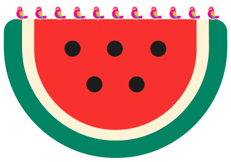
Week11a

- the Fibonacci sequence
- recursion
- dynamic programming



WARMUP

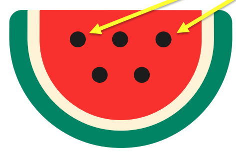
What does this print?

```
PRINT(0.1 + 0.2);
PRINT(SQRT(2.0) * SQRT(2.0));
```

WARMUP

What is F_7?

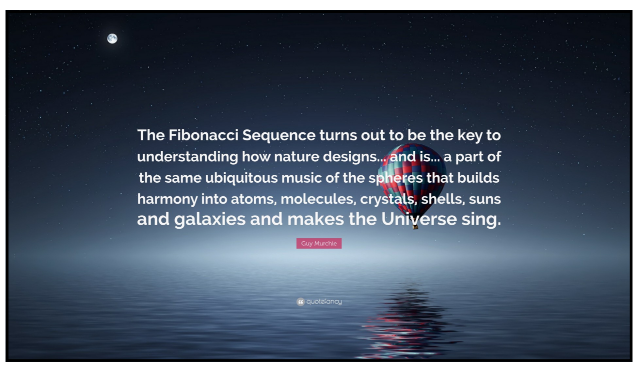
$F_0 = 0$
$F_1 = 1$
$F_k = F_{k-1} + F_{k-2}$

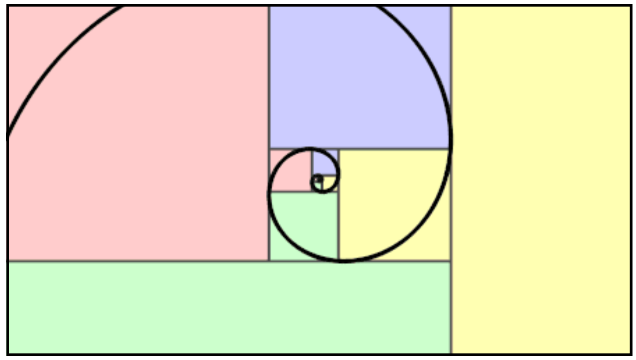


what are these?

the SEEDS OF LEARNING®

the Fibonacci sequence





```
// F_0                     =  0
// F_1                     =  1
// F_2 = F_1 + F_0 = 1 + 0 =  1
// F_3 = F_2 + F_1 = 1 + 1 =  2
// F_4 = F_3 + F_2 = 2 + 1 =  3
// F_5 = F_4 + F_3 = 3 + 2 =  5
// F_6 = F_5 + F_4 = 5 + 3 =  8
// F_7 = F_6 + F_5 = 8 + 5 = 13
// ...
```

recursion

review: recursion basics

recursion

- a **recursive function** is a function that calls itself
  - each call must make progress towards a **base case** (when the function finally returns without calling itself)
    - ✨ when in doubt, try something like zero for your base case

```
class Main {
    static int digitSum(int n) {
        if (n == 0) {
            return 0;
        }
        return digitSum(n / 10) + (n % 10);
    }

    public static void main(String[] arguments) {
        PRINT(digitSum(256)); // 13
    }
}
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 0;
return digitSum(0) + 2;
return digitSum(2) + 5;
return digitSum(25) + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 0;
return digitSum(0) + 2;
return digitSum(2) + 5;
return digitSum(25) + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 0 + 2;
return digitSum(2) + 5;
return digitSum(25) + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 2;
return digitSum(2) + 5;
return digitSum(25) + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 2;
return digitSum(2) + 5;
return digitSum(25) + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 2 + 5;
return digitSum(25) + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 7;
return digitSum(25) + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 7;
return digitSum(25) + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 7 + 6;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 13;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

```
return 13;
int a = digitSum(256);
```

```
static int digitSum(int n) {
    if (n == 0) {
        return 0;
    }
    return digitSum(n / 10) + (n % 10);
}
```

`int a = 13;`

fin.

⚠️ recursion hazard 1

repeated computation

🐌 **example:** slow very slow fibonnaci

(couldn't we just use a for loop...?)

```
// F_0 = 0
// F_1 = 1
// F_2 = F_1 + F_0 = 1 + 0 = 1
// F_3 = F_2 + F_1 = 1 + 1 = 2
// F_4 = F_3 + F_2 = 2 + 1 = 3
// F_5 = F_4 + F_3 = 3 + 2 = 5
// ...
static long fib(long k) {
    if (k == 0) { return 0; }
    if (k == 1) { return 1; }
    return fib(k - 1) + fib(k - 2);
}
```

`fib(4)`

`(fib(3) + fib(2))`

```
((fib(2) + fib(1)) + (fib(1) + fib(0)))
```

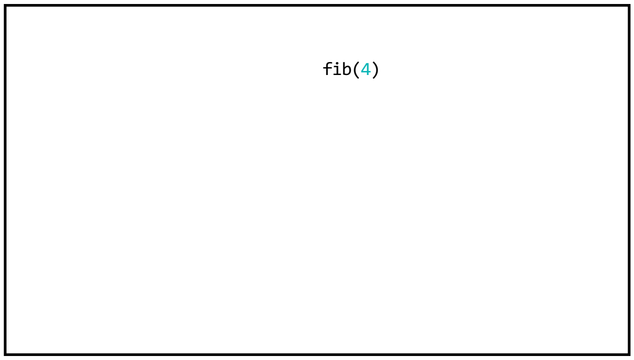
```
(((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0)))
```

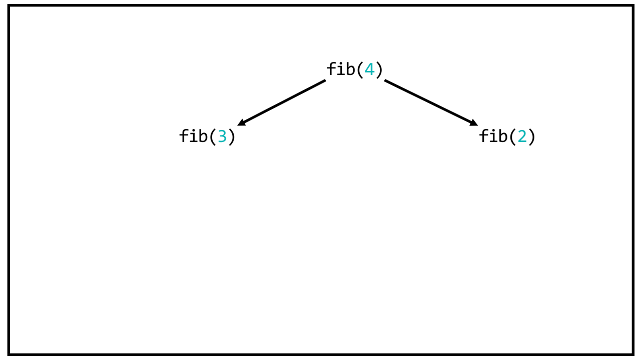
```
(((1 + 0) + 1) + (1 + 0))
```

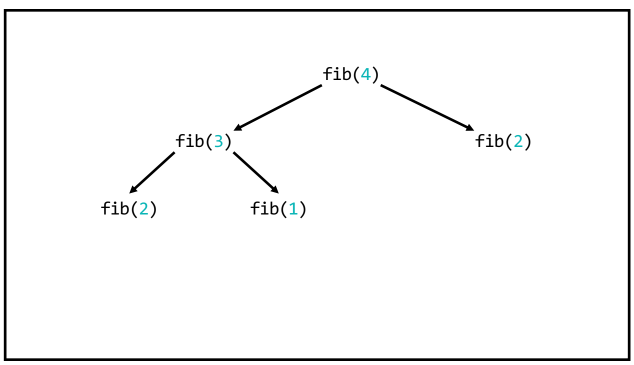
```
((1 + 1) + 1)
```

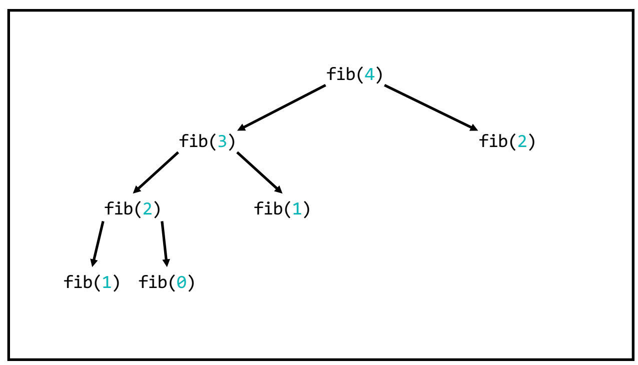
```
(2 + 1)
```

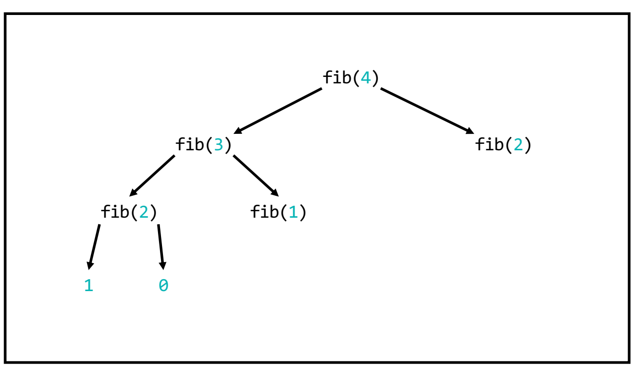
```
3
```

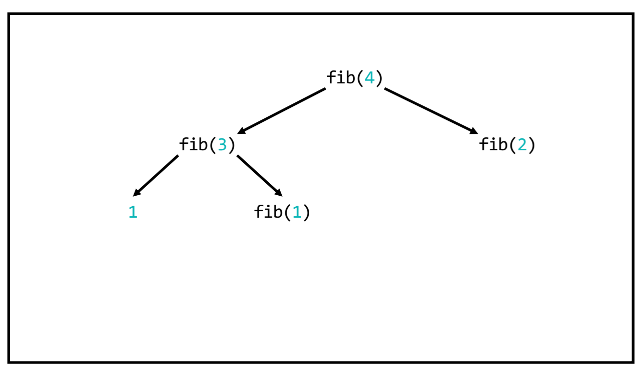

fib(4)

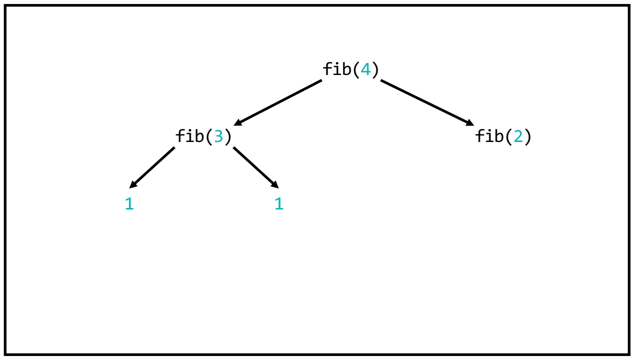

fib(4)
fib(3)
fib(2)

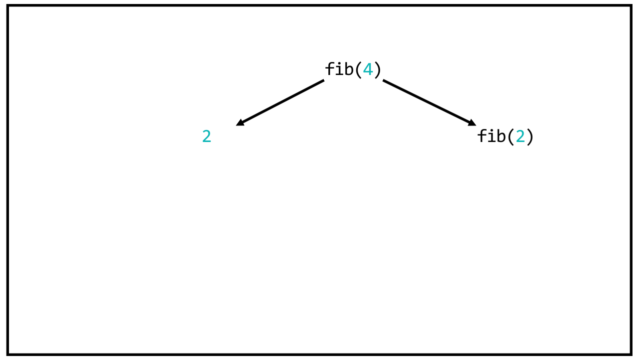

fib(4)
fib(3)
fib(2)
fib(2)
fib(1)

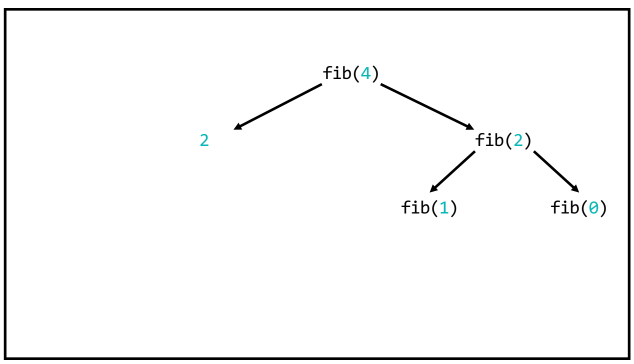

fib(4)
fib(3)
fib(2)
fib(2)
fib(1)
fib(1)
fib(0)

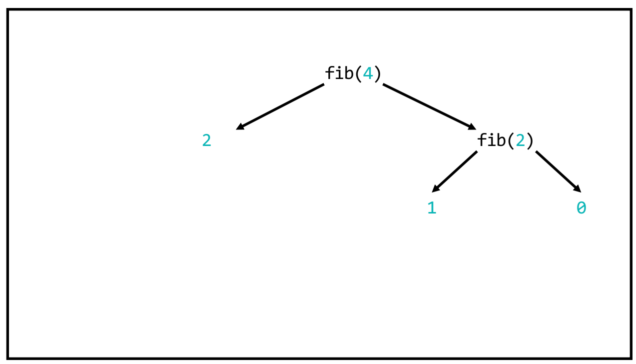

fib(4)
fib(3)
fib(2)
fib(2)
fib(1)
1
0

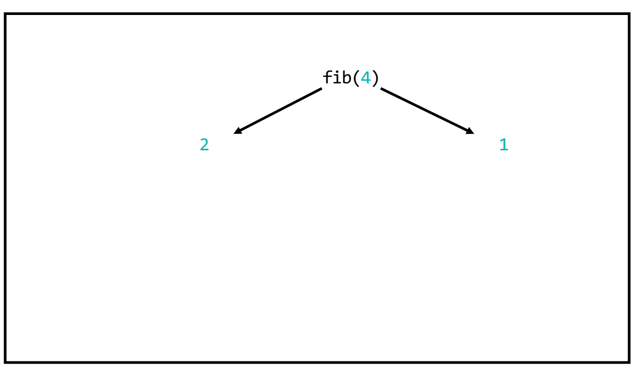

fib(4)
fib(3)
fib(2)
1
fib(1)

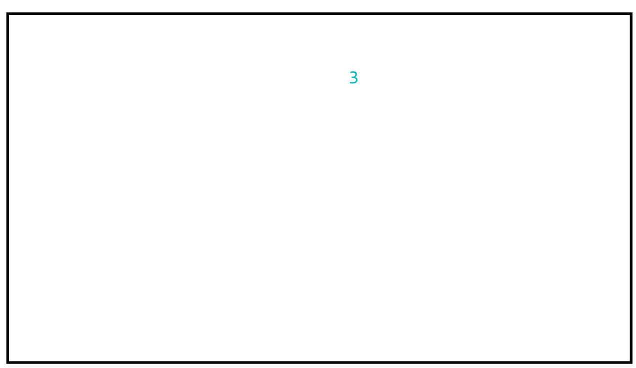

fib(4)
fib(3)
fib(2)
1
1


fib(4)
2
fib(2)


fib(4)
2
fib(2)
fib(1)
fib(0)

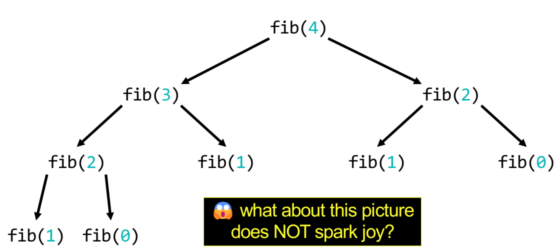

fib(4)
2
fib(2)
1
0

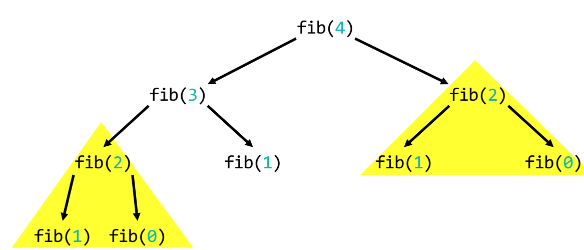

fib(4)
2
1


3

3

fib(4)


fib(4)
fib(3)
fib(2)
fib(2)
fib(1)
fib(1)
fib(0)
fib(1)
fib(0)
what about this picture
does NOT spark joy?

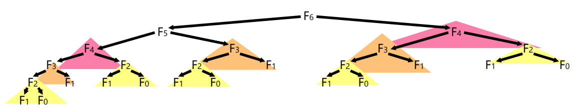

fib(4)
fib(3)
fib(2)
fib(2)
fib(1)
fib(1)
fib(0)
fib(1)
fib(0)

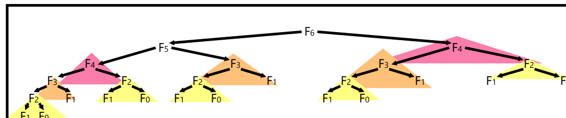
we computed `fib(2)` from scratch two seperate times

😱 we are **repeating computation**!

(what a waste 😞)

and for bigger n...
there is (much) more repetition





🧠 who's that
big O runtime?

exponential 🙀

`[fib(5), fib(36), fib(77) demo]`

⚠️ recursion hazard 2

stack overflow

☠️ 🐌 dangerous slow $\sum_{i=1}^{n} i = 1 + \cdots + n$

