Winter 2015

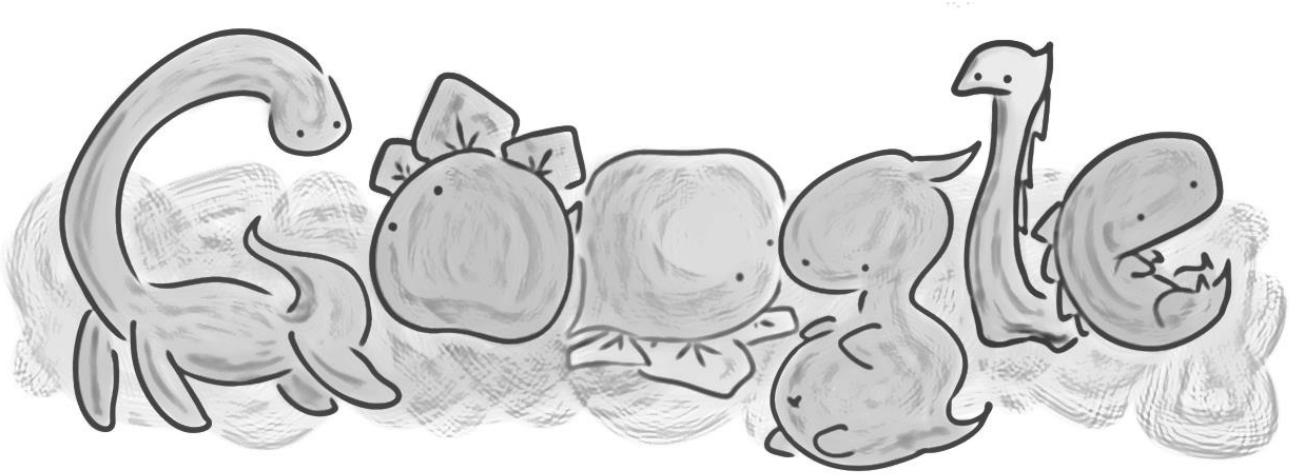
WIC's Beginner's Programming Competition

Dawn of the Dinosaurs: The Problem Set

DO NOT OPEN UNTIL TOLD TO DO SO

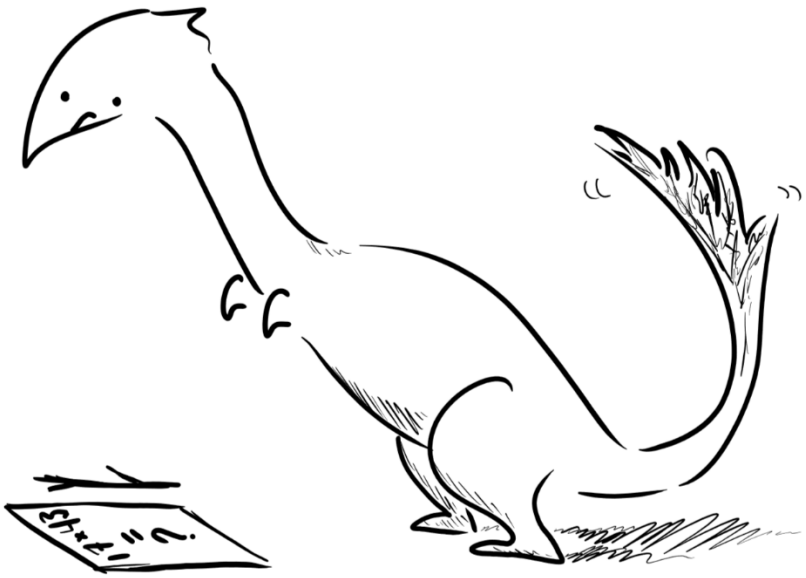
Sponsored by Google

Good Luck, Have Fun, & Enjoy!



Problem 1 : Counting to One One

Marisa the Mononykus has two very strong but stubby arms with only one claw each. This makes it very difficult to talk to her friends about her favorite subject, math... Help her count by teaching her binary numbers!



Given an integer, return the integer that represents it in binary form. Negative numbers will not be tested.

Examples:

Number		Result
3	=>	11
4	=>	100

Sample File Format:
Sample In: One line per input. Each line contains one integer.
Sample Out: One line per output. Each line contains one integer.

Required Method Signature:

```
/*
 * Parameters:        int number - to be turned into binary form
 * Return:            int number in binary form
 */
public static int twoClawCounting(int number) {
     // TODO
}
```

Problem 2 : Take a Lot of Breathers

Victoria the Velociraptor runs cross country for Dinoderpia Academy, where she runs up to 40 miles per hour. The problem is, she has math class right after practice! Unfortunately, she gets no sympathy for being out of breath from practice and gets called on very often to read large numbers in class. To help her read the numbers one digit at a time, split up large integers into arrays of one digit numbers.

Given an integer, return an array of one-digit integers representing each digit of the original number.

Examples:

```
Number          Result
123        =>   [1, 2, 3]
0          =>   [0]
10         =>   [1, 0]
```

Sample File Format:
Sample In: One line per input. Each line contains a single positive integer.
Sample Out: One line per output. Each line contains the array of digits.

Required Method Signature:

```
/*
 * Parameters:   int num - number to be turned into array
 * Return:       int[] - array of ints representing each digit
 *                       of the number, in the order they appear.
 */
public static int[] breakItDownNow(int num) {
     // TODO
}
```

Problem 3 : Colors of the Wind

Peter the Pterodactyl is obsessed with crayons! He wants to count how many unique colors of crayons he has in his vast collection.

Given an array of color names, return the number of unique colors in the array. Strings will only include alphabetical letters, and no spaces.

Example:
`["Blue", "Silver", "Gold", "Blue", "Green"] =>  4`

Sample File Format:
Sample In: One line per input. Each line contains an array of color names separated by spaces.
Sample Out: One line per output. Each line contains the number of unique colors.

Required Method Signature:
```
/*
 * Parameters: String[] colors - names of colors on his crayons
 * Return:      int - number of unique colors
 */
public static int detectBraggingRights( String[] colors ) {
     // TODO
}
```

Problem 4 : Fib? Oh, Not He!

Larry the Leptoceratops habitually lies, but he's working hard to begin an honest life! Now, he claims to have memorized the entire Fibonnacci Sequence by heart. He says he can quickly respond with the Fibonacci Sequence Value of any position in the Sequence. Let's test him on that. Plot twist: For this problem, you are Larry!

Given an integer, find the Fibonacci Sequence Value of that position in the sequence. The 0th position is 0, and the 1st position is 1. Negative numbers will not be tested.

Examples:

Position		Value
5	=>	5
6	=>	8

Sample File Format:
Sample In: One line per input. Each line contains one integer.
Sample Out: One line per output. Each line contains one integer.

Required Method Signature:

```
/*
 * Parameters:   int position - the position in the Fibonacci
 *                              Sequence that we are solving for
 * Return:       int - the value of the integer at the position
 */
public static int noMoreFibbing( int position ) {
     // TODO
}
```

Problem 5 : Infinite Days of Deliciousness

Gary the Gillespisaurus goes to a super-fertilized crazy-magical coconut tree each day to pick coconuts for coconut water. Every day, the tree grows a varying number of coconuts, each with a varying volume of coconut water. Gary is trying to optimize the amount of coconut water he can drink each day. He is limited not by the volume of coconut water, but by the number of coconuts he can take from the tree (If he takes too many, the coconut tree will die!).

Given an array of integers representing the volumes of coconuts on the tree and the maximum number of coconuts he can take, return the total maximum volume of coconut water he can drink today. Negative "volume" and "max" values will not be tested.

Examples:

```
Volumes          Max        Result
[5, 7, 2, 3]     3     =>   15
[3]              5     =>   3
[2, 1, 2, 3]     2     =>   5
```

Sample File Format:

Sample In: Two lines per input. First line contains many integers separated by spaces. Second line contains the integer number of coconuts he can take.
Sample Out: One line per output. Each line contains an integer.

