New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6033: Written Homework 1.
Due Thursday, Feb. 05, 2026, 11:59pm.

Collaboration is allowed on this problem set, but solutions must be written-up individually. If we suspect that you copied your solution directly from AI, we will set up a meeting with you where you will need to explain your solution. If you fail to do so, you will receive zero points for the homework.

Problem 1: Asymptotic time complexity

Determine the asymptotic time complexity of the following piece of code, showing your reasoning and your work.

```
def f(n):
    i = 1
    while i <= n:
        i *= 2
        for j in range(i**2):  # <-- note the range!
            print(i, j)
```

Hint: you might need to think back to calculus to remember the formula for the sum of a geometric progression... or you can check wikipedia.[1]

Problem 2: Formal proof for asymptotic growth

State the growth of each function below using Θ notation in as simplest of terms possible, and prove your answer by finding constants which satisfy the definition of Θ notation.

E.g., if $f(n)$ were $3n^2 + 5$, we would write $f(n) = \Theta(n^2)$ and not $\Theta(3n^2)$.

1. $$f_1(n) = \frac{n^3 + 10n}{\sqrt{n} - 100}$$

2. $$f_2(n) = \log(n^2 + 10)$$

3. Using properties of Θ, what is the asymptotic growth of $f(n)$ defined as
$$f(n) = f_1(n) \times f_2(n)$$

Problem 3: Properties of of asymptotic notation

Suppose $T_1(n), \ldots, T_6(n)$ are functions describing the runtime of six algorithms. Furthermore, suppose we have the following bounds on each function:

$$T_1(n) = \Theta(n^2)$$
$$T_2(n) = O(n \log n)$$
$$T_3(n) = \Omega(n)$$
$$T_4(n) = O(n^2) \text{ and } T_4 = \Omega(n)$$
$$T_5(n) = \Theta(n^3)$$
$$T_6(n) = \Theta(\log n)$$
$$T_7(n) = O(n^{1.5} \log n) \text{ and } T_7 = \Omega(n \log n)$$

[1] https://en.wikipedia.org/wiki/Geometric_series

What are the best bounds that can be placed on the following functions?
For this problem, you do not need to show work.
Example: $T_1(n) + T_2(n)$.
Solution: $T_1(n) + T_2(n)$ is $\Theta(n^2)$.

1. $T_1(n) + T_5(n) + T_2(n)$
2. $T_4(n) + T_5(n)$
3. $T_7(n) + T_4(n)$
4. $T_3(n) + T_1(n)$
5. $T_2(n) + T_6(n)$
6. $T_1(n) \cdot T_4(n)$

Problem 4: Average or expected time complexity

In each of the problems below compute the average case time complexity (or expected time) of the given code. State your answer using asymptotic notation. Show your work for this problem by stating what the different cases are, the probability of each case, and how long each case takes. Also show the calculation of the expected time.

You may assume that *math.sqrt* and *math.log2* take $\Theta(1)$ time.

1.
```python
import random
import math
def boo(n):
    # draw a number uniformly at random from 0, 1, 2, ..., n-1 in Theta(1)
    x = random.randrange(n)
    if x < math.log2(n):
        for i in range(n**2):
            print("Very unlucky!")
    elif x < math.sqrt(n):
        for i in range(n):
            print("Unlucky!")
    else:
        print("Lucky!")
```

2.
```python
def bogosearch(numbers, target):
    """search by randomly guessing. `numbers` is an array of n numbers"""
    n = len(numbers)

    while True:
        # randomly choose a number between 0 and n-1 in constant time
        guess = np.random.randint(n)
        if numbers[guess] == target:
            return guess
```

In this part, you may assume that the numbers are distinct and that the target is in the array.

Problem 5: Theoretical lower bounds

For each problem below, state the largest theoretical lower bound that you can think of and justify. Provide justification for this lower bound. You do not need to find an algorithm that satisfies this lower bound.

Example: Given an array of size n and a target t, determine the index of t in the array.

Example Solution: $\Omega(n)$, because in the worst case any algorithm must look through all n numbers to verify that the target is not one of them, taking $\Omega(n)$ time.

1. Given an array of n numbers, find the second largest number.
2. Given an array of n numbers, check to see if there are any duplicates.
3. Given an array of n integers (with $n \geq 2$), check to see if there is a pair of elements that add to be an even number.