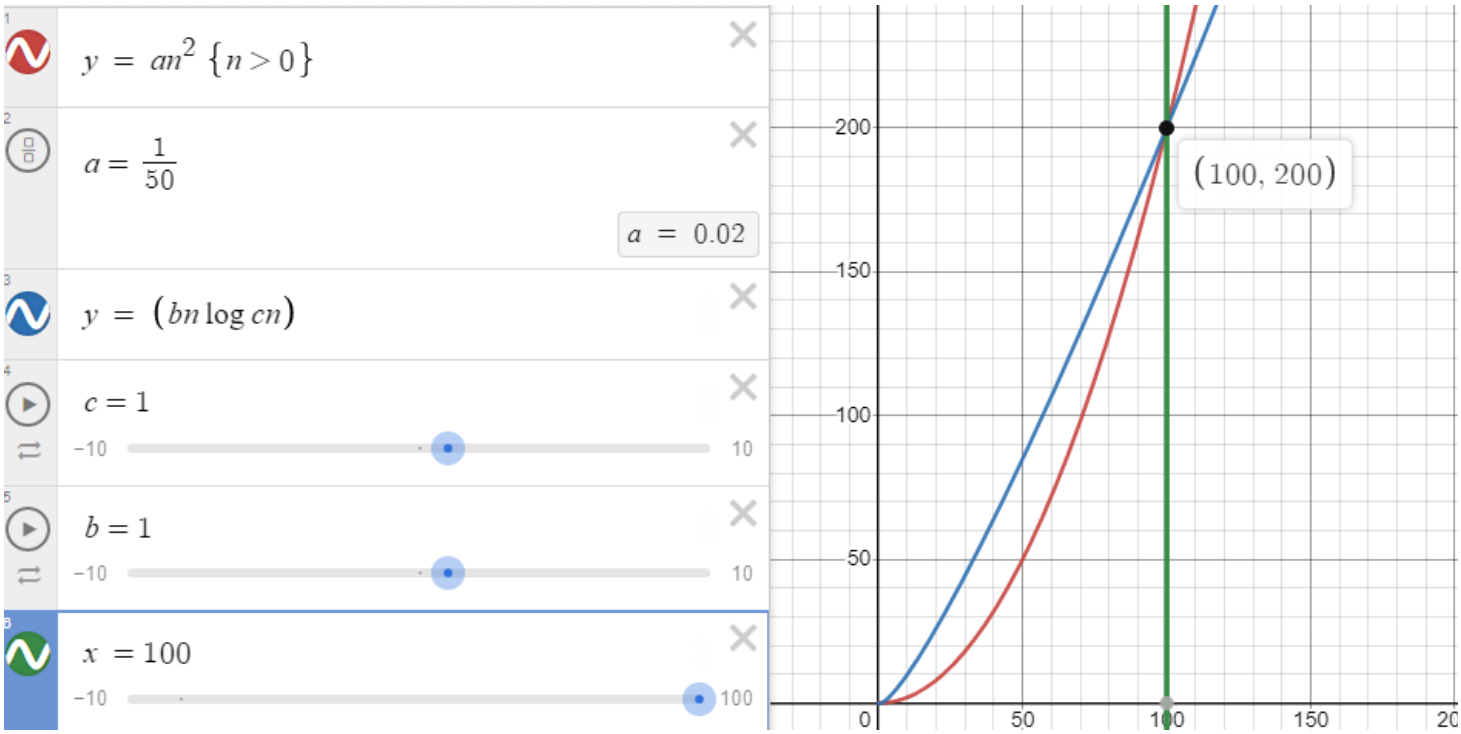
Midterm 1

Question 2

> Al and Bill are arguing about the performance of their sorting algorithms. Al claims that his O(n log n)-time algorithm is always faster than Bill's O(n2)-time algorithm. To settle the issue, they implement and run the two algorithms on many randomly generated data sets. To Al's dismay, they find that if n < 100 the O(n2)-time algorithm actually runs faster, and only when n ≥ 100 the O(n log n)-time one is better. Explain why the above scenario is possible. You may give numerical examples



In Big O(n) notation, the general idea is that as $\lim_{n \to \infty}$ all constants are ignored since a constant*∞ is simply ∞. In Al's cause since we are dealing with very small n numbers (< 100) any constant in front of

$$a * n^2$$

or

$$b * nlog(c * n)$$

where each value of a, b, or c would have a significant effect on n<100. In the example above, when a is <0.02, Bill's graph would have a lower runtime when n<100 but a larger one when n>100.

Question 4

> Write a single tail recursive C++ function that will rearrange
> a vector of int values so that all the even values appear before all the odd values.

```
std::vector<int> recursiveSeparator(std::vector<int> vec, int index){
 cout << index <<endl;
  if (index == vec.size()){
    return vec;
  }
  for(int i = index +1;i<vec.size();i++){
    if (vec[index] % 2 == 1){ //if number is odd
      if (vec[i] % 2 == 0){
          int temp = vec[index];
          vec[index] = vec[i];
          vec[i] = temp;
      }
    }
  }
  return recursiveSeparator(vec, index+1);
}
```

Question 5

> Show by counterexample that the following statement is false: For any positive constant c, f(cn) = O(f(n))

