Data Structures and Algorithms
National University of Singapore
Seth Gilbert

This is a quiz from Week 5 of 2015 of CS2020.

Beware that CS2020 covered some topics not covered by CS2040S, and conversely we have covered some topics in CS2040S that were not covered by Week 5 in CS2020.

Quiz 1

- Don't Panic.
- Write your name on every page.
- The quiz contains seven problems. You have 100 minutes to earn 100 points.
- The quiz contains 16 pages, including this one and 2 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.
- Good luck!

Problem #	**Name**	**Possible Points**	**Achieved Points**
1	Sorting Jumble	10	
2	Algorithm Analysis	16	
3	Barley Mow	10	
4	Buggy Code I	16	
5	Matrix Math	14	
6	Buggy Code II	14	
7	Mind the Gap	20	
Total:		100	

Name: _____ Matric. Num.: _____

Please circle your discussion group:

Pallav Tues. 2pm	Davin Tues. 2pm	Mingyu Tues. 4pm	Sharon Tues. 4pm	Shi-Jie Thurs. 10am	Yik Jiun Thurs. 10am	Xuanyi Thurs. 12pm	Chuyu Thurs. 4pm

One of my favorite sorting questions. You remember how all the sorting algorithms work, right?

A hint: you actually don't need to remember most of the step-by-step details, if you can just remember the *invariants* that each of the sorting algorithms maintain. If a step violates an the invariant for algorithm XYZ, then it is not XYZ.

Problem 1. Sorting Jumble. [10 points]

Your goal is to identify which sorting algorithm is being used in which case. The first column in the table below contains an unsorted list of telephones. The last column contains a sorted list of telephones. Each intermediate column contains a partially sorted list of telephones.

Each intermediate column was constructed by beginning with the unsorted list at the left and running one of the sorting algorithms that we learned about in class, stopping at some point before it finishes. Identify, below, which column was (partially) sorted with which algorithm.

Hint: Do not just execute each sorting algorithm, step-by-step, until it matches one of the columns. Instead, think about the invariants that are true at every step of the sorting algorithm.

Unsorted	**A**	**B**	**C**	**D**	**E**	**Sorted**
OnePlus	Apple	Apple	Motorola	Apple	Apple	Apple
Vodafone	Ericsson	Ericsson	LG	Asus	Ericsson	Asus
Samsung	HTC	HTC	Blackberry	Blackberry	HTC	Blackberry
Yotaphone	Nokia	Nokia	Huawei	Ericsson	Asus	Ericsson
Nokia	OnePlus	OnePlus	HTC	HTC	Kyocera	HTC
HTC	Samsung	Samsung	Apple	Nokia	Huawei	Huawei
Apple	Vodafone	Vodafone	Ericsson	OnePlus	Blackberry	Kyocera
Ericsson	Yotaphone	Yotaphone	Kyocera	Yotaphone	LG	LG
Kyocera	Kyocera	Asus	Asus	Kyocera	Nokia	Motorola
Asus	Asus	Kyocera	Nokia	Vodafone	Motorola	Nokia
Sony	Sony	Huawei	OnePlus	Sony	OnePlus	OnePlus
Huawei	Huawei	Sony	Sony	Huawei	Samsung	Samsung
Blackberry	Blackberry	Blackberry	Yotaphone	Samsung	Sony	Sony
LG	LG	LG	Samsung	LG	Vodafone	Vodafone
Xiaomi	Xiaomi	Xiaomi	Xiaomi	Xiaomi	Xiaomi	Xiaomi
Motorola	Motorola	Motorola	Vodafone	Motorola	Yotaphone	Yotaphone
Unsorted	**A**	**B**	**C**	**D**	**E**	**Sorted**

A 3
B 4
C 5
D 2
E 1

1. BubbleSort
2. SelectionSort
3. InsertionSort
4. MergeSort (top down)
5. QuickSort (with first element pivot)

Always good to start with a little theory in the morning, amirite?

Problem 2. Algorithm Analysis [16 points]
For each of the following, choose the best (tightest) asymptotic function T from among the given options. Some of the following may appear more than once, and some may appear not at all.

