1.Question 1

Growth of functions

Review the math notes on the [growth of functions](https://www.coursera.org/learn/principles-of-computing-2/supplement/OTIAb/math-notes-on-growth-rates-of-functions). Which of the following functions grow at the same rate as

**Correct**

Correct. The ratio of  and tends to 2 as *n* grows large.

2.Question 2

Many algorithms for sorting a list of numbers use comparisons (like greater than or less than) to determine the sorted order of the list. The problem of building fast sorting algorithms using comparisons is well-studied. In fact, we will consider a very elegant sorting algorithm of this type next week.

The fastest algorithms for sorting a list of size *n* share a bound (specified as a simple expression in *n*) for the minimal number of comparisons required to sort **any** list of length *n*.

Use a web search engine (like Google) to look up this estimate and select the answer below that grows at the same rate as this expression.

***n*log(*n*) comparisons**

**Correct**

Yes. In fact, you can prove that comparison sorts require some multiple of *n*log(*n*) comparisons to sort a list.

3.Question 3

Review this [week's practice activity](https://www.coursera.org/learn/principles-of-computing-2/supplement/CG4Vb/practice-activity-sorting-strings) on [sorting strings](http://www.codeskulptor.org/#poc_string_sort.py). The activity discusses a grid-based method for sorting strings that does not require comparisons. Given a list of *n* three-letter words, which expression grows as the same rate as the number of statements executed during this sort?

***n***

**Correct**

Correct. Remember a constant multiple of *n* grows at the same rate as *n*.

4.Question 4

### Stacks and queues

Consider a stack in which we have performed *n* pushes followed by *n* pops. Which of the following are true statements concerning this sequence of operations?

The last element pushed onto the stack is the first element popped off of the stack.

**Correct**

The first element pushed onto the stack is the last element popped off of the stack.

**Correct**

5.Question 5

Consider a queue in which we have performed *n* enqueues followed by *n* dequeues. Which of the following are true statements concerning this sequence of operations?

The last element enqueued into the queue is the last element dequeued out of the queue.

**Correct**

The first element enqueued into the queue is the first element dequeued out of the queue.

