Quiz 1

- Don't Panic. Often, things aren't as difficult as they may first appear.
- Write your student id *clearly* in the top right on every page (Do this now).
- The quiz contains 3 problems. You have 90 minutes to earn 45 points.
- The quiz contains 10 pages, including this one and 3 pages of scratch paper.
- The quiz is closed book. You may bring one double-sided sheet of A4 paper to the quiz. (You may not bring any magnification equipment!) You may **not** use a calculator, your mobile phone, or any other electronic device.
- Write your solutions in the space provided. If you need more space, please use the scratch paper at the end of the quiz. Do not put part of the answer to one problem on a page for another problem.
- Read through the problems before starting. Do not spend too much time on any one problem.
- Show your work. Partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- If you finish early, drop off your quiz in the front, and leave quietly.
- You may not bring any part of this quiz (even the scratch pages) out of the room.
- Have integrity. Do not share any information about this quiz publicly or with others who have not yet taken it.
- Good luck!

Problem #	**Name**	**Possible Points**	**Achieved Points**
1	Algorithm Analysis	20	
2	Sorting Boss's Cats (Again!)	10	
3	k-way Merge	15	
Total:		45	

Student id.: _____

IMPORTANT: Please ensure your student id is correct and legible.

Problem 1. Algorithm Analysis [20 points]

For each of the following, choose the best (tightest) asymptotic upper bound from among the given options. Some of the following may appear more than once, and some may appear not at all. Each problem is worth 5 points. **Please write the letter in the blank space beside the question.**

A. $O(1)$	B. $O(\log n)$	C. $O(n)$	D. $O(n \log n)$
E. $O(n^2)$	F. $O(n^3)$	G. $O(2^n)$	H. None of the above.

Problem 1.a.

$$T(n) = \frac{n^2}{n-2} + \left(\frac{2n}{5}\right)\left(\frac{n^2}{2}\right) + n^2 \log n \qquad T(n) = \boxed{}$$

Problem 1.b. The running time of the following code, as a function of n:

```
public static int loopy(int n){
  for (int i=0; i<n; i++) {
    for (int k=i; k<50; k++) {
      for (int j=0; j<100; j++) {
        System.out.println("Boss!");
      }
    }
  }
  return i;
}
```

$T(n) =$ $\boxed{}$

Hint: Read the code carefully.

Problem 1.c. $T(n)$ is the running time of a divide-and-conquer algorithm that divides the input of size n into two *unequal-sized parts* and recurses on both of them. The first part is of size $3n/5$ and the second part is $2n/5$. It uses $O(n)$ work in dividing/recombining the two parts (and there is no other cost, i.e., no other work done). The base case for the recursion is when the input is of size 1, which costs $O(1)$.

$T(n) =$

Problem 1.d. The running time of the following code, as a function of n:

```
public static int recursiveloops(int n){

  if (n == 1) {
    return 1;
  }

  int a = doWork(n);
  return recursiveloops(a/2);
}

public static int doWork(int n) {
  int j=0;
  for (int i=0; i<n; i++) {
    j++;
  }
  return j;
}
```

$T(n) =$ [ ]

Hint: Given a real number $|r| < 1$, the sum of the series $\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$.

Problem 2. Sorting Boss's Cats (Again!) [10 points]

Someone let all of Boss's cats out of their cages and they're all out of order. We need to get all these cats sorted by their sizes. But moving the cats is really difficult (they scratch and bite!). However, comparing the size of two cats is straightforward and easy (we just look at the two cats we want to compare). So, we want to minimize the worst-case number of moves that needs to be done on the cats. You have a choice of the following sorting algorithms:

a. Insertion Sort

b. Selection Sort

c. QuickSort

d. HeapSort

Which sorting algorithm should you use? Justify your answer: How many swaps or moves do each of these algorithms make?

Problem 3. k**-Way Merging** [15 points]

In class, we learned how Mergesort is able to merge two arrays in linear $O(n)$ time with the help of an extra buffer. Recall that we had two *sorted arrays* that we had to merge into a single sorted array. For example, given arrays $A = [1, 2, 4, 5, 7]$ and $B = [3, 6, 8, 9, 10]$, the merge algorithm would output a result $R = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$.

For this problem, we want to merge k sorted *linked lists* and return one linked list comprising all their elements in sorted order. There are n/k elements in each sorted linked list and all of them have the same number of elements. For example, for $k = 3$ and the following linked lists (each with 4 elements, so $n = 12$):

- $A = 3 \to 5 \to 6 \to 9$
- $B = 2 \to 4 \to 7 \to 8$
- $C = 2 \to 3 \to 9 \to 10$

your algorithm should return:

- $R = 2 \to 2 \to 3 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9 \to 9 \to 10$

Describe a time-efficient algorithm for performing the k-way merge. State the running time of your algorithm in $O(\cdot)$. Be brief but precise. Pseudocode or Java is not necessary unless it helps you to explain. You can assume you already have access to all the algorithms and data structures we have discussed in class. Unless you make a modification, you do not have to describe how the standard methods work.

Problem 4. Extra Problem: Just for Fun. (Guessing Games) [0 points]

Problem 4.a. Consider the following game that can be played by the entire CS2040 class at once!

- Each person in the class submits a "bid" consisting of an integer between 0 and 100.
- Once the bidding is complete, we compute the average value of a bid.
- Whichever bid is closest to $2/3$ of the average, wins.

What should you bid?

Problem 4.b. The following is a class of games designed for three people. It can be played at different levels. The 0-game is played as follows:

- Each player bids an integer between 1 and 8.
- The player that bids the median value wins. (If there are repeated values, then no one wins.)

A level k-game is played as follows:

- Play 8 $(k-1)$-games.
- Whoever wins the median number of games, wins. That is, let x, y, and z be the number of wins of the three players in the $(k-1)$-games. Then whoever wins MEDIAN(x,y,z) is the winner. (If there is no unique median, then it is a tie.)

After the quiz, go play a 2-game and see how it goes!

Scratch Paper

Scratch Paper

Scratch Paper