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

Problem 2.a.

$$T(n) = \frac{n^6 - 4n^2 + 2n - 17}{2n^4 + 2n + 2020}$$

$T(n) =$ E

Problem 2.b.

$$T(n) = \sum_{i=1}^{n} \frac{n}{i}$$

$T(n) =$ c

diverges - not upper bounded by 2

Problem 2.c.

$$\begin{aligned} T(n) &= 6T(n/6) + 6n \\ T(1) &= 1 \end{aligned}$$

$T(n) =$ D

Problem 2.d. The running time of the following code, as a function of n:

```
public static int loopy(int n){
    int j = 0;
    while (n > 0) {
        j++;
        n /= 2;
    }
    return j;
}
```

$T(n) =$ B

Bonus points for singing it loudly and in tune??
No one took me up on it during the midterm.

(This was a pretty tricky question, it turns out.)

Problem 3. Jeff Erickson's Barley Mow [10 points]

Jeff Erickson is a professor at UIUC in Illinois. One of Jeff Erickson's favorite songs is "The Barley Mow," which he loves to sing to his algorithms classes. The "Barley Mow" is a traditional Devonian/Cornish drinking song that can be constructed according to the following algorithm, as adapted by Jeff Erickson. Assume that `container[i]` is the name of a container[1], and that it holds 2^i ounces of your favorite beverage.

```
BarleyMow(n)

  Here's a health to the barley-mow, my brave boys,
  Here's a health to the barley-mow!
  We'll drink it out of the jolly brown bowl,
  Here's a health to the barley-mow!

  For i = 1 to n do
    We'll drink it out of the container[i], boys,
    Here's a health to the barley-mow!
    For j = i downto 1 do
        The  container[j],
    And the jolly brown bowl!
    Here's a health to the barley-mow, my brave boys,
    Here's a health to the barley-mow!
```

Problem 3.a. Suppose each container name `container[i]` is a single word, and you can sing four words a second. How long would it take you to sing `BarleyMow(n)`? Give a tight asymptotic bound.

O(n^2)

[1] One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon,half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

Problem 3.b. Suppose each time you mention the name of a container, you drink the corresponding amount of your favorite beverage: one ounce for the jolly brown bowl, and 2^i ounces for each container[i]. Exactly how many ounces of beer would you drink if you sang BarleyMow(n)? (Give an exact answer, not just an asymptotic bound.)

2^(n+1) + n2^n - 1

Problem 3.c. *Extra credit: 1 point*
If you want to sing this song for $n > 20$, you will have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases[2]. Suppose `container[n]` has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing `BarleyMow(n)`? Give a tight asymptotic bound. *Hint: Recall that* $\log(n!) = \Theta(n \log n)$.

n^2 logn

[2]We'll drink it out of the hemisemidemiyottapint, boys!

CS2020 spent a lot more time on Java (because it had to cover all of CS2030 and CS2040 in one semester), so there were more questions on Java. (However, we have done *some* Java in CS2040.)

Problem 4. Buggy Code: AddEmUp. [16 points]

As part of your work for MumbleGrumbleCorp, you have been asked to debug the following code written by your predecessor who was recently fired. Strangely, it does not seem to work.

```
1:  public class AddEmUp
2:  {
3:      public static void sumThings(int[] intArray)
4:      {
5:           int n = intArray.length - 1;
6:           int sum = 0;
7:           while (n >= 0) {
8:                value = intArray(n);
9:                if (value < 0);
10:                   value = -1*value;
11:               sum += value;
12:          }
13           return sum;
14:     }
15:
16: }
```

Problem 4.a. There are several bugs that will *prevent this program from compiling*. Only list bugs that prevent compiling (not warnings, bad coding style, or bugs that yield the wrong answer.) Identify (in fifteen words or less, each) two such problems with this code. Specify the line number.

Bug 1.

Line 13: Expect return type of void but got return type of int

Bug 2.

Line 8: intArray used as method rather than an array

Problem 4.b. Once you have fixed those compilation errors, there are *still* several bugs that keep the code from working as intended. The program is supposed to calculate the sum of the absolute values of the integers in the array. Identify (in fifteen words or less, each) two such problems with this code.