Required Method Signature:

```
/*
 * Parameters:   int[] coconutVolumes - volumes of available coconuts
 *               int max - number of coconuts he can take
 * Return:       int - the volume of coconut water he can drink
 */
public static int poweredByCoconuts( int[] coconutVolumes, int max ) {
     // TODO
}
```

Problem 6 : What Even Are Hands?!

Brandon the Brachiosaurus is learning to tell time. To help, we're going to make an analog to digital converter for him. The current time will be given as an array of three integers in the format [Degree of Hour Hand, Degree of Minute Hand, Degree of Second Hand], where the degrees are calculated counter-clockwise from the positive X axis.

Given such an array of three integers representing the degrees, return the current time in the format [Hour, Minute, Second]. All degrees given will be positive numbers. Round all resulting Hour, Minute, and Second numbers down to the nearest integer.

Example:

Clock Angles		Result
`[138, 305, 330]`	=>	`[10, 24, 20]`

Sample File Format:
Sample In: One line per input. Each line contains 3 integers.
Sample Out: One line per output. Each line contains 3 integers.

Required Method Signature:

```
/*
 * Parameters:  int[] clockAngles - angles of clock hands in the
 *                                  format [H, M, S]
 * Return:      int[] - time [H, M, S]
 */
public static int[] analogIsSoLastEra( int[] clockAngles ) {
     // TODO
}
```

Problem 7 : Reverse, esreveR!

Pam the Plesiosaur has a unique form of dyslexia where she reads every word in a sentence backwards. This means that when she sees the sentence “Dinosaurs are great”, she reads “sruasoniD era taerg”. Our job is to make a device that will help her read more easily.

Given a string with only alphabetical letters and spaces, flip each word (delimited by spaces) and return it so Pam can properly read it. There will be no numbers or punctuation. Maintain the capitalization of each letter.

Example:
```
“Dinosaurs are great”      =>    “sruasoniD era taerg”
```

Sample File Format:
Sample In: One line per input. Each line contains a String.
Sample Out: One line per output. Each line contains a String.

Required Method Signature:
```
/*
 * Parameters:  String text - text to “read”
 * Return:      String - text with words individually reversed
 */
public static String ssoBAekiLgnidaer( String text ) {
     // TODO
}
```

Problem 8 : Evidently, Eggs Came First

Titanosaurs love their offspring, but have very small brains despite their large size; needless to say, they aren't the greatest at counting. When they lay eggs, they like to rawr (talk) to their friends about how many eggs they have. However, the number of eggs they say they have does not always match up with how many eggs they actually have. Figure out if these chatty dinosaurs can count accurately or not.

Given an integer array, check if each number *n* in it appears *n* times. If yes, return true. If no, return false. Negative numbers will not be tested.

Examples:

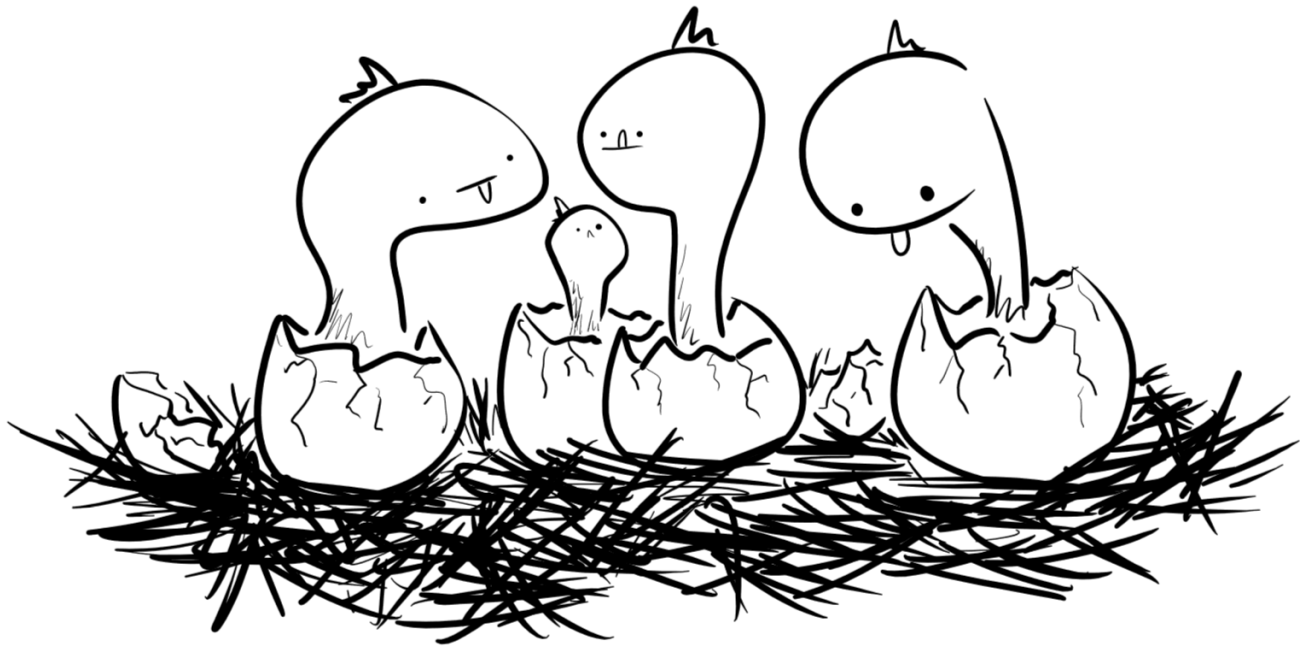
```
[1,2,2,3,3,3,4,4,4,4]       =>    True
[3,1,3,3]                   =>    True
[3,3]                       =>    False
```

Sample File Format:

Sample In: One line per input. Each line contains an array of integers separated by spaces.
Sample Out: One line per output. Each line contains one boolean.

Required Method Signature:

```
/*
 * Parameters:   int[] rawrray - an array of ints to be checked
 * Return:       bool - A boolean telling whether or not every
 *                      number n in the rawrray appeared n times
 */
public static bool ohBabies( int[] rawrray ) {
     // TODO
}
```



Problem 9 : The Friendship Factor

Anthony the Ankylosaurus is unfortunately not very educated, so he thinks that having a prime number of back spikes indicates a prime status in society. Out of a group of Ankylosauruses, he wants to pick out the three individuals with the largest prime numbers of back spikes.

Given an array of integers, find the three largest prime numbers and return the product of them all. If there are less than three primes, return the product of the primes you find. If there are no primes, return 0.

Examples:

Numbers of spikes		Result
`[5, 8, 7]`	=>	`35`
`[2, 3]`	=>	`6`
`[72, 13, 46, 20, 5, 98, 30, 31]`	=>	`2015`

Sample File Format:
Sample In: One line per input. Each line contains an array of numbers separated by spaces.
Sample Out: One line per output. Each line contains an integer.

Required Method Signature:

```
/*
 * Parameters:        int[] numbers - array of numbers
 * Return:            int - largest primes multiplied together
 */
public static int isItOverNineThousandYet( int[] numbers ) {
      // TODO
}
```

Problem 10 : Napping Kin

T-Rexes have great feasts after hunting their prey, and usually go back home to nap after their meals. An array will represent their cave. Each T-Rex is represented by their integer size x, and takes up x spots in the array. Napping T-Rexes can pile on top of each other. Your job is to map out what this cave will look like after all the T-Rexes are home from their meals.

Given an array of integers representing the T-Rexes by their size, and an array of integers indicating the placements of each dinosaur, return the cave array where each index has the sum of the sizes of all dinosaurs in that spot.

The two input arrays, "sizes" and "positions", will always be the same length, as the numbers in "sizes" correspond to the numbers in "positions" for each index. A number x in "sizes" spans x indices in the final cave array. For example, if a T-Rex of size 5 is placed at index 2, there must be 5's at indices 2, 3, 4, 5, and 6 in the final cave array. If there are collisions, where the T-Rexes pile on top of each other, add the sizes of the piled T-Rexes together. If there are no T-Rexes at a certain position, there should be a 0 in the final cave array. The final cave array should not have any 0's at the end. Negative numbers will not be tested.