```
// 1 + ... + n
static int sum(int n) {
    if (n == 0) return 0;
    return n + sum(n - 1);
}
```

sum(100000)

sum(99999)

↑

sum(100000)

sum(0)

↑

⋮

↑

sum(99999)

↑

sum(100000)

😱 what about this picture does NOT spark joy?

the height of the callstack is O(n)

😱 big n → **stack overflow**!

😞

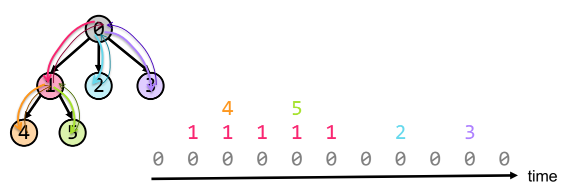
[sum(100000) demo]

🧠 our depth-first binary (search) tree traversals were recursive...

...should we be worried about them overflowing the callstack?

no. (assuming your tree is ~balanced)

callstack height for depth-first **balanced** binary tree traversal is log(n)



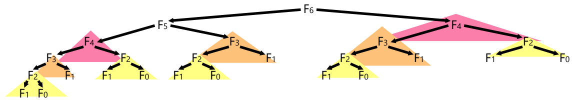
*additionally, some languages/compilers** have "tail-call optimization," which would prevent a stack overflow for `sum(100000)`

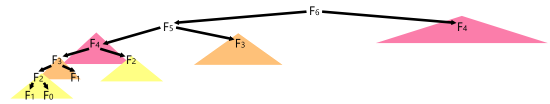
**Java is not one of these languages (as our demo showed)

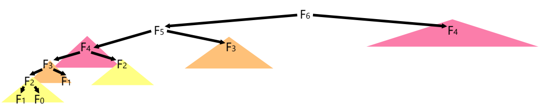
dynamic programming

dynamic programming is when you use the result of previous computation

(this is a squishy definition)







🧠 who's that big O runtime?

linear 🙄 👍

memoization

memoization means storing the results of previous functions calls, so we don't have to repeat work when the function is called again

memoized fibonacci

```java
static HashMap<Long, Long> table = new HashMap<>();

static long memoizedFib(long k) { // NOTE: long is an integer type
                                  //       that can store larger numbers than int
    if (k == 0) { return 0; }
    if (k == 1) { return 1; }

    long Fkm1;
    if (table.containsKey(k - 1)) {
        Fkm1 = table.get(k - 1);
    } else {
        Fkm1 = memoizedFib(k - 1);
        table.put(k - 1, Fkm1);
    }

    long Fkm2;
    if (table.containsKey(k - 2)) {
        Fkm2 = table.get(k - 2);
    } else {
        Fkm2 = memoizedFib(k - 2);
        table.put(k - 2, Fkm2);
    }

    return Fkm1 + Fkm2;
}
```

what's the runtime of each call to memoizedFib(...)?

