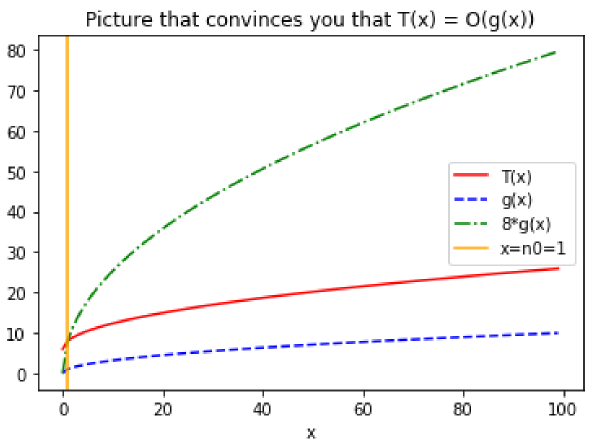
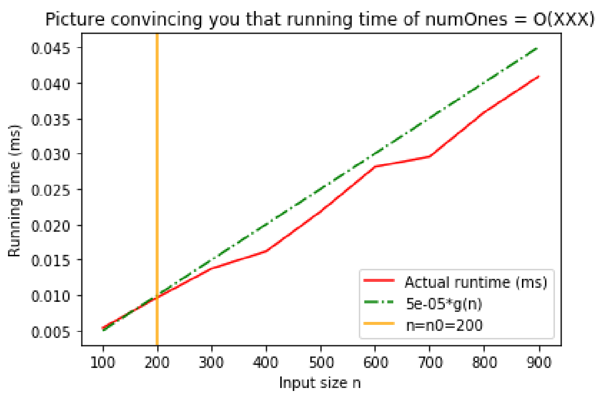
Exercises

1. Choose $c = 8$, $n_0 = 1$ s.t.
   $cg(x) - T(x) = 8\sqrt{x} - 2\sqrt{x} - 6 = 6\sqrt{x} - 6 \geq 0, \forall n \geq n_0 = 1$



2. (a) O(n). The list is traversed once. Each traversal requires constant time.
   (b) Choose $c = 5e-5$, $n_0 = 200$, $g(n) = n$. Both numOnes and g(n) are linear. Choose c and n_0 high enough to guarantee O() property.



   (c) Years. Running time $= c * n =$ 5e10 ms $\sim$ 578.7 years.

3. (a) As in part 1, take $c = 8$, $n_0 = 1$. $c\sqrt{x} - (2\sqrt{x} + 6) = 6\sqrt{x} + 6 \geq 0, \forall n \geq 1$.
   (b) Take $c = 1$, $n_0 = 0$. $n^2 - n \geq 0, \forall n \geq 0$.
   (c) Take $c = log_2(e)$, $n_0 = 1$. $log_2(n) - c * ln(n) = log_2(n) - log_2(e)\frac{log_2(n)}{log_2(e)} = 0, \forall n \geq 1$.
   (d) Prove by contradiction. Suppose $\exists c, n_0$ s.t. $c2^n \geq 4^n, \forall n \geq n_0$. Then $c \geq \frac{4^n}{2^n} = 2^n \to \infty$, so such a c doesn't exist.

Problems

1. (a) Ok.
   (b) Ok.
   (c) Not Ok. The number of multiplications decreases but each multiplication is more expensive.
2. (a) Explanation: First have all toads evaluation each other, which takes $O(n^2)$ time. A trustworthy toad would receive at least floor(n/2) "trustworthy votes."
   Pseudocode:

```
# assume this function is given:
def toadToToadComparison(toad1, toad2):
  '''
  input: two toads
  output tuple: (what toad2 says about toad1,
                 what toad1 says about toad2)

  toad1        toad2         output
  trustworthy  trustworthy   (1, 1)
  trustworthy  tricky        (1, 0) or (0, 0)
  tricky       trustworthy   (0, 1) or (0, 0)
  tricky       tricky        (1, 1), (1, 0), (0, 1) or (0, 0)
  '''
  ...

def naive(toads):
  #input: a list of toads
  #output: a list of trustworthy toads
  nn = len(toads)
  comp = np.zeros((nn, nn), dtype=int) # all toad-to-toad comparisons
  for i in range(nn):
    for j in range(nn):
      if i == j:
        comp[i, i] = 0
        continue
      temp = toadToToadComparison(toads[i], toads[j])
      comp[i, j] = temp[0]
      comp[j, i] = temp[1]
  outputList = []
  for i in range(nn):
    if sum(comp[i, :]) >= nn//2:
      outputList.append(i)
  #print(comp)
  return outputList
```

   (b) We arrange all toads in pairs. Every toad is only assigned once. We have each pair do toad-to-toad comparison. If both report trustworthy, we randomly eliminate one from consideration. Otherwise, we eliminate both.
   To argue correctness, we define the following:
   - The invariant: the fact that there are strictly more trustworthy toads than tricky ones.
   - "One each" pair: a pair of one trustworthy toad and one tricky toad.
   - "Same" pair: a pair of two trustworthy toads, or a pair of two tricky toads.

Argument for correctness:

- There are exactly n/2 comparisons, by design.
- The output has at most n/2 toads, because for each pair, we eliminate at least one toad.
- The output has at least 1 toad. Because to eliminate all toads requires all pairs be "one each" pair, but but there are more trustworthy toads.
- The invariant is true for the output, because

(1) "One each" pairs are eliminated and don't affect the invariant.

(2) Because of (1), "same" pairs alone satisfy the invariant. Among all "same" pairs, the tricky pairs might be eliminated entirely or by half. The trustworthy pairs only get eliminated by half. Therefore the invariant stays true.

```
def divide_even(toads):
  if len(toads) == 2:
    del toads[0]
  for i in range(len(toads)-2, -1, -2):
    if toadToToadComparison(toads[i], toads[i+1]) == (1, 1):
      del toads[i]
    else:
      del toads[i:i+2]
```

(c) We leave alone toad 0, and process the rest even number toads as in (b). The resulting number of toads might be even or odd. If even, we add toad 0 to the result and return.

```
def divide_odd(toads):
  if len(toads) == 1:
    return
  for i in range(len(toads)-2, 0, -2):
    if toadToToadComparison(toads[i], toads[i+1]) == (1, 1):
      del toads[i]
    else:
      del toads[i:i+2]
  if len(toads)%2 == 0:
    del toads[0]
```

(d) Use the function in (b) and (c) to shrink the list to one toad, which is guaranteed to be trustworthy:

```
def findOneEvenOdd(toads):
  while len(toads) > 1:
    if len(toads)%2:
      divide_odd(toads)
    else:
      divide_even(toads)
  return toads[0]
```

(e) Base case: When n=1, because the number of trustworthy toads is strictly larger than that of tricky toads, the one toad must be trustworthy.

Induction: Suppose the program works for all $n < k$. For $n = k$, because divide_even() and divide_odd() turn the number of toads to a number $n' < k$, the next iteration of while is correct per induction hypothesis. The end of the while loop is therefore correct.

(f) $T(n) \leq O(n) + T(n/2)$ so the complexity is $O(n)$ according to the master theorem.

Brute force: Say $T(n) = n + T(n/2)$, $T(1) = 1$.

$T(n) = n + T(n/2) = n + n/2 + T(n/4) = n + n/2 + n/4 + ... = n * 1/(1 - 1/2)$ so $O(n)$

(g) Have the trustworthy toad found in (d) evaluate all other toads.