    }

    *ptr = GetInt();
    printf("You entered %d.\n", *ptr);

    free(ptr);
}
```