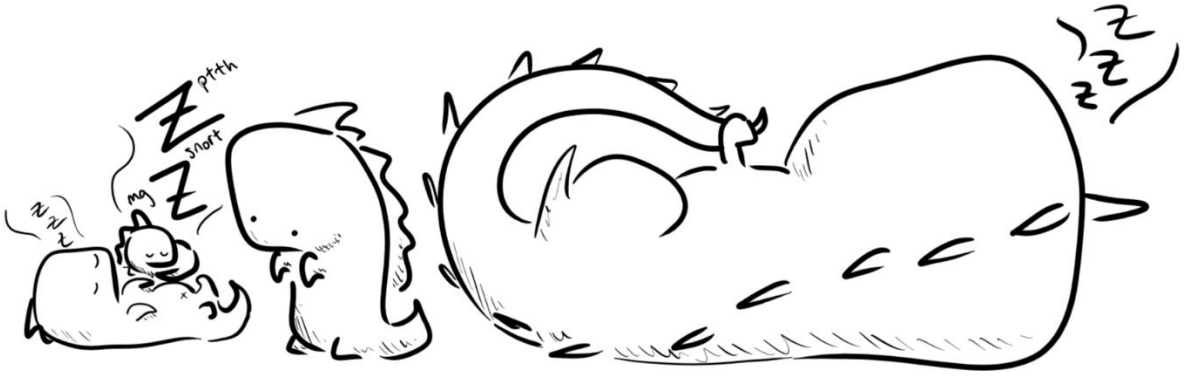
Examples:

Sizes	Positions		Resulting Cave Array
`[6]`	`[2]`	=>	`[0,0,6,6,6,6,6,6]`
`[1,2,3]`	`[0,1,3]`	=>	`[1,2,2,3,3,3]`
`[1,2,3]`	`[0,2,3]`	=>	`[1,0,2,5,3,3]`

Sample File Format:
Sample In: Two lines per input. The first line contains the array of numbers to be placed and the second line contains the array of positions the numbers must be placed in.
Sample Out: One line per output. Each line contains integers separated by spaces.

Required Method Signature:

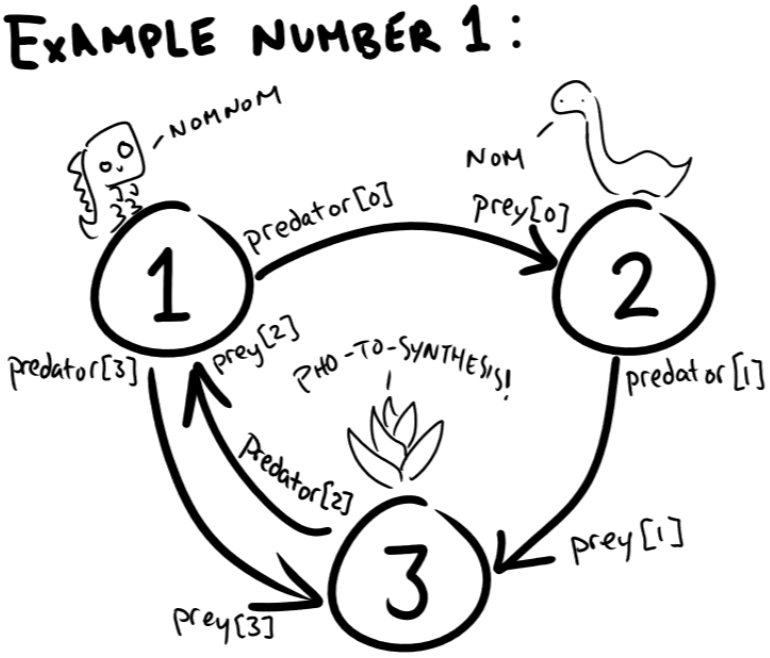
```
/*
 * Parameters:   int[] sizes - the numbers n that are to be
 *                             placed on an array, reappearing n times
 *               int[] positions - the positions that the integers
 *                                 in the "sizes" array must be placed in
 * Return:       int[] - the new array with the integers from
 *                       the "sizes" array in the spots
 *                       indicated by the "positions" array.
 */
public static int[] hushBigDinoBaby( int[] sizes, int[] positions ) {
     // TODO
}
```



Problem 11 : Food Science

Trudy the scientist Troodon (pronounced "TRUE-oh-don") is trying to keep track of who eats who in Dinoderpia with a graph of vertices representing dinosaurs, and directed edges (i.e. arrows) representing the predator-prey relationships. If there are cycles in her graph, then she likely has faulty data!