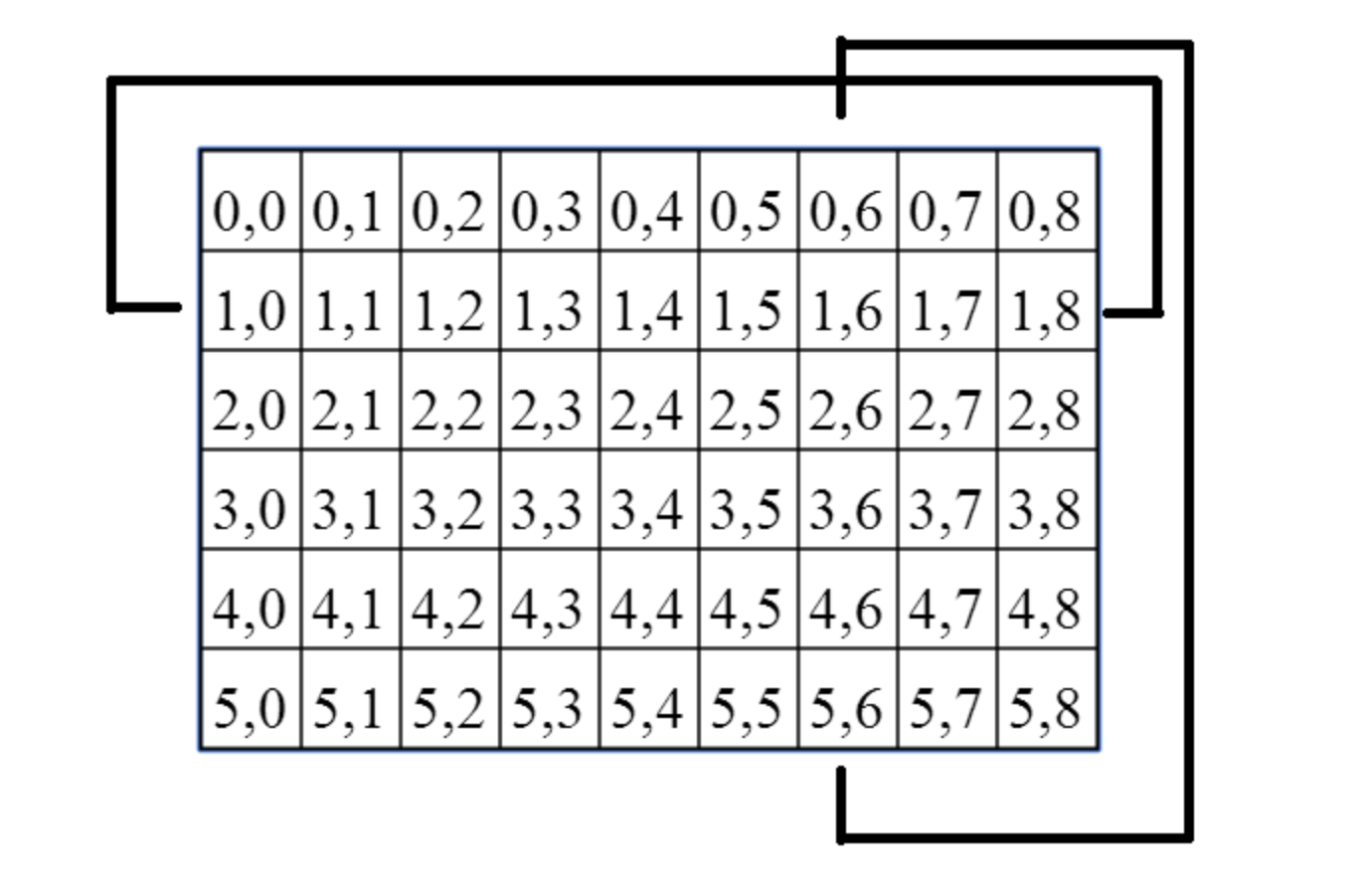
**Correct**

6.Question 6

### 2D grids

Review the [provided implementation](http://www.codeskulptor.org/#poc_grid.py) for this week's grid class. In this implementation, the methods four\_neighbors and  eight\_neighbors  treat the boundaries of the grid as being impassable.

An alternative approach is to treat cells with the same row index on the left and right boundaries as being adjacent and cells with the same column index on the top and bottom boundaries as being adjacent. The thick lines in the figure below indicate that the cells (1, 0) and (1, 8) are horizontally adjacent while the cells (0, 6) and (5, 6) are vertically adjacent.



Which of the following code fragments correctly computes four\_neighbors(row, col)

when the top/bottom rows and left/right columns are treated as being adjacent?

**up = (row - 1) % self.\_grid\_height**

**down = (row + 1) % self.\_grid\_height**

**left = (col - 1) % self.\_grid\_width**

**right = (col + 1) % self.\_grid\_width**

**return [[up, col], [down, col], [row, left], [row, right]]**

**Correct**

Yes. This version computes neighbors correctly.

7.Question 7

### Breadth first search

Consider the [wildfire demo](http://www.codeskulptor.org/#poc_wildfire_student.py) from lecture, which line in the implementation of  update\_boundary

checks whether the fire can spread to an unburned cell?

**if self.is\_empty(neighbor[0], neighbor[1]):**

**Correct**

Yes. This line checks whether the neighbor cell is unburned (EMPTY) or burned (FULL).

8.Question 8

Consider the case in which one steps through the entire breadth first search of the grid in the wildfire demonstration. Which of the following expressions grows at the same rate as the number of statements executed during this breadth first search? Assume the grid has size *m*- by -*n*.

***mn***

**Correct**

Yes. The number of statement executed during breadth first search is proportional to the size of the grid, which is *m*×*n*.

9.Question 9

### Depth first search

Complete [this template](http://www.codeskulptor.org/#poc_stack_template.py) and implement a Stack class. Once your implementation is complete, uncomment the test code at the end of the template and enter the number printed out by this template.

You may wish to create your own test data if you are having trouble with this problem.

**77**

**Correct**

Correct.

10.Question 10

Take the provided Queue class available [here](http://www.codeskulptor.org/#poc_queue.py) and modify the enqueue and dequeue methods to behave like the push and pop methods for your Stack class.

Save this modified class definition. Then, take the [wildfire demo](http://www.codeskulptor.org/#poc_wildfire_student.py) and import this modified definition for the Queue class at the top of the wildfire demo code. In CodeSkulptor, the modified import statement would have the form:

import userXX\_XXXXX as poc\_queue

Now, run this modified demo and add a single cell in the middle of the canvas to the boundary queue prior to starting the search. Which of the images below correspond to a possible state of the grid during the resulting depth first search?

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated