ACADEMIND

Fibonacci Sequence

Starts with 1, 1

The Fibonacci Sequence

`[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …]`

Every other element is the sum of its two preceding elements.

Popular Interview Question

Return the nth element of the Fibonacci sequence.
e.g. *`fib(4)` yields 5*

Algorithm needs to do **two things**:

(1) Calculate the sequence up to the element we're looking for

(2) Return that element

ACADEMIND

Primality Test

Determine whether an input number is a prime number

```
isPrime(9); // false
isPrime(5); // true
```

Algorithm needs to do **one thing**:

(1) Try dividing the number by all smaller numbers and return true if it's only divisible by itself and 1

ACADEMIND

Big O – Best Case, Worst Case, Average Case

For some problems (e.g. sorting array items), you have **different cases** with different time complexities

Example: “Sort the numbers in array from small to big”
Algorithm used: “Bubble Sort” *(covered later)*

Best Case: Already sorted
`nums = [1, 2, 3]`

O(n)

Avg. Case: Random order
`nums = [2, 3, 1]`

$O(n^2)$

Worst Case: Inverse order
`nums = [3, 2, 1]`

$O(n^2)$

ACADEMIND

Primality Test - Improved!

Two Examples:

5 is a prime: Divisible by itself and by 1
9 is NOT a prime: Divisible by itself, by 1 and by 3

Every number, that's NOT a prime has a product that consists of **two factors a & b** that are both **neither 1 nor the number** itself.

9 = 3 * 3

At least one factor is smaller or equal to the square root of the number.

sqrt(9) = 3 ➔ 9 = 3 * 3 ➔ Both factors are equal to the square root

ACADEMIND

Practice Time!

Write **two algorithms**:
(1) The first algorithm should take **an array of numbers** as input and **return the minimum value** in the array (i.e. the smallest number)
(2) The second (**independent**) algorithm should take **a number as input** and return **"true" if it's an even number**, "false" for odd numbers

Also define the **time complexities** and **possible cases** for both algorithms!

ACADEMIND

Is Power Of Two

Determine whether an input number is a power of two

```
isPowerOfTwo(8); // true
isPowerOfTwo(5); // false
```

Algorithm needs to do **one thing**:

(1) Divide number and future division results by two, until 1 is reached and check if the remainder is always 0

ACADEMIND

Is Power Of Two - Improved!

Bitwise Magic!

(Unsigned) Powers of two, in binary form, always have just one bit:

```
1: 1
2: 10
4: 100
```

A bitwise operation can be used to check if a number is power of two:

```
number & (number - 1)  === 0 // true: it's power of two
```

ACADEMIND

Factorial

Calculate the factorial of a number

```
fact(3); // 3 * 2 * 1 = 6
fact(5); // 5 * 4 * 3 * 2 * 1 = 120
```

Algorithm needs to do **one thing**:

(1) Go through all smaller numbers and multiply them with each other (and with the input number)

ACADEMIND

Determining the "Nature of an Algorithm"

O(n)	Higher n leads to a linear increase in runtime	Look for (single) loops
O(1)	Higher n does not affect runtime	Look for functions without loops and without any function calls
O(log n)	Runtime grows with n but at a much slower pace	Look for functions where n is split (divided) into smaller "chunks"