```java
public static void main(...) {
    int n = ...;
    long a = memoizedFib(n);
    long b = memoizedFib(n - 1);
    long c = memoizedFib(n + 1);
}
```

```java
int n = ...;
long a = memoizedFib(n);      // O(n)
long b = memoizedFib(n - 1);  // O(1)
long c = memoizedFib(n + 1);  // O(1)
```

closed form fibonacci

Computation by rounding [edit]

Since

$$\frac{|\psi|^n}{\sqrt{5}} < \frac{1}{2}$$

for all $n \geq 0$, the number F_n is the closest integer to

$$\frac{\varphi^n}{\sqrt{5}}.$$

Therefore it can be found by rounding, or in terms of the floor function:

$$F_n = \left\lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor,\ n \geq 0.$$

Or the nearest integer function:

$$F_n = \left[\frac{\varphi^n}{\sqrt{5}} \right],\ n \geq 0.$$

Similarly, if we already know that the number $F > 1$ is a Fibonacci number, we can determine its index within the sequence by

$$n(F) = \left\lfloor \log_\varphi \left(F \cdot \sqrt{5} + \frac{1}{2} \right) \right\rfloor$$

approximate closed-form fibonnaci

```java
// NOTE: Because of floating point error, this does not work for big n.
//       (On my computer, returns wrong result for n > 70.)
static long closedFormFib(long n) {
    final double goldenRatio = (1.0 + Math.sqrt(5.0)) / 2.0;
    return Math.round(Math.pow(goldenRatio, n) / Math.sqrt(5.0));
}
```

exponentiation by squaring

note: there is actually a log(n) algorithm using matrices and "exponentiation by squaring"

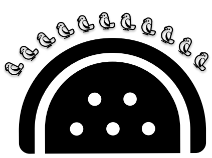
this algorithm does NOT have floating point problems (all numbers are integers)

N=1	N=2	N=4	N=8
$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix}$	$\begin{pmatrix} 34 & 21 \\ 21 & 13 \end{pmatrix}$
1	2	4	8

Times matrix is multiplied with itself

Week11b

- Fibonacci wrapup
- recursion example



WARMUP

Say n = 64

Your goal is to compute 3^n ("3 to the n-th power") using only the multiplication operator (*)

Give an O(n) algorithm

Give an O(log n) algorithm

---

recursive fibonacci wrapup

---

review: bad bad very bad recursive O(2^n) fibonacci

🐌 recursive O(2^n) fibonacci

```
static long fib(long k) {
    if (k == 0) { return 0; }
    if (k == 1) { return 1; }
    return fib(k - 1) + fib(k - 2);
}
```

review:
recursive memoized
O(n) fibonacci

memoized recursive O(n) fibonacci

```
// NOTE: Could also have used an array, with, for example, value 0 meaning "not yet computed."
static HashMap<Integer, Long> table = new HashMap<>();
static long fib(int k) {
  if (k == 0) { return 0; }
  if (k == 1) { return 1; }

  long Fkm1;
  if (table.containsKey(k - 1)) {
    Fkm1 = table.get(k - 1);
  } else {
    Fkm1 = fib(k - 1);
    table.put(k - 1, Fkm1);
  }

  long Fkm2;
  if (table.containsKey(k - 2)) {
    Fkm2 = table.get(k - 2);
  } else {
    Fkm2 = fib(k - 2);
    table.put(k - 2, Fkm2);
  }

  return Fkm1 + Fkm2;
}
```

can we get the best of both worlds?

(easy to read, fast)

[add a helper function]

recursive O(n) fibonnaci

```
static HashMap<Integer, Long> table = new HashMap<>();

static long fibHelper(int k) {
  long result;
  if (table.containsKey(k)) {
    result = table.get(k);
  } else {
    result = fib(k);
    table.put(k, result);
  }
  return result;
}


static long fib(int k) {
  if (k == 0) { return 0; }
  if (k == 1) { return 1; }
  return fibHelper(k - 1) + fibHelper(k - 2);
}
```

alternate fibonacci approaches

iterative (not-recursive) O(n) fibonacci

iterative O(n) fibonnaci

```java
static long fib(int n) {
  long F_i   = 1;
  long F_im1 = 0;
  for (int i = 2; i <= n + 1; ++i) {
    // (F_i, F_im1) <- (F_i + F_im1, F_i)
    long tmp = F_i;
    F_i += F_im1;
    F_im1 = tmp;
  }
  return F_im1;
}
```

TODO make better

```python
def fib(n):
    F_i = 1
    F_im1 = 0
    for i in range(2, n + 2):
        F_i, F_im1 = (F_i + F_im1), F_i
    return F_im1
```

$(F_1, F_0) = (1, 0)$

$(F_2, F_1) \leftarrow (1 + 0, 1) = (1, 1)$

$(F_3, F_2) \leftarrow (1 + 1, 1) = (2, 1)$

$(F_4, F_3) \leftarrow (2 + 1, 2) = (3, 2)$

$(F_5, F_4) \leftarrow (3 + 2, 3) = (5, 3)$

closed form fibonacci

Computation by rounding [edit]

Since

$$\frac{|\psi|^n}{\sqrt{5}} < \frac{1}{2}$$

for all $n \geq 0$, the number F_n is the closest integer to

$$\frac{\varphi^n}{\sqrt{5}}.$$

Therefore it can be found by rounding, or in terms of the floor function:

$$F_n = \left\lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor, \ n \geq 0.$$

Or the nearest integer function:

$$F_n = \left[\frac{\varphi^n}{\sqrt{5}} \right], \ n \geq 0.$$

Similarly, if we already know that the number $F > 1$ is a Fibonacci number, we can determine its index within the sequence by

$$n(F) = \left\lfloor \log_\varphi \left(F \cdot \sqrt{5} + \frac{1}{2} \right) \right\rfloor$$

approximate closed-form fibonacci

