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 odd page (Do this now).
- The quiz contains 3 problems. You have 75 minutes to earn 60 points.
- The quiz contains 14 pages, including this one and 4 pages of scratch paper.
- The quiz is open book. 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.
- Good luck!

Problem #	**Name**	**Possible Points**	**Achieved Points**
1	Algorithm Analysis	20	
2	Reverse Polish Calculator	20	
3	List Partitioning	20	
Total:		60	

Student id.: ______________________________
IMPORTANT: Please ensure your student id is correct and legible.
Please circle your tutorial group:

Ayush 9am-10am	Irham 10am-11am	Irham 11am-12pm	Irham 12pm-1pm	Ryan 1pm-2pm	Ryan 2pm-3pm	Enzio 4pm-5pm

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.

$$T(n) = \frac{n^3 - n}{3n^2} + \left(\frac{1}{5n}\right)\left(\frac{n^2}{2}\right)$$

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

Problem 1.b. The running time of the `mycode`, as a function of n:

```
public static int mycode(int n){
  return bar(n);
}

public static int bar(int n) {
  if (n <= 1) return 1;

  return fu(n);
}

public static int fu(int n) {
  if (n <= 1) return 0;
  return bar(n/2);
}
```

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

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 $4n/5$ and the second part is $n/5$. It uses $O(1)$ 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 $O(1)$.

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

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/3);
}

public static int doWork(int n) {
  int j=0;
  for (int i=0; i< n; i++) {
    j = j + 2;
  }
  return j;
}
```

$T(n) = \boxed{}$

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

Problem 2. Reverse Polish Calculator [20 points]

There are several ways we can write arithmetic expressions. You should be familiar with the commonly-used *infix notation*, e.g.,"$2 + 3$" (which gives 5) and "$(2 + 3) \times 2$" (which gives 10).

In Reverse Polish notation (RPN), the operator symbol is placed *after* the arguments being operated on. For example, rather than "$2 + 5$", the RPN expression is: "$2, 5, +$". RPN has the nice property that parentheses (or brackets) are not required to specify order — RPN expressions are always evaluated from left to right. For example, the arithmetic expression "$(2+3) \times 4$" in RPN is "$2, 3, +, 4, \times$". More precisely, a list is in RPN if it follows the following (recursive) rules:

1. it contains a single integer
2. it is of a form "$a, b, \circ$" where a and b are RPN expressions and $\circ$ is a (binary) operator, i.e., $+, -, \times, \div$.

Your task in this problem is to evaluate a given RPN. Rather than a string, assume that an RPN expression is provided as a *list* A of size n, and the first element is the left-most element in the RPN. Each of the elements $A[i]$ is an integer or a binary operator. Here are some more examples of infix and RPN, and how the RPN expressions are evaluated:

Infix	Reverse Polish (RPN)	Result	Remarks
$(2+3) \times 4$	$2, 3, +, 4, \times$	24	$+$ is applied to 2 and 3 to give a. Then, $\times$ is applied to a and 4.
$2 \times (3+4)$	$2, 3, 4, +, \times$	14	$+$ is applied to 3 and 4 to give a. Then, $\times$ is applied to 2 and a.
$(3+6) \div (1+2)$	$3, 6, +, 1, 2, +, \div$	3	$+$ is applied to 3 and 6 to give a. Then, $+$ is applied to 1 and 2 to give b. Finally, $\div$ is applied to a and b, i.e., $a \div b$.

Problem 2.a. [6 points] Among the data structures we have learnt thus far, which one would be helpful in evaluating an RPN expression? Provide a brief justification for your answer.

Problem 2.b. [7 points] **Describe the most time-efficient algorithm you can think of to evaluate the RPN A and return the result.** Assume that A is *guaranteed* to be a valid RPN expression. Be precise, but 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. Write the time and space complexity of your method below:

Running time: Space:

Your Algorithm:

Problem 2.c. [7 points] In the previous subproblem, we assumed that the list A was a valid RPN. Let us remove that assumption: **describe the most time-efficient algorithm you can think of to check if the RPN A is a valid RPN, i.e., it obeys the previously stated rules.** Be precise, but 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. Write the time and space complexity of your method below:

Running time: Space:

Your Algorithm:

Problem 3. List Partitioning [20 points]

As we have learnt, partitioning (or pivoting) plays a crucial role in Quicksort. We focussed mainly on sorting *arrays* in the lectures, but what if we had to apply Quicksort on a singly linked list? For the sub-problems below, assume that you are given a singly linked list A that has n integer elements; the elements may or may not be sorted. You may assume you have head and tail pointers.

Problem 3.a. [10 points] **Describe the most time-efficient algorithm you can think of to partition A.** For this sub-problem, **use the first element as the pivot**. Be precise, but 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. Write the time and space complexity of your method below:

Running time: Space:

Your Algorithm:

Problem 3.b. [5 points] **Given your partitioning method in the previous subproblem, what would be overall worst-case time complexity of applying quicksort to a singly linked list?** Briefly justify your answer.

Worst-Case Running time:

Brief justification/explanation:

Problem 3.c. [5 points] **How would your partitioning algorithm given in problem 3.a. change if you had to pick a random element to be the pivot?** Briefly explain the changes and the impact the changes would have on the *average-case running time of using the random partitioning in Quicksort on a singly-linked list*.

Average-case Running Time:

Brief justification/explanation:

Scratch Paper

Scratch Paper

Scratch Paper

Scratch Paper

— End of Paper —