Bug 1.

Infinite recursion as n>= 0 always true

Bug 2.

line 10 will always be called even for positive value

A little bit of problem solving… see if you can apply some techniques from class to come up with a simple and clean solution to this problem!

There are a few different ways that you might solve this problem.

Problem 5. Matrix Math. [14 points]

Assume you are given an $n \times n$ matrix M as input:

3	7	5	12	8	4
19	4	3	8	9	1
5	8	14	22	2	1
18	8	5	2	6	17
4	9	8	3	7	11
15	12	3	25	4	8

Your job is to find the maximum value that appears in every row. For example, in the matrix above, the answer is 8: the value 8 appears in every row, and no larger value appears in every row.

Describe the most time-efficient algorithm you can think of to solve this problem: (Be precise and detailed, but pseudocode or Java is not necessary unless it helps you to explain. You can assume you already have access to all the algorithms we have discussed in class. You do not need to restate how they work.)

```
Create an array to store duplicate values.
Use a hashmap.
For each row, search the hashmap for the value.
If (data == null) {
        add a data.counter = 0
        add data.latest row = current row
} else if (data.latest row == current row) {                    O(n*n)
        // duplicate values in same row
        do nothing
} else {
        increase the data.counter by 1
        update the data.latest row to current row
}

if (row == last row) {
        add values to array if data.counter == total number of rows
}

Use quickselect to find largest value in array                  O(n)
```

What is the running time of your solution?

O(n^2)

And some more Java… This was all about interfaces and inheritance (which we haven't talked about much in CS2040S).

Problem 6. Buggy Code: Urban Farming. [14 points]

Problem 6.a. Consider the following code:

```
1. public class Vegetables implements IGreen, IPurple extends Food {
2.         private static int numVegetables = 0;
3.         private int vegetableIndex = -1;
4.
5.         Vegetables(){
6.                 super(numVegetables);
7.         }
8.
9.         public void countVeggies(){
10.                if (numVegetables > 10)
11.                        System.out.println("Too many vegetables!");
12.        }
13.
14.        @Override for IGreen
15.        public int getGreen(){
16.                return vegetableIndex;
17.        }
18.
19.        @Override for IPurple
20.        public int getPurple(){
21.                return -1;
22.        }
23. }
```

Briefly describe one bug that will prevent this from compiling.

extends then implements

Problem 6.b. Consider the following code:

```
1.   public class Building {
2.        public void washFruit() {
3.             System.out.println("Washing fruit and vegetables.");
4.        }
5.   }
6.
7.   public class GreenHouse extends Building {
8.        public void cleanVegetables() {
9.             System.out.println("Cleaning the roof.");
10.       }
11.  }
12.
13.  public class UrbanFarm {
14.       public static void doChores(House a, House b)
15.       {
16.            a.washFruit();
17.            b.cleanVegetables();
18.       }
19.
20.       public static void main(String[] args) {
21.            Building b = new GreenHouse();
22.            GreenHouse h = new GreenHouse();
23.            doChores(b, h);
24.       }
25.  }
```

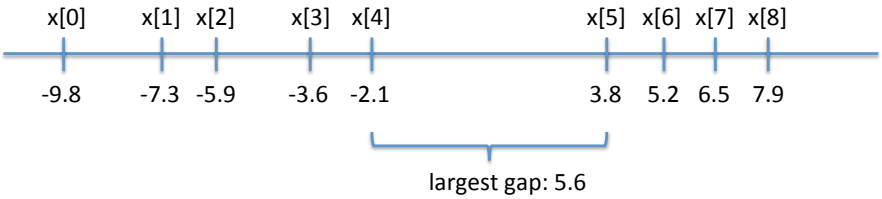
Briefly describe one bug that will prevent this from compiling.

Cannot resolve symbol 'House'

Here's another nice chance to practice your problem solving skills… This was a problem that I really liked (and actually saw show up in a semi-real-world setting). The hardest part might be understanding what is actually going on! Once you see that, it is very similar to problems we have seen already.