```java
// NOTE: Because of floating point error, this does not work for big n.
//       (On my computer, returns wrong result for n > 70.)
static long closedFormFib(long n) {
    final double goldenRatio = (1.0 + Math.sqrt(5.0)) / 2.0;
    return Math.round(Math.pow(goldenRatio, n) / Math.sqrt(5.0));
}
```

O(log n)
matrix exponentiation by squaring (oh no)

problem: calculate fib(64)
observation: 64 = 2 * 2 * 2 * 2 * 2 * 2

[matrix multiplication review]

[update rule as a matrix multiplication]

[matrix exponentiation by squaring]

recursion example

subset sum

problem overview

given a finite set of numbers $\{a, b, c, \ldots\}$,
is there **any** subset that sums to target T?

no. --Mark

examples

- is there any subset of { 1, 3, 5 } that sums to 4?
  - yes; { 1, 3 }
- is there any subset of { 1, 3, 5 } that sums to 9?
  - yes; { 1, 3, 5 }
- is there any subset of { 1, 3, 5 } that sums to 0?
  - yes; {}
- is there any subset of { 1, 3, 5 } that sums to 7?
  - no

solution method

given a finite set of numbers $\{a, b, c, \ldots\}$,
is there **any** subset that sums to target T?

any

any

- **for each subset...**
  - if it sums to target...
    - `return true;` 😀
- `return false;`

- **for each subset...**
  - if it sums to target...
    - `return true;` 😀
- `return false;`

🧠 why is this maybe not so easy?

"hard" means "hard for a computer to solve quickly"

---

what are the subsets of $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$?

∅ | {1} | {2} | {3} | {4} | {5} | {6} | {7} | {8} | {9} | {1, 2} | {1, 3} | {1, 4} | {1, 5} | {1, 6} | {1, 7} | {1, 8} | {1, 9} | {2, 3} | {2, 4} | {2, 5} | {2, 6} | {2, 7} | {2, 8} | {2, 9} | {3, 4} | {3, 5} | {3, 6} | {3, 7} | {3, 8} | {3, 9} | {4, 5} | {4, 6} | {4, 7} | {4, 8} | {4, 9} | {5, 6} | {5, 7} | {5, 8} | {5, 9} | {6, 7} | {6, 8} | {6, 9} | {7, 8} | {7, 9} | {8, 9} | {1, 2, 3} | {1, 2, 4} | {1, 2, 5} | {1, 2, 6} | {1, 2, 7} | {1, 2, 8} | {1, 2, 9} | {1, 3, 4} | {1, 3, 5} | {1, 3, 6} | {1, 3, 7} | {1, 3, 8} | {1, 3, 9} | {1, 4, 5} | {1, 4, 6} | {1, 4, 7} | {1, 4, 8} | {1, 4, 9} | {1, 5, 6} | {1, 5, 7} | {1, 5, 8} | {1, 5, 9} | {1, 6, 7} | {1, 6, 8} | {1, 6, 9} | {1, 7, 8} | {1, 7, 9} | {1, 8, 9} | {2, 3, 4} | {2, 3, 5} | {2, 3, 6} | {2, 3, 7} | {2, 3, 8} | {2, 3, 9} | {2, 4, 5} | {2, 4, 6} | {2, 4, 7} | {2, 4, 8} | {2, 4, 9} | {2, 5, 6} | {2, 5, 7} | {2, 5, 8} | {2, 5, 9} | {2, 6, 7} | {2, 6, 8} | {2, 6, 9} | {2, 7, 8} | {2, 7, 9} | {2, 8, 9} | {3, 4, 5} | {3, 4, 6} | {3, 4, 7} | {3, 4, 8} | {3, 4, 9} | {3, 5, 6} | {3, 5, 7} | {3, 5, 8} | {3, 5, 9} | {3, 6, 7} | {3, 6, 8} | {3, 6, 9} | {3, 7, 8} | {3, 7, 9} | {3, 8, 9} | {4, 5, 6} | {4, 5, 7} | {4, 5, 8} | {4, 5, 9} | {4, 6, 7} | {4, 6, 8} | {4, 6, 9} | {4, 7, 8} | {4, 7, 9} | {4, 8, 9} | {5, 6, 7} | {5, 6, 8} | {5, 6, 9} | {5, 7, 8} | {5, 7, 9} | {5, 8, 9} | {6, 7, 8} | {6, 7, 9} | {6, 8, 9} | {7, 8, 9} | {1, 2, 3, 4} | {1, 2, 3, 5} | {1, 2, 3, 6} | {1, 2, 3, 7} | {1, 2, 3, 8} | {1, 2, 3, 9} | {1, 2, 4, 5} | {1, 2, 4, 6} | {1, 2, 4, 7} | {1, 2, 4, 8} | {1, 2, 4, 9} | {1, 2, 5, 6} | {1, 2, 5, 7} | {1, 2, 5, 8} | {1, 2, 5, 9} | {1, 2, 6, 7} | {1, 2, 6, 8} | {1, 2, 6, 9} | {1, 2, 7, 8} | {1, 2, 7, 9} | {1, 2, 8, 9} | {1, 3, 4, 5} | {1, 3, 4, 6} | {1, 3, 4, 7} | {1, 3, 4, 8} | {1, 3, 4, 9} | {1, 3, 5, 6} | {1, 3, 5, 7} | {1, 3, 5, 8} | {1, 3, 5, 9} | {1, 3, 6, 7} | {1, 3, 6, 8} | {1, 3, 6, 9} | {1, 3, 7, 8} | {1, 3, 7, 9} | {1, 3, 8, 9} | {1, 4, 5, 6} | {1, 4, 5, 7} | {1, 4, 5, 8} | {1, 4, 5, 9} | {1, 4, 6, 7} | {1, 4, 6, 8} | {1, 4, 6, 9} | {1, 4, 7, 8} | {1, 4, 7, 9} | {1, 4, 8, 9} | {1, 5, 6, 7} | {1, 5, 6, 8} | {1, 5, 6, 9} | {1, 5, 7, 8} | {1, 5, 7, 9} | {1, 5, 8, 9} | {1, 6, 7, 8} | {1, 6, 7, 9} | {1, 6, 8, 9} | {1, 7, 8, 