Given an array of integers representing vertices (where each integer represents a different dinosaur), an array of integers representing the predators, and an array of integers representing the corresponding preys, determine if there are any cycles.



The "predators" and "prey" arrays will always be the same length. An edge is described by the same index in both arrays (e.g. predators[0] to prey[0] is one directed edge, from predators[0] → prey[0]). A cycle is present if you can begin at one node, travel along one or more edges, and end up at that same node. There will be no test case with an empty "vertices" array. Invalid edges (edges from or to a nonexistent vertex) will not be tested. All vertices in the vertex array will be unique (e.g. [3,3] is not a valid vertex array).

Examples:

Vertices	Predators	Prey		Result
`[1,2,3]`	`[1,2,3,1]`	`[2,3,1,3]`	=>	`True`
`[1,2,3]`	`[2,3]`	`[1,2]`	=>	`False`

Sample File Format:
Sample In: Three lines per input. The first line contains an array representing the vertices of the graph. The second line contains an array representing where edges begin from. The third line contains an array representing where the same set of edges from the second array go to.
Sample Out: One line per output. Each line contains one boolean.

Required Method Signature:

```
/*
 * Parameters:   int[] vertices - the vertices of the graph
 *               int[] predators – edge beginnings (where they originate)
 *               int[] prey – edge ends (where they point)
 * Return:       boolean - whether or not the graph contains cycles
 */
public static bool hasCircleOfDeath( int[] vertices,
                                     int[] predators, int[] prey ) {
     // TODO
}
```

Problem 12 : Ping Baby Ping, Don't Ever Look Back...

Pong the Pokasaur is a Papa of many baby Pokasaurs. When adult Pokasaurs want to contact their friends, they send their baby Pokasaurs to go poke them! Each run by a baby Pokasaur is called a ping. But Papa Pong's Pokasaurs just had some sugar! With the sugar rush, their pings are originally very quick! Soon, though, they start getting sleepier and slower. Papa Pong sends his Pokasaurs out one by one. The first ping lasts 2 seconds. Each following ping starts when the prior ping is halfway done, and lasts twice as long as the prior ping.

Given an integer representing how many pings there are, or in other words, how many baby Pokasaurs are sent on runs, return the time it takes for all of the pings to finish. Negative numbers will not be tested.

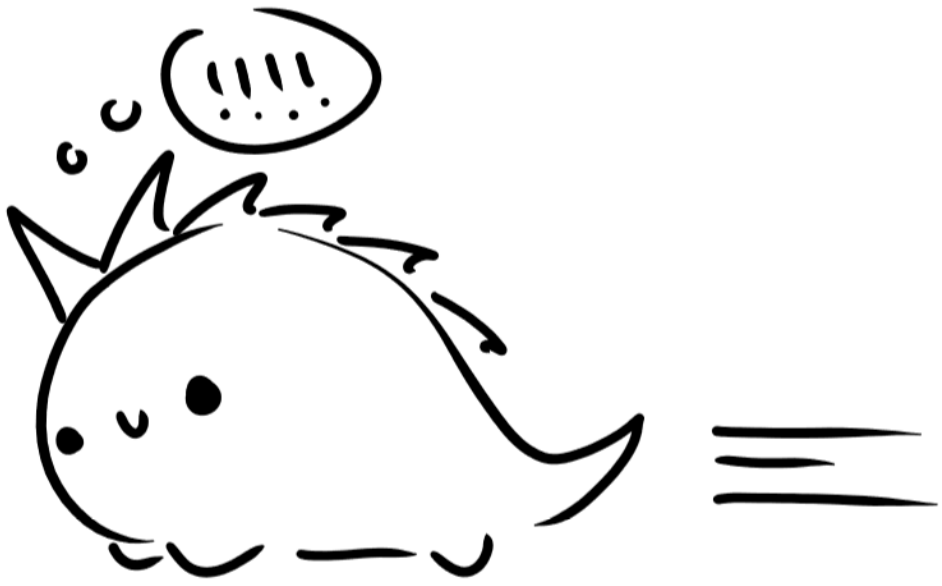
Examples:

numPings		Total Time
1	=>	2
2	=>	5
3	=>	11

Sample File Format:
Sample In: One line per input. Each line contains one integer.
Sample Out: One line per output. Each line contains one integer.

Required Method Signature:

```
/*
 * Parameters:   int numPings - the number of pings to time
 * Return:       int - the seconds timed from the beginning of
 *                     the first ping to the end of the last ping
 */
public static int timePongsPings( int numPings ) {
     // TODO
}
```



Problem 13 : Tuba-saur

Because of the fancy crests on their heads, Parasaurolophuses are likely the loudest dinosaurs! Panda the Parasaurolophus is trying to call her friend over with a loud roar! While she waits for her friend to respond, she decides to do her physics homework on special roars; these special roars have sound waves that increase in height with each wave or cycle. In her homework, sound waves are estimated to be zig zags with straight lines, like a simplified sine wave.

Assume a sound wave begins at y = 0. In one second, the wave travels 3 inches up. In another second, it travels 3 inches down. In the next two seconds, it travels another 3 inches down, and then three inches up to return back to 0. This up, down, down, up trip will be called a cycle. The slope of the wave will always be either 3 inches per second, or -3 inches per second. In the second cycle, the wave doubles its height, taking twice as long to complete its cycle. In the third cycle, the wave doubles its height again, taking twice as long as the second cycle! It continues in this fashion.

Given an integer representing the number of seconds a roar travels, return the maximum height it reaches in inches above the y axis. Round your results down to the nearest integer.

Examples:

Seconds		Max Height
2	=>	3
6	=>	6

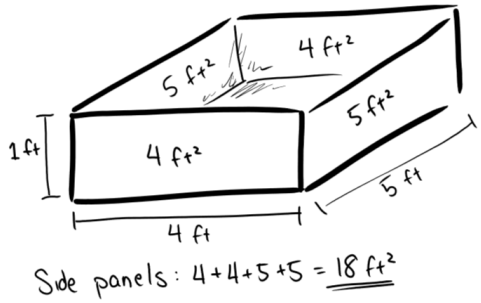
Sample File Format:
Sample In: One line per input. Each line contains one integer.
Sample Out: One line per output. Each line contains one integer.

Required Method Signature:

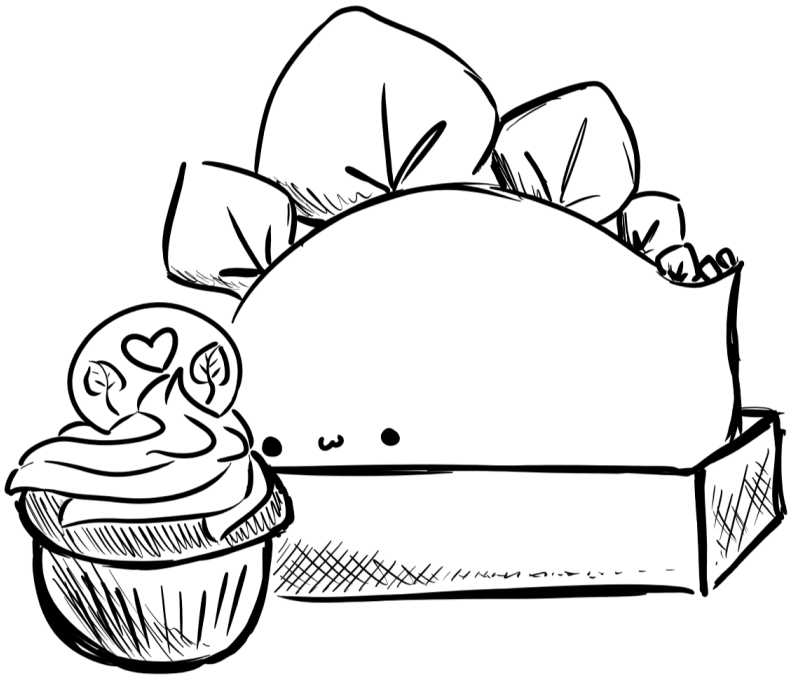
```
/*
 * Parameters:   int seconds - the numbers of second that the
 *                             sound wave has travelled
 * Return:       int - the max height the sound wave achieves
 */
public static int canYouHearMeNow( int seconds ) {
      // TODO
}
```

Problem 14 : Let's Go Boxing!

Steven the Stegosaurus runs a stupendous bakery! He makes custom sized boxes for bowlcakes when customers call in special orders. The boxes he makes must fit the exact number of bowlcakes in each order without having extra space. Bowlcakes take up a 1x1 square foot in a box that is 1 foot tall. The challenge here? He must use as little material as possible! A 1x20 box would take up more material than a 4x5 box as the side panels would measure 42 square feet as opposed to 18 square feet.



Given the number of bowlcakes ordered, return a string with the optimal dimensions of the box in the form "*width*x*length*", with the width being smaller than or equal to the length. All numbers given will be greater than or equal to 0.



Examples:

```
20 => "4x5"
17 => "1x17"
16 => "4x4"
```

Sample File Format:

Sample In: One line per input. Each line contains one integer.
Sample Out: One line per output. Each line contains one string.

Required Method Signature:

```
/*
 * Parameters:   int bowlcakes - number of bowlcakes ordered
 * Return:       String – box dimensions in the format "widthxheight"
 */
public static String feedTheHangry( int bowlcakes ) {
      // TODO
}
```

Problem 15 : The Bouncer

Alicia the Archaeopteryx owns *Success*, a hopping club at the top of a tower. To get into this club, you have to get past the Bouncer! The Bouncer is a tricky contraption consisting of hundreds of trampolines arranged like steps of a staircase. Each trampoline varies in bounciness and will allow you to jump up a certain number of "steps". Some trampolines are broken or missing from the tower, and if you try to land there, you fall back to the bottom.

A trampoline's "Bounciness" indicates the maximum number of steps you can reach from that trampoline. For example, if the trampoline at index 0 has a Bounciness of 2, the farthest you can bounce from that trampoline is to the trampoline at index 2.

Given a 0-indexed array of trampolines represented by their integer Bounciness, find out if you can make it to *Success* or not. Assume you begin at *index 0* and the club is *one step past the final trampoline in the array*. If you can reach *Success*, return the least number of jumps it takes to get there. If you cannot reach *Success*, return the index of the highest/furthest trampoline you *can* reach. All arrays given will have a size of at least 1.

Examples:

```
[5, 2, 3, 4, 1, 0, 0, 3, 0, 0]   =>    3
You start at the Trampoline 0, jump to Trampoline 3 (Bounciness 4), then to
Trampoline 7 (Bounciness 3), and finally, from there to Success, which lies
one step past Trampoline 9.

[1, 2, 0, 0, 5] =>    1
You start at Trampoline 0, jump to Trampoline 1, but cannot reach any more
trampolines; the next trampoline is 3 indices away from Trampoline 1 but
Trampoline 1's Bounciness is only 2.
```

Sample File Format:

Sample In: One line per input. Each line contains integers separated by spaces.
Sample Out: One line per output. Each line contains an integer.

Required Method Signature:

```
/*
 * Parameters:   int[] bouncer - indicates bounciness of each
 *                               trampoline in the Bouncer
 * Return:       int - smallest number of jumps it takes to get to
 *                     the top (if you can get to the top) OR
 *                     the index of the highest trampoline you can
 *                     get to (if you can't get to the top).
 */
public static int defeatTheBouncer( int[] bouncer ) {
     // TODO
}
```