Problem 7. Mind the Gap [20 points]

Assume you are given a set of n points on a line. The points are given to you as an array of real numbers $x[0\,..\,n-1]$ where $x[0]$ is the leftmost point on the line, $x[1]$ is the next point, $x[2]$ is the next point, etc. The goal, in the first part of the problem, is to find the pair of consecutive points with the largest gap, i.e., to find an i such that $|x[i+1]-x[i]|$ is largest. Then, the second part asks about finding a gap that is at least as large as the *average* gap. Here is an example input:



Problem 7.a. Give an efficient algorithm to find the largest gap. Your algorithm should return a pair $(i, i+1)$ that maximizes $|x[i+1]-x[i]|$.

In less than four words, what is your basic approach:

Iterate through the array

What is the running time of your approach?

O(n)

Describe your algorithm briefly, in 2-3 sentences. (Use pseudocode here only if necessary.)

Calculate the gap between x[i+1] and x[i], then compare it with current max
if its gap is larger then swap with current max. Return max at end of all iterations.

Problem 7.b. Give an efficient algorithm to find a gap that is at least as big as the *average* gap. The average gap is of size $A = \frac{1}{n}\sum_{i=0}^{n-2} |x[i+1] - x[i]|$. Your algorithm should return a pair $(i, i+1)$ such that $|x[i+1] - x[i]| \geq A$.

In less than four words, what is your basic approach:

Binary search using averages

What is the running time of your approach?

O(logn)

Describe your algorithm briefly, in 2-3 sentences, and then give pseudocode (or Java).

Half the array and calculate the average gap of the left half and right half.
Recurse to the side that have the higher average. If the pointer of start and end
is equal to 1, return the 2 point as the result pair.

```
def binarysearch(A, start, end) {
        if (start - end <= 1) {
                return (A[start], A[end])
        else
                mid = start + (end - start)/2
                leftA =
```

Scratch Paper

Scratch Paper

That was the end of the midterm. Here's another question from later in the year… Look familiar?

Problem 4. Cool cats. [20 points]

Cats are cool. Cats can also be fat. Your goal is to build a dynamic data structure that maintains a database of cats and supports the following type of query:

> Find me the coolest cat that weighs at most 4.3kg.

In more detail, a cat is defined by three things: its name (a string), its coolness (a non-negative real number), and its weight (a non-negative real number). There is no known upper bound on the coolness or weight of a cat. The data structure should support two operations:

- **insertCat(String name, double cool, double weight)**: Add a cat with the specified parameters to the database.
- **findCoolestCat(double w)**: Let C be the set of cats in the database with weights $\leq w$. You should return the coolest cat in the set C.

Your goal is to develop a data structure to efficiently implement these two operations. Your algorithm should be both time and space efficient.

AVL Tree
- Store max Coolness
- Built by comparing weight

Example.

```
insertCat("Poof", 17.2, 4.1);
insertCat("Spots", 42.0, 8);
findCoolestCat(5) → "Poof";
findCoolestCat(9.4) → "Spots";
insertCat("Dot", 9.0, 12);
insertCat("Puddles", 0.8, 2.2);
findCoolestCat(4) → "Puddles";
findCoolestCat(4.1) → "Poof";
```

Continued on next page.

Augmented AVL tree, sorted by coolness
- Store min weight of each subtree at every node

Problem 4.a. Suppose that there are currently n cats in the database. How efficiently can we support these operations?

insertCat(...): O(logn)

findCoolestCat(...): O(logn)

Problem 4.b. Describe (briefly) the main idea behind your data structure for implementing these operations:

Use an AVL tree build the tree by comparing the weights of cat.
- Left subtree (<= weight of parent)
- Right subtree (> weight of parent)

Store the max coolness of subtree in each node - max(left.maxCoolness, right.maxCoolness, coolness)

Insert:
Travel down the tree according to the cat's weight and insert the cat at the right place.
Check balance and use rotations to correct the balance & maxCoolness if necessary
Travel back up to the root while updating the maxCoolness

findCoolestCat:
Search for predecessor of w.
From node p,
    if travel right subtree,
        max(findCoolestCat(p.weight),