9} | {2, 3, 4, 5} | {2, 3, 4, 6} | {2, 3, 4, 7} | {2, 3, 4, 8} | {2, 3, 4, 9} | {2, 3, 5, 6} | {2, 3, 5, 7} | {2, 3, 5, 8} | {2, 3, 5, 9} | {2, 3, 6, 7} | {2, 3, 6, 8} | {2, 3, 6, 9} | {2, 3, 7, 8} | {2, 3, 7, 9} | {2, 3, 8, 9} | {2, 4, 5, 6} | {2, 4, 5, 7} | {2, 4, 5, 8} | {2, 4, 5, 9} | {2, 4, 6, 7} | {2, 4, 6, 8} | {2, 4, 6, 9} | {2, 4, 7, 8} | {2, 4, 7, 9} | {2, 4, 8, 9} | {2, 5, 6, 7} | {2, 5, 6, 8} | {2, 5, 6, 9} | {2, 5, 7, 8} | {2, 5, 7, 9} | {2, 5, 8, 9} | {2, 6, 7, 8} | {2, 6, 7, 9} | {2, 6, 8, 9} | {2, 7, 8, 9} | {3, 4, 5, 6} | {3, 4, 5, 7} | {3, 4, 5, 8} | {3, 4, 5, 9} | {3, 4, 6, 7} | {3, 4, 6, 8} | {3, 4, 6, 9} | {3, 4, 7, 8} | {3, 4, 7, 9} | {3, 4, 8, 9} | {3, 5, 6, 7} | {3, 5, 6, 8} | {3, 5, 6, 9} | {3, 5, 7, 8} | {3, 5, 7, 9} | {3, 5, 8, 9} | {3, 6, 7, 8} | {3, 6, 7, 9} | {3, 6, 8, 9} | {3, 7, 8, 9} | {4, 5, 6, 7} | {4, 5, 6, 8} | {4, 5, 6, 9} | {4, 5, 7, 8} | {4, 5, 7, 9} | {4, 5, 8, 9} | {4, 6, 7, 8} | {4, 6, 7, 9} | {4, 6, 8, 9} | {4, 7, 8, 9} | {5, 6, 7, 8} | {5, 6, 7, 9} | {5, 6, 8, 9} | {5, 7, 8, 9} | {6, 7, 8, 9} | {1, 2, 3, 4, 5} | {1, 2, 3, 4, 6} | {1, 2, 3, 4, 7} | {1, 2, 3, 4, 8} | {1, 2, 3, 4, 9} | {1, 2, 3, 5, 6} | {1, 2, 3, 5, 7} | {1, 2, 3, 5, 8} | {1, 2, 3, 5, 9} | {1, 2, 3, 6, 7} | {1, 2, 3, 6, 8} | {1, 2, 3, 6, 9} | {1, 2, 3, 7, 8} | {1, 2, 3, 7, 9} | {1, 2, 3, 8, 9} | {1, 2, 4, 5, 6} | {1, 2, 4, 5, 7} | {1, 2, 4, 5, 8} | {1, 2, 4, 5, 9} | {1, 2, 4, 6, 7} | {1, 2, 4, 6, 8} | {1, 2, 4, 6, 9} | {1, 2, 4, 7, 8} | {1, 2, 4, 7, 9} | {1, 2, 4, 8, 9} | {1, 2, 5, 6, 7} | {1, 2, 5, 6, 8} | {1, 2, 5, 6, 9} | {1, 2, 5, 7, 8} | {1, 2, 5, 7, 9} | {1, 2, 5, 8, 9} | {1, 2, 6, 7, 8} | {1, 2, 6, 7, 9} | {1, 2, 6, 8, 9} | {1, 2, 7, 8, 9} | {1, 3, 4, 5, 6} | {1, 3, 4, 5, 7} | {1, 3, 4, 5, 8} | {1, 3, 4, 5, 9} | {1, 3, 4, 6, 7} | {1, 3, 4, 6, 8} | {1, 3, 4, 6, 9} | {1, 3, 4, 7, 8} | {1, 3, 4, 7, 9} | {1, 3, 4, 8, 9} | {1, 3, 5, 6, 7} | {1, 3, 5, 6, 8} | {1, 3, 5, 6, 9} | {1, 3, 5, 7, 8} | {1, 3, 5, 7, 9} | {1, 3, 5, 8, 9} | {1, 3, 6, 7, 8} | {1, 3, 6, 7, 9} | {1, 3, 6, 8, 9} | {1, 3, 7, 8, 9} | {1, 4, 5, 6, 7} | {1, 4, 5, 6, 8} | {1, 4, 5, 6, 9} | {1, 4, 5, 7, 8} | {1, 4, 5, 7, 9} | {1, 4, 5, 8, 9} | {1, 4, 6, 7, 8} | {1, 4, 6, 7, 9} | {1, 4, 6, 8, 9} | {1, 4, 7, 8, 9} | {1, 5, 6, 7, 8} | {1, 5, 6, 7, 9} | {1, 5, 6, 8, 9} | {1, 5, 7, 8, 9} | {1, 6, 7, 8, 9} | {2, 3, 4, 5, 6} | {2, 3, 4, 5, 7} | {2, 3, 4, 5, 8} | {2, 3, 4, 5, 9} | {2, 3, 4, 6, 7} | {2, 3, 4, 6, 8} | {2, 3, 4, 6, 9} | {2, 3, 4, 7, 8} | {2, 3, 4, 7, 9} | {2, 3, 4, 8, 9} | {2, 3, 5, 6, 7} | {2, 3, 5, 6, 8} | {2, 3, 5, 6, 9} | {2, 3, 5, 7, 8} | {2, 3, 5, 7, 9} | {2, 3, 5, 8, 9} | {2, 3, 6, 7, 8} | {2, 3, 6, 7, 9} | {2, 3, 6, 8, 9} | {2, 3, 7, 8, 9} | {2, 4, 5, 6, 7} | {2, 4, 5, 6, 8} | {2, 4, 5, 6, 9} | {2, 4, 5, 7, 8} | {2, 4, 5, 7, 9} | {2, 4, 5, 8, 9} | {2, 4, 6, 7, 8} | {2, 4, 6, 7, 9} | {2, 4, 6, 8, 9} | {2, 4, 7, 8, 9} | {2, 5, 6, 7, 8} | {2, 5, 6, 7, 9} | {2, 5, 6, 8, 9} | {2, 5, 7, 8, 9} | {2, 6, 7, 8, 9} | {3, 4, 5, 6, 7} | {3, 4, 5, 6, 8} | {3, 4, 5, 6, 9} | {3, 4, 5, 7, 8} | {3, 4, 5, 7, 9} | {3, 4, 5, 8, 9} | {3, 4, 6, 7, 8} | {3, 4, 6, 7, 9} | {3, 4, 6, 8, 9} | {3, 4, 7, 8, 9} | {3, 5, 6, 7, 8} | {3, 5, 6, 7, 9} | {3, 5, 6, 8, 9} | {3, 5, 7, 8, 9} | {3, 6, 7, 8, 9} | {4, 5, 6, 7, 8} | {4, 5, 6, 7, 9} | {4, 5, 6, 8, 9} | {4, 5, 7, 8, 9} | {4, 6, 7, 8, 9} | {5, 6, 7, 8, 9} | {1, 2, 3, 4, 5, 6} | {1, 2, 3, 4, 5, 7} | {1, 2, 3, 4, 5, 8} | {1, 2, 3, 4, 5, 9} | {1, 2, 3, 4, 6, 7} | {1, 2, 3, 4, 6, 8} | {1, 2, 3, 4, 6, 9} | {1, 2, 3, 4, 7, 8} | {1, 2, 3, 4, 7, 9} | {1, 2, 3, 4, 8, 9} | {1, 2, 3, 5, 6, 7} | {1, 2, 3, 5, 6, 8} | {1, 2, 3, 5, 6, 9} | {1, 2, 3, 5, 7, 8} | {1, 2, 3, 5, 7, 9} | {1, 2, 3, 5, 8, 9} | {1, 2, 3, 6, 7, 8} | {1, 2, 3, 6, 7, 9} | {1, 2, 3, 6, 8, 9} | {1, 2, 3, 7, 8, 9} | {1, 2, 4, 5, 6, 7} | {1, 2, 4, 5, 6, 8} | {1, 2, 4, 5, 6, 9} | {1, 2, 4, 5, 7, 8} | {1, 2, 4, 5, 7, 9} | {1, 2, 4, 5, 8, 9} | {1, 2, 4, 6, 7, 8} | {1, 2, 4, 6, 7, 9} | {1, 2, 4, 6, 8, 9} | {1, 2, 4, 7, 8, 9} | {1, 2, 5, 6, 7, 8} | {1, 2, 5, 6, 7, 9} | {1, 2, 5, 6, 8, 9} | {1, 2, 5, 7, 8, 9} | {1, 2, 6, 7, 8, 9} | {1, 3, 4, 5, 6, 7} | {1, 3, 4, 5, 6, 8} | {1, 3, 4, 5, 6, 9} | {1, 3, 4, 5, 7, 8} | {1, 3, 4, 5, 7, 9} | {1, 3, 4, 5, 8, 9} | {1, 3, 4, 6, 7, 8} | {1, 3, 4, 6, 7, 9} | {1, 3, 4, 6, 8, 9} | {1, 3, 4, 7, 8, 9} | {1, 3, 5, 6, 7, 8} | {1, 3, 5, 6, 7, 9} | {1, 3, 5, 6, 8, 9} | {1, 3, 5, 7, 8, 9} | {1, 3, 6, 7, 8, 9} | {1, 4, 5, 6, 7, 8} | {1, 4, 5, 6, 7, 9} | {1, 4, 5, 6, 8, 9} | {1, 4, 5, 7, 8, 9} | {1, 4, 6, 7, 8, 9} | {1, 5, 6, 7, 8, 9} | {2, 3, 4, 5, 6, 7} | {2, 3, 4, 5, 6, 8} | {2, 3, 4, 5, 6, 9} | {2, 3, 4, 5, 7, 8} | {2, 3, 4, 5, 7, 9} | {2, 3, 4, 5, 8, 9} | {2, 3, 4, 6, 7, 8} | {2, 3, 4, 6, 7, 9} | {2, 3, 4, 6, 8, 9} | {2, 3, 4, 7, 8, 9} | {2, 3, 5, 6, 7, 8} | {2, 3, 5, 6, 7, 9} | {2, 3, 5, 6, 8, 9} | {2, 3, 5, 7, 8, 9} | {2, 3, 6, 7, 8, 9} | {2, 4, 5, 6, 7, 8} | {2, 4, 5, 6, 7, 9} | {2, 4, 5, 6, 8, 9} | {2, 4, 5, 7, 8, 9} | {2, 4, 6, 7, 8, 9} | {2, 5, 6, 7, 8, 9} | {3, 4, 5, 6, 7, 8} | {3, 4, 5, 6, 7, 9} | {3, 4, 5, 6, 8, 9} | {3, 4, 5, 7, 8, 9} | {3, 4, 6, 7, 8, 9} | {3, 5, 6, 7, 8, 9} | {4, 5, 6, 7, 8, 9} | {1, 2, 3, 4, 5, 6, 7} | {1, 2, 3, 4, 5, 6, 8} | {1, 2, 3, 4, 5, 6, 9} | {1, 2, 3, 4, 5, 7, 8} | {1, 2, 3, 4, 5, 7, 9} | {1, 2, 3, 4, 5, 8, 9} | {1, 2, 3, 4, 6, 7, 8} | {1, 2, 3, 4, 6, 7, 9} | {1, 2, 3, 4, 6, 8, 9} | {1, 2, 3, 4, 7, 8, 9} | {1, 2, 3, 5, 6, 7, 8} | {1, 2, 3, 5, 6, 7, 9} | {1, 2, 3, 5, 6, 8, 9} | {1, 2, 3, 5, 7, 8, 9} | {1, 2, 3, 6, 7, 8, 9} | {1, 2, 4, 5, 6, 7, 8} | {1, 2, 4, 5, 6, 7, 9} | {1, 2, 4, 5, 6, 8, 9} | {1, 2, 4, 5, 7, 8, 9} | {1, 2, 4, 6, 7, 8, 9} | {1, 2, 5, 6, 7, 8, 9} | {1, 3, 4, 5, 6, 7, 8} | {1, 3, 4, 5, 6, 7, 9} | {1, 3, 4, 5, 6, 8, 9} | {1, 3, 4, 5, 7, 8, 9} | {1, 3, 4, 6, 7, 8, 9} | {1, 3, 5, 6, 7, 8, 9} | {1, 4, 5, 6, 7, 8, 9} | {2, 3, 4, 5, 6, 7, 8} | {2, 3, 4, 5, 6, 7, 9} | {2, 3, 4, 5, 6, 8, 9} | {2, 3, 4, 5, 7, 8, 9} | {2, 3, 4, 6, 7, 8, 9} | {2, 3, 5, 6, 7, 8, 9} | {2, 4, 5, 6, 7, 8, 9} | {3, 4, 5, 6, 7, 8, 9} | {1, 2, 3, 4, 5, 6, 7, 8} | {1, 2, 3, 4, 5, 6, 7, 9} | {1, 2, 3, 4, 5, 6, 8, 9} | {1, 2, 3, 4, 5, 7, 8, 9} | {1, 2, 3, 4, 6, 7, 8, 9} | {1, 2, 3, 5, 6, 7, 8, 9} | {1, 2, 4, 5, 6, 7, 8, 9} | {1, 3, 4, 5, 6, 7, 8, 9} | {2, 3, 4, 5, 6, 7, 8, 9} | {1, 2, 3, 4, 5, 6, 7, 8, 9}

---

there are 2^9 of them

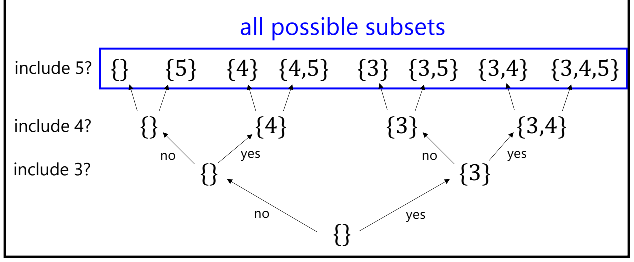
🧠 why?

each of the 9 elements is either included or not included (excluded) in the subset

9 include/exclude decisions => 2^9

what are the subsets of $\{3, 4, 5\}$?

what are the subsets of $\{3, 4, 5\}$?



hint

Question: "can any subset of $\{a, b, c, \ldots\}$ sum to T?"

key insight

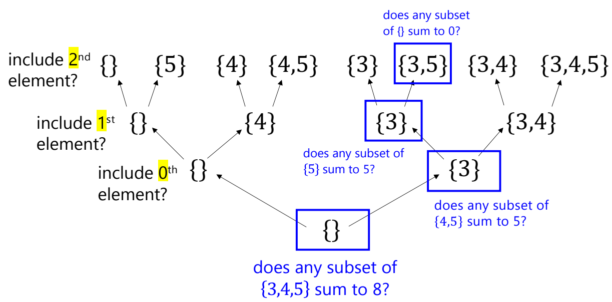
Considering just a, we have **two cases:**

1) exclude a

in this case, **Equivalent Question:** "can any subset of $\{b, c, \ldots\}$ sum to T?"

2) include a

in this case, **Equivalent Question:** "can any subset of $\{b, c, \ldots\}$ sum to $T - a$?"

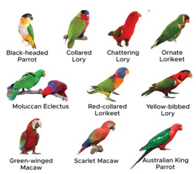


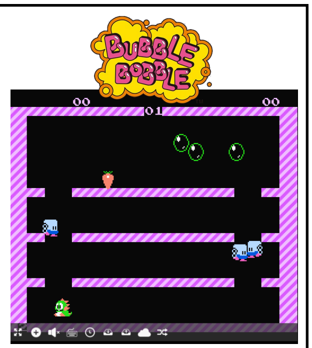
okay cool good luck

Week11c

Final Project

WARMUP: What part of this game would you implement... first? last?





final project

final project

You may **do your final project on whatever you like**, provided you can **answer the following questions**.

1. What is the **title** of my project?
2. What **data structures** will I use?
   Note: Arrays count.
3. **What** is the game/app that I am proposing?
   What does it do?
