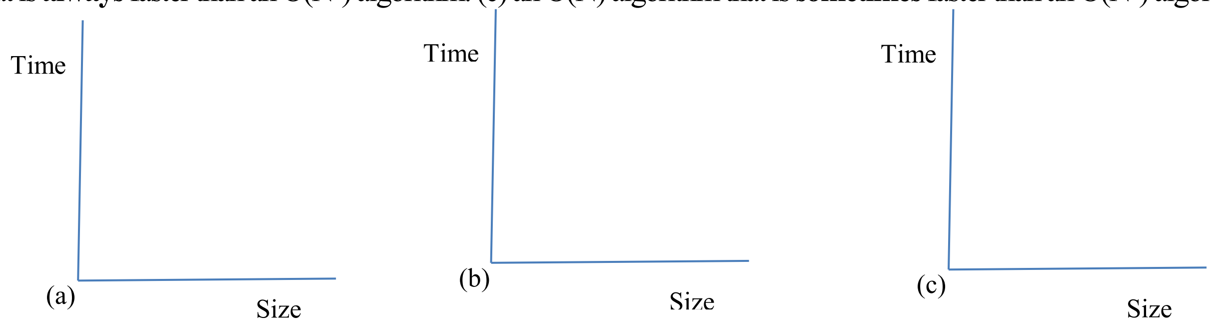
Go to Gradescope to download the pdf file you will print (if you can), fill in, and submit!

When working on this quiz, recall the rules stated on the Academic Integrity statement that you signed. Submit your completed written quiz on Gradescope by Friday December 4th at 11:30pm (in the Irvine time zone). I will post my solutions to Piazza reachable via the **Solutions** link on Saturday morning.

---

1. (3 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 **never** faster than an $O(N^2)$ algorithm. (b) an O(N) algorithm that is **always** faster than an $O(N^2)$ algorithm. (c) an O(N) algorithm that is **sometimes** faster than an $O(N^2)$ algorithm.



2. (2 pts) (a) What "better-known"/simpler complexity class is equivalent to **$O(N \log N^2)$**; briefly explain why. (b) Explain under what conditions `sorted(set(l))` runs faster than `set(sorted(l))` for a `list l` (they both produce the same answer); state the worst-case complexity class of each.

(a).

(b)

3. (6 pts) In 1972, Intel's **8008** processor could execute 200,000 (200 thousand) instructions per second; at present, an Intel **Core 2** processor can execute 3,000,000,000 (3 billion) instructions per second (15,000 times faster). Let's assume that we program the **8008** to run a fast **$O(N \log_2 N)$** sorting algorithm, and program the **Core 2** to run a slow **$O(N^2)$** sorting algorithm. Assume the time to sort **N** values on the **8008** is exactly **$10/200{,}000\ N \log_2 N$** seconds; assume the time to sort **N** values on the **Core 2** is **$2/3{,}000{,}000{,}000\ N^2$** seconds. Here the constant for fast sorting on the **8008** is 5 times as big as the constant for slow sorting on the **Core 2** (both constants are divided by the speed of the machines the algorithm runs on).

a1) About how long does it take the **8008** to sort 1,000 values?

a2) About how long does it take the **Core 2** to sort 1,000 values?

b1) About how long does it take the **8008** to sort 1,000,000 values?

b2) About how long does it take the **Core 2** to sort 1,000,000 values?

c1) For what problem sizes **N** is it faster to use the **8008** for sorting?

c2) For what problem sizes **N** is it faster to use the **Core 2** for sorting?

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 two functions each determine the distance between the two closest values in list `l`, with `len(l)` = `N`. (a) Write the complexity class of each statement in the box on its right. (b) Write the **full calculation** that computes the complexity class for the entire function. (c) Simplify what you wrote in (b).

```
def closest(l:[int])->int:
  a = set()
  for i in range(len(l)):
    for j in range(len(l)):
      if i != j:
        a.add(abs(l[i]-l[j]))
  return min(a)
```

(b)

(c)

```
def closest(l:[int])->int:
  a = sorted(l)
  m = -1
  for i in range(len(a)-1):
    if m==-1 or abs(a[i+1]-a[i])<min
      m = abs(a[i+1]-a[i])
  return m;
```

(b)

(c)

5. (5 pts) Assume that function **f** is in the complexity class **O(**$N\ (Log_2N)^3$**)**, and that for **N = 1,000,000** the program runs in **80 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, finish/simplify all the arithmetic.

(2) Compute how long it will take to run when **N** = **1,000,000,000**. Show your work/calculations by hand, approximating logarithms, finish/simplify all the arithmetic.

6. (3 pts) Fill in the last line (for each complexity class), which shows the size of a problem (**M**) that can be solved in the **same amount of time** on a new machine that **runs twice as fast as the old machine**. Solve by hand when you can, use Excel or a calculator when you must: I used external tools only for **O(N** $\mathbf{Log_2}$ **N)** and **O(**$\mathbf{N^2}$**)** solving those to 3 significant figures. Solving a problem in the same amount of time on the new/faster machine is equivalent to solving a problem that takes ten times the amount of time on the old machine. See the **O(N)** for an example. Check that your answers aren't crazy. Hint: write a formula equating problem sizes N and M using the speedup and complexity class. For linear complexity: 2*c*N = c*M; then solve for M.

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