

Next up:

BIT MAGIC!

Next largest power of 2

```
uint32_t nlpo2(uint32_t x)
{
    x |= (x >> 1);
    x |= (x >> 2);
    x |= (x >> 4);
    x |= (x >> 8);
    x |= (x >> 16);
    return (x+1);
}
```

Next largest power of 2

000001000000000010111011001011101

000000100000000001011101100101110

000001100000000011111111101111111

000000011000000000111111111011111

000001111000000011111111111111111

000000000111100000001111111111111

000001111111100111111111111111111

000000000000011111111001111111111

000001111111111111111111111111111

...

+1

00001000000000000000000000000000

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
{
    int count = 0;

    for (int i = 0; i < 32; i++) {
        if ((x >> i) & 1) count++;
    }

    return count;
}
```

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
```

```
{
```

```
    int count = 0;
```

Loop through each bit!



```
    for (int i = 0; i < 32; i++) {
```

```
        if ((x >> i) & 1) count++;
```

```
    }
```

```
    return count;
```

```
}
```

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
```

```
{
```

```
    int count = 0;
```

```
    for (int i = 0; i < 32; i++) {
```

```
        if ((x >> i) & 1) count++;
```

```
    }
```

```
    return count;
```

```
}
```



If current bit is a ONE

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
{
    int count = 0;

    for (int i = 0; i < 32; i++) {
        if (x & 1) count++;
        x >>= 1;
    }

    return count;
}
```

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
{
    int count = 0;

    while (x) {
        if (x & 1) count++;
        x >>= 1;
    }

    return count;
}
```


Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
{
    int count = 0;

    while (x) {
        count += (x & 1);
        x >>= 1;
    }

    return count;
}
```

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
{
    int count = 0;

    while (x) {
        count += (x & 1);
        x >>= 1;
    }

    return count;
}
```

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
{
    int count = 0;

    while (x) {
        while (! (x&1)) x >>= 1;
        count += (x & 1);
        x >>= 1;
    }

    return count;
}
```

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
{
    int count = 0;

    while (x) {
        while (!(x&1)) x >>= 1;
        count++;
        x >>= 1;
    }

    return count;
}
```

Counting 1's in a 32-bit integer

```
int ones32(uint32_t x)
{
    int count = 0;
    while (x) {
        x &= x - 1;
        count++;
    }
    return count;
}
```

x = 0011001011011001

x-1 = 0011001011011000

x = 0011001011011000

x-1 = 0011001011010111

x = 0011001011010000

x-1 = 0011001011001111

0011001011000000

Counting 1's – a whole new direction!

```
int ones32(uint32_t x)
{
    x = (x&0x55555555) + ((x>>1) & 0x55555555);
    x = (x&0x33333333) + ((x>>2) & 0x33333333);
    x = (x&0x0f0f0f0f) + ((x>>4) & 0x0f0f0f0f);
    x = (x&0x00ff00ff) + ((x>>8) & 0x00ff00ff);
    x = (x&0x0000ffff) + ((x>>16) & 0x0000ffff);
    return x;
}
```

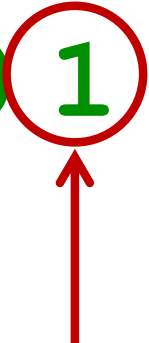
So, what's the big idea here?

0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1



How many ones are at this position?

0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1



How many ones are at this position?

All we need to do is sum the individual bits!

But why add them one at a time?

x =

0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1

0x5555 =

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

x & 0x5555 =

0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1

(x>>1) & 0x5555 =

0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0

x =

0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1

0 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1

+

0 0 0 0 0 0 0 0 0 0 0 0

0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1

x =

0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1

0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1

x =

0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1

(original)

0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1

x =

0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1
(original)

0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1

0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0

0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

Counting 1's – a whole new direction!

```
int ones32(uint32_t x)
{
    x = (x&0x55555555) + ((x>>1) & 0x55555555);
    x = (x&0x33333333) + ((x>>2) & 0x33333333);
    x = (x&0x0f0f0f0f) + ((x>>4) & 0x0f0f0f0f);
    x = (x&0x00ff00ff) + ((x>>8) & 0x00ff00ff);
    x = (x&0x0000ffff) + ((x>>16) & 0x0000ffff);
    return x;
}
```

Counting 1's – a whole new direction!

```
int ones32(uint32_t x)
{
    x -= ((x >> 1) & 0x55555555);
    x = (((x >> 2) & 0x33333333)
          + (x & 0x33333333));
    x = (((x >> 4) + x) & 0x0f0f0f0f);
    x += (x >> 8);
    x += (x >> 16);
    return (x & 0x0000003f);
}
```

Counting 1's (Still not fast enough?)