   How does it *feel*?
4. Will the viewer/player **interact** with my project?
   How so?
5. Does Jim think my project is **doable**?
   What is my fallback plan if my project ends up being harder than I expect?
   What extensions can I do if my project ends up being easier than I expect?
6. What is the very **first thing I will implement**?
   (Drawing "the data" is usually a good first step.)

do your final project on whatever you like
answer the following questions

1. **title**
2. **data structures**
3. **What**
4. **interact**
5. **doable?**
6. **first thing I will implement**

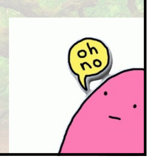
example

- Woo!-doku
- 2D array to represent the board.
- A colorful sudoku board, that does a happy dance when you solve it.
- Click to select cells. Type numbers on the keyboard to fill in numbers.
- Yes! And you can write a sudoku solver or automatic board generation if you have extra time!
- Store a board I found on the internet as a 2D array (0's for empty cells) and draw it to the Terminal using print statements (or...just skip straight to Cow)

how are we feeling?

why am i making you make a thing

The only way you're going to grow is by pushing yourself beyond what you think is possible.
--David Goggins



how to
make a thing

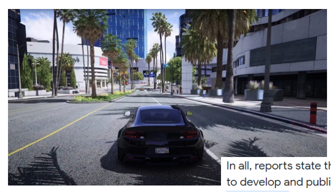
how not to make a thing

note: this advice is like...just advice

feel free to ignore (at your own peril) 😜

project selection

⚠️ picking something really, really hard

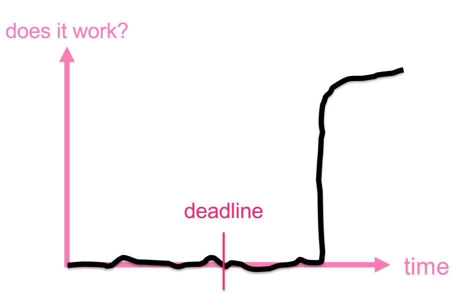


In all, reports state that Grand Theft Auto 5 cost a whopping $265 million to develop and publish. This includes the game's core budget of around **$140 million**, plus some staggering marketing costs befitting of such a release. Nov 9, 2023

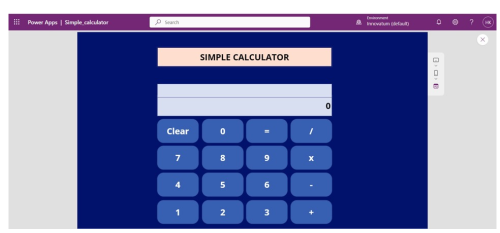
Vintage is The New Old
https://www.vintageisthenewold.com › game-pedia › wh...

What was GTA 5 original budget? - Vintage is The New Old

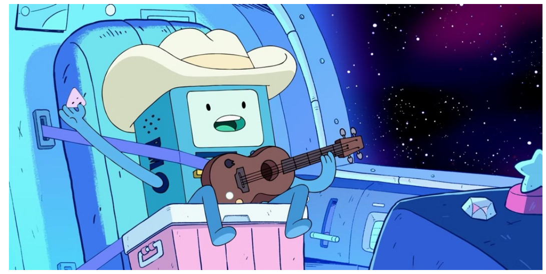
⚠️ picking something with a spooky progress curve



⚠️ picking something you aren't really interested in
(or can't really be **extended**)



note: though if you love calculators, go for it!



who (doesn't) have a final project idea?

discuss amongst each other

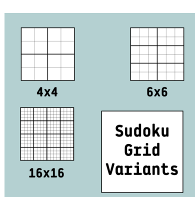
focus

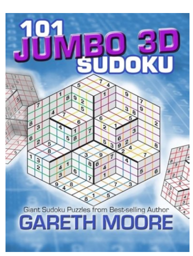
(just make sudoku)

⚠️ solving a more general problem than you need to

don't support arbitrary board sizes!

just make sudoku!



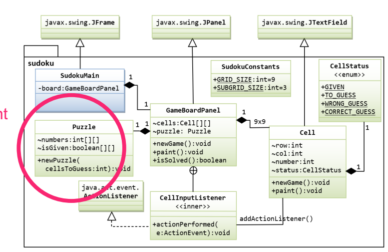


what?!

⚠️ writing more general code than you need to

don't support multiple different kinds of puzzles

just make sudoku!



⚠️ doing the fun part first

don't make a fun animated menu screen

just make sudoku!

⚠️ skipping development steps

```
-------------------------
| 0 1 2 | 3 4 5 | 6 7 8 |
| 0 1 2 | 3 4 5 | 6 7 8 |
| 0 1 2 | 3 4 5 | 6 7 8 |
-------------------------
| 0 1 2 | 3 4 5 | 6 7 8 |
| 0 1 2 | 3 4 5 | 6 7 8 |
| 0 1 2 | 3 4 5 | 6 7 8 |
-------------------------
| 0 1 2 | 3 4 5 | 6 7 8 |
| 0 1 2 | 3 4 5 | 6 7 8 |
| 0 1 2 | 3 4 5 | 6 7 8 |
-------------------------
```

consider making a board print to the terminal before trying to get it working in Cow.java

consider just filling it with nonsense before filling it with a real board

consider hardcoding a starting board by hand before implementing automatic board generation

then again, sometimes you just gotta go for it.

YOLO!

who (doesn't) know what they're going to implement first?

discuss amongst each other

final thoughts

final thoughts

- don't be afraid to write code
- don't be afraid to delete code
- don't be (too) afraid to fail
- you will be graded primarily on effort

how are we feeling?