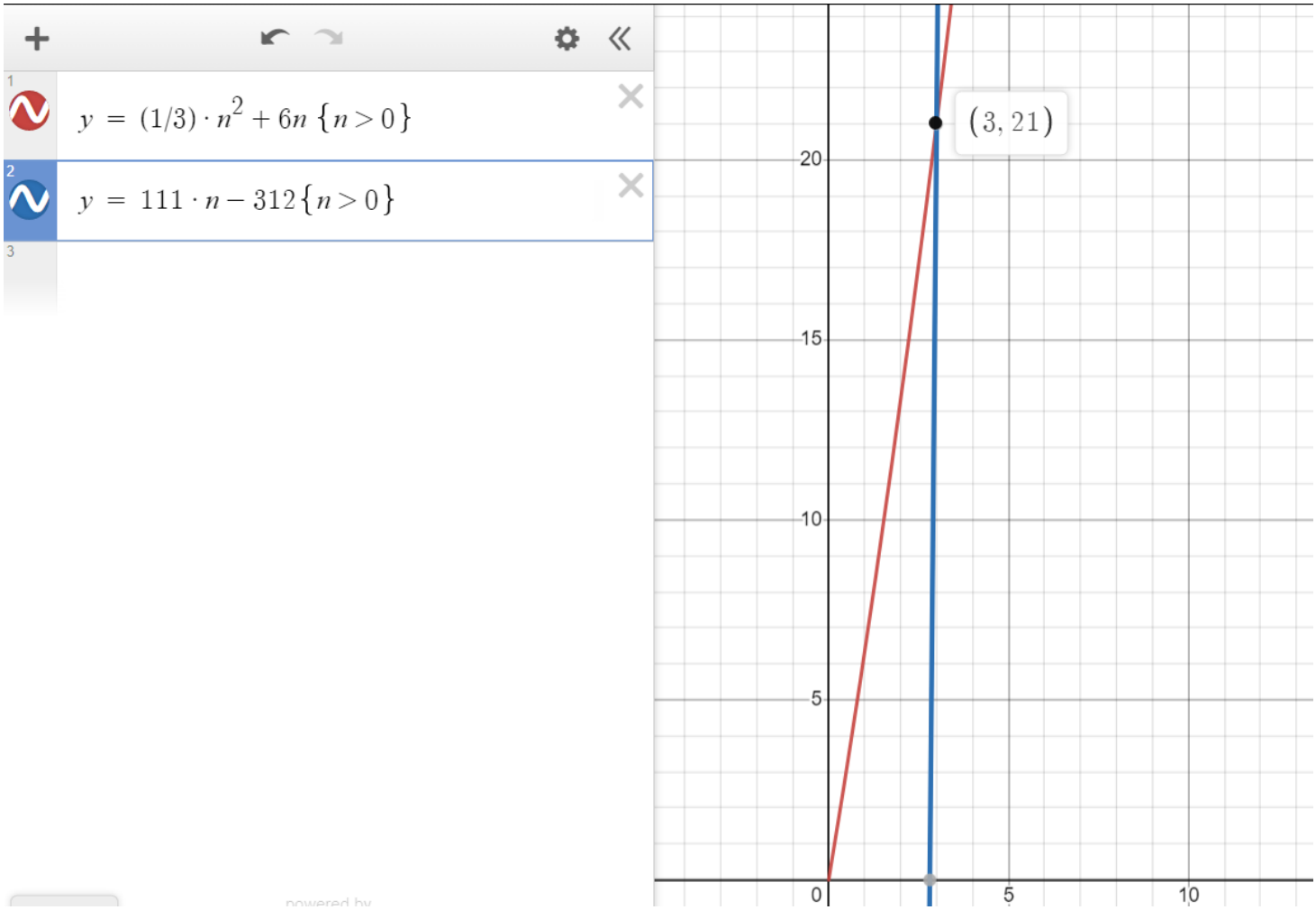
Example: When f(n) = 2^{cn} and O(f(n)) = 2^n, if c = 4 we would have the equation 2^{4n} = 2^n or 16^n = 2^n which is incorrect.

For the two functions 2^n and 2^{4n} calculating the limit for $\lim_{n\to\infty}(2^{4n}/2^n) = \infty$. This is because since 2^{4n} is equivalent to 16^n which clearly grows towards infinity at a faster rate than 2^n. Since the expression above is stating that f(cn) = O(f(n)), where 2^{cn} = 2^n. The $\lim_{n\to\infty}(2^{cn}/2^n)$ would be continous and result in 0 or ∞ for every value of c other 1.

Question 6

> Part 1: Suppose it is known that the running time of an algorithm is (1/3)n2 + 6n, and that the running time of another algorithm for solving the same problem is 111n − 312. Which one would you prefer, assuming all other factors equal?

For the two algorithms O($n^2/3 + 6n$) and O($111n - 312$), when considering big O notation as $\lim_{n \to \infty}$ they would simplify to O(n^2) and O(n). It would almost always be better to use the O(n) algorithm (111n − 312) as for most cases it would be faster and more efficient for larger sample sizes. Looking at the graph below, the only times the (1/3)n2 + 6n algoritm is faster is when n<3 which in most algoritms is irrelevant.



Part 2: Of course, “all other factors” are never exactly equal for two algorithms. There is no question that execution time is important, but there are other factors to consider when choosing an algorithm. Can you think of other considerations ? (Provide 3 other considerations for full marks)

1. A lot of devices that use C/C++ are microcontroller or smaller devices where the programmer has to consider the maximum memeory usage a device has to offer for a given algorithm. An algorithm

with a execelent time complexity does not mean that it will have a good space complexity as well.

2. Another factor to consider is how approachable the function is to for other developers in the workplace. A function with $n * log(2^{ln(n^3)})$ would be much more complex and difficult to implement in other areas or to fix when maintaining certain code. This is because although a alternate algorithm with for example $n * log(n^3)$ time would be worse for bigger n values, it might be more intuitive to use for others.
3. Lastly another factor to consider is how error-proof an algorthm is for unexpected inputs, edge cases, or security risks. Often times when working with user inputted values accounting for such errors can cause a program to be more secure but less efficient in terms of big O time.