```
static const unsigned char BitsSetTable256[] = {
0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4, 1, 2, 2, 3, 2, 3, 3, 4,
2, 3, 3, 4, 3, 4, 4, 5, 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 1, 2, 2, 3, 2, 3, 3, 4,
2, 3, 3, 4, 3, 4, 4, 5, 2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 3, 4, 4, 5, 4, 5, 5, 6,
4, 5, 5, 6, 5, 6, 6, 7, 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 2, 3, 3, 4, 3, 4, 4, 5,
3, 4, 4, 5, 4, 5, 5, 6, 3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 3, 4, 4, 5, 4, 5, 5, 6,
4, 5, 5, 6, 5, 6, 6, 7, 3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
4, 5, 5, 6, 5, 6, 6, 7, 5, 6, 6, 7, 6, 7, 7, 8 };

int ones32(uint32_t x)
{
    return BitsSetTable256 [x & 0xff ]
        + BitsSetTable256 [(x >> 8) & 0xff ]
        + BitsSetTable256 [(x >> 16) & 0xff ]
        + BitsSetTable256 [x >> 24 ];
}
```


Counting 1's (take 4)

Previous techniques not fast enough for you?

```
int ones32(uint32_t x)
{
    return BitsSetTable256 [x & 0xff ]
        + BitsSetTable256 [(x >> 8) & 0xff ]
        + BitsSetTable256 [(x >> 16) & 0xff ]
        + BitsSetTable256 [x >> 24 ];
}
```

Least significant 1-bit

$$(x \ \& \ ((\sim x) + 1))$$

This can be simplified to:

$$(x \ \& \ -x)$$

Log2 of an integer

```
int floor_log2(uint32_t x)
{
    x |= (x >> 1);
    x |= (x >> 2);
    x |= (x >> 4);
    x |= (x >> 8);
    x |= (x >> 16);

    return (ones32(x >> 1));
}
```

Bit Reversal

```
uint32_t reverse(uint32_t x)
{
    x = ((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1);
    x = ((x & 0xcccccccc) >> 2) | ((x & 0x33333333) << 2);
    x = ((x & 0xf0f0f0f0) >> 4) | ((x & 0x0f0f0f0f) << 4);
    x = ((x & 0xff00ff00) >> 8) | ((x & 0x00ff00ff) << 8);

    return((x >> 16) | (x << 16));
}
```

Swap values without a temporary variable

$x = x \wedge y;$

$y = x \wedge y;$

$x = x \wedge y;$

Next up:

Identifying Available
Information
(aka Puzzles!)

Scale Problem

You have eight coins: one is a fake and heavier than the rest. How many weighings on a balance scale do you need to determine the fake?



Scale Problem

What if the fake coin may be lighter *or* heavier than the rest. How many weighings do you need to determine the fake among 12?



Hat Problem

Three people sit in a circle, and each has a colored hat placed on their head, either red or blue. Each is then asked, in turn, what color their own hat is. They may *guess* or *pass*. If they guess right the group wins, if they guess wrong the group loses, and if they pass, the next person goes. If all three pass, the group loses.

With optimal pre-planning, what is the probability of the group winning?

Hat Problem

Possibilities:

	A	B	C
1:	R	R	R
2:	R	R	B
3:	R	B	R
4:	B	R	R
5:	B	B	R
6:	B	R	B
7:	R	B	B
8:	B	B	B

Hat Problem – Part 2

Now we make the problem harder:

What if all three people must write their answer at the same time, without communication. What is the best they can do?

If all three write “pass” they fail, but they also fail if *anyone* guesses wrong.

Hat Problem

Possibilities:

	A	B	C
1:	R(B)	R(B)	R(B)
2:	R(pass)	R(pass)	B(B)
3:	R(pass)	B(B)	R(pass)
4:	B(B)	R(pass)	R(pass)
5:	B(pass)	B(pass)	R(R)
6:	B(pass)	R(R)	B(pass)
7:	R(R)	B(pass)	B(pass)
8:	B(R)	B(R)	B(R)

Lottery Tickets!

You and 99 friends of your choice are given a chance to win a large sum of money.

You are isolated and each given *identical* scratch-off lottery tickets with 100 silver spots; each spot covers one of your names. Each of you may scratch off up to 50 spots trying to find your own name. If ALL 100 of you succeed, you each get \$1 million. Otherwise no one gets anything.

What's your best strategy?