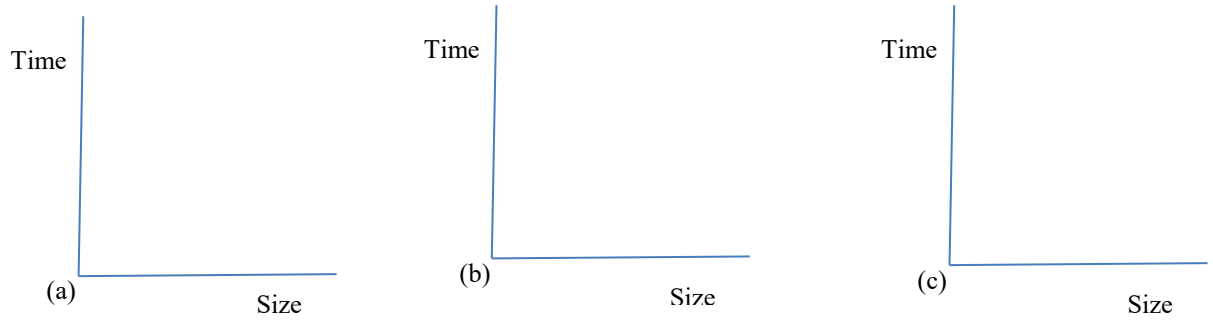
IMPORTANT: Go to Gradescope and download the Quiz7 pdf file: print it, fill it in, and submit it (or use blank pages laid-out with boxes like that pdf)! Do not fill in these pages or you will lose points!

When working on this quiz, recall the rules stated on the Academic Integrity statement that you signed. There is no helper project file for this assignment. Submit your completed written quiz by 11:30pm on **Friday** (1 day later because of In-Lab #3). I will post my solutions to EEE reachable via the **Solutions** link on Saturday.

---

1. (2 pts) Sketch Size vs. Time curves for the two algorithmic complexity classes required in each of the pictures below: for one, write **Impossible** instead: (a) an O(N) algorithm that is **always** faster than an $O(N^2)$ algorithm, (b) an O(N) algorithm that is **sometimes** faster than an $O(N^2)$ algorithm, (c) an O(N) algorithm that is **never** faster than an $O(N^2)$ algorithm..



2. (3 pts) (a) Explain why the logarithmic complexity needs no base: e.g., explain why **$O(Log_2N)$** is the same as **$O(Log_eN)$**. (b) If function f_a has a lower complexity class than function f_b, state the most important/useful fact that is guaranteed by our study of Analysis of Algorithms? (c) Briefly explain under what conditions complexity class analysis does not help us understand the behavior of different algorithms accurately?

3. (6 pts) Suppose that we time three functions in Python: **l**inear **s**earching, **b**inary **s**earching, and **s**orting. We determine their running times, based on the length of their `list` argument as: **$Tls(N) \sim 3.00x10^{-9}N$**, **$Tbs(N) \sim 2.00\ x10^{-8}\ Log_2(N)$**, and **$Ts(N) \sim 6.00\ x10^{-7}N\ Log_2\ N$**. Suppose that we have a `list` of size **N** and we must search it **N** times. Compute the time using these **T** functions for two different approaches to searching: we can do a linear search **N** times or we can first sort the data (just once) and then do a binary search **N** times How long does it take… (write each answer in scientific notation with 3 significant digits, in the form $A.AAx10^B$; compute logarithms exactly in base 2):

a1) to do a **linear search** of **100** values **100** times?

a2) to **sort 100** values and do a **binary search** of them **100** times?

b1) to do a **linear search** of **10,000** values **10,000** times?

b2) to **sort 10,000** values and do a **binary search** of them **10,000** times?

c1) For what problem sizes **N** is it faster do linear searching?

c2) For what problem sizes **N** is it faster to sort and use binary searching?

In problems c1 and c2 only, compute your answer to the closest integer value (you can ignore decimal places). Use a calculator, spreadsheet, or a program to compute (possibly to guess and refine) your answer.

4. (6 pts) The following functions each determine if any **two values** in `alist` sum to `asum`. As is shown in the notes, (a) write the complexity class of each statement on its **right**, where `N` is `len(alist)`. (b) Write the **full calculation** that computes the complexity class for the entire function. (c) Simplify what you wrote in (b).

```
def sumsto_1 (alist,asum):
    for f in alist:
        for s in alist:
            if f+s == asum:
                return (f,s)
    return None
```

(b)

(c)

```
def sumsto_2 (alist,asum):
    aset = set(alist)
    for v in alist:
        if asum-v in aset
            return(v,asum-v)
    return None
```

(b)

(c)

(d) For both functions, which takes longer: finding a solution or finding no solution?

5. (5 pts) Assume that function **f** is in the complexity class **O(N $(Log_2 N)^2$)**, and that for **N = 1,000** (which is **10^3**) the program runs in **.003 seconds**.

(1) Write a formula, **T(N)** that computes the approximate time that it takes to run **f** for any input of size **N**. Show your work/calculations by hand, approximating logarithms (use no calculator), finish/simplify all the arithmetic.

(2) Compute how long it will take to run when **N = 1,000,000 (10^6)**. Show your work/calculations by hand, approximating logarithms (use no calculator), finish/simplify all the arithmetic.

6. (3 pts) Fill in the last line (for each complexity class), which shows what size of a problem (**M**) can be solved in the **same amount of time** on a new machine that **runs 10 times as fast as the old machine**. Solve by hand when you can, use Excel or a calculator when you must. **Hint 1**: Solving a problem in the same amount of time on the new/faster machine is equivalent to solving a problem that takes **10** times the amount of time on the old machine. **Hint 2**: write a formula equating the times to solve problem sizes **N** and **M** using the speedup factor and complexity class. For **O(N)**: **c*M = 10*c*N**; then solve for **M**. Check that your answers aren't crazy: all **Ms** should be bigger than **Ns** for faster machines; the higher the complexity class, the closer **M** should be to **N**.

N = Problem Size	Complexity Class	Time to Solve on Old Machine (secs)	M Solvable in the same Time (1 sec) on a New Machine 10x as Fast
10^6	O(Log_2 N)	1	
10^6	O(N)	1	10 x 10^6
10^6	O(N Log_2 N)	1	
10^6	O(N^2)	1	