Comments on CS1020 Exam (AY2015/6 Semester 2)

Section A: MCQs

The following tables show the percentage of students who chose the correct answers, and of those who chose the most popular wrong answers. Except for Q2, the correct answers were chosen by the majority of the class.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
%students who chose the correct answer	E (62.4%)	E (37.0%)	A (71.0%)	C (83.0%)	B (76.7%)	D (69.9%)	C (63.6%)	C (69.3%)	D (91.9%)	B (75.5%)
%students who chose the most popular wrong answer	B (20.3%)	B (37.3%)	B (18.5%)	E (11.3%)	D (15.8%)	C (11.9%)	B (17.3%)	B (12.2%)	B (4.8%)	D (13.7%)

The easiest question is Q9, and the hardest is Q2, with only slightly more than one third of the class who got it right. If you have selected the most popular wrong answers, please find out your mistakes.

Q11. Answer and Comments

(Comments by Aaron)

This is supposed to be an easy question. If you have attempted last year semester 2's exam paper, you would have realised that it is similar to finding the largest difference in the area of a pair of beads in a list of beads in that paper. You should also be aware that the complexity of this is linear time, i.e. O(n) where n is the size of the list.

Two possible answers are shown on the next page. Variants of them are also acceptable.

The following are some common mistakes made by students:

- Using a doubly nested loop to compare all pairs of fractions. This would give $O(n^2)$ time complexity.
- Sorting the list in increasing order of the fraction value, then finding the difference between the last fraction (which is the largest fraction) and first fraction (which is the smallest). This requires the implementation of the sort algorithm. If you use Collections.sort(), you must implement the compare() method. In any case, this would give O(n log n) time complexity.
- Using the array syntax [ ] on ArrayList.
- Assuming that for a negative fraction, the negative sign follows the numerator. In general, the sign may follow the numerator or the denominator.
- Comparing a fraction directly with an integer, for example: "if (fract1 < 0)", "if (fract1.minus(fract2) < 0)".
- Computing the product of the denominators of all the fractions. This will result in a very large value which is likely to exceed the range of integers.
- Writing the following, which will not compute the value correctly:
  double value = (double) (fractions.get(i).getNumer() / fractions.get(i).getDenom());

- Writing complex code to find the value of a fraction or the difference of two fractions.
- Creating (unnecessary) temporary array, for example, to hold the values of the fractions, or to hold the numerators of the fractions.

```
public static FractionI
       largestDiff( ArrayList<FractionI> fractions ) {

   double minVal = (double) fractions.get(0).getNumer()
                          / fractions.get(0).getDenom();
   double maxVal = minVal;
   int minIdx = 0, maxIdx = 0;

   for (int i=1; i<fractions.size(); i++) {
      double value = (double) fractions.get(i).getNumer()
                            / fractions.get(i).getDenom();
      if (value < minVal) {
         minVal = value;
         minIdx = i;
      }
      if (value > maxVal) {
         maxVal = value;
         maxIdx = i;
      }
   }
   return fractions.get(maxIdx).minus(fractions.get(minIdx));
}
```

Or alternatively,

```
public static FractionI
       largestDiff( ArrayList<FractionI> fractions ) {

   FractionI largest = fractions.get(0);
   FractionI smallest = largest;

   for (FractionI f: fractions) {
      if (f.minus(largest).getNumer()
          * f.minus(largest).getDenom() > 0) {
         largest = f;
      }
      if (f.minus(smallest).getNumer()
          * f.minus(smallest).getDenom() < 0) {
         smallest = f;
      }
   }
   return largest.minus(smallest);
}
```

Q12. Answer and Comments

(Comments by Aaron)

A possible answer is shown below.

```
// 'list' is the given linked list.
// getHead() returns the reference of the first node of 'list'.
ListNode<Integer, String> curr = list.getHead();
ListNode<Integer, String> prev = null;
ListNode<Integer, String> next;
while (curr != null) {
   String instr = curr.getInstruction();

   if (instr.equals("Triple")) {
      curr.setElement(curr.getElement() * 3);
      curr.setInstruction("Done");
   }
   else if (instr.equals("Swap")) {
      if (curr.getNext() != null) { //something ahead to swap
         next = curr.getNext();
         curr.setNext(next.getNext());
         prev.setNext(next);
         next.setNext(curr);
         next.setInstruction("Done");
      }
      curr.setInstruction("Done");
   }
   prev = curr;
   curr = curr.getNext();
}
```

This is quite a well-attempted question. The following are some common mistakes:

- Not using a loop to process all nodes.
- Not updating the three links correctly for a "swap" operation.
- For students who use this while condition: "while (curr.getNext() != null)", forgetting to process the last node of the list.
- Using "==" instead of "equals()" method to compare strings.

Q13. Answers and Comments

(Comments by Ivan)

Q13a. Answers:

Statement	*front*	*back*
Queue Q(5);	'\0'	'\0'
Q.enqueue('A');	'A'	'A'
Q.enqueue('B');	'A'	'B'
Q.enqueue('C');	'A'	'C'
char c = Q.dequeue();	'B'	'C'
Q.enqueue('A');	'B'	'A'

* We also accepted the values (indexes) of instance variables `front` and `back`.

(i) *Not sure how Queue works*

A number of students confused Queue with Stack, while others failed to **enqueue() to the back** and **dequeue() from the front** of the Queue.

An easy way to remember how data is stored in a Queue is, how you (are supposed to) queue for food/bus. It is not right to enqueue() yourself to the front of a FIFO queue!

(ii) *Errors*

Many students identified that there is an error with the line where the Queue Q is supposedly declared and instantiated. The syntax is indeed incorrect, if the code is written in Java. However, the following are **NOT reasons** for the error:

- Datatype of Queue not specified (What is `Queue` then?)
- The Queue is not initialized

Another "error" which is **incorrect** is that there will be a type conversion error for the statement `char c = Q.dequeue()`. Even if each element is a reference to a `Character`, what happens when you assign a `Character` to a `char` variable in Java?

(iii) *Default char value*

Which of the following is the default value of a char, i.e. the value that a char instance variable is initialized with?

- **`null`** char (what does null mean?)
- `''` (nothing?)
- `' '` (space)
- `'\n'` (newline)
- **`'0'`** (ASCII 48_{10} or ASCII 0x30)
- **`'\0'` (ASCII 0)**
- More than one of the above
- None of the above

Also, 'a' is a char, while "a" is a String literal.

Q13b. Answers:

Next token	Actions	Content of stack (left-most item is at the top of the stack)
+	**v1 = S.pop() = 30** **v2 = S.pop() = 6** **S.push(v2+v1)**	**[6 72]** **[72]** **[36 72]**
–	**v1 = S.pop() = 36** **v2 = S.pop() = 72** **S.push(v2-v1)**	**[72]** **[]** **[36]**
9	**S.push(9)**	**[9 36]**
/	**v1 = S.pop() = 9** **v2 = S.pop() = 36** **S.push(v2/v1)**	**[36]** **[]** **[4]**
8	**S.push(8)**	**[8 4]**
5	**S.push(5)**	**[5 8 4]**
*	**v1 = S.pop() = 5** **v2 = S.pop() = 8** **S.push(v2*v1)**	**[8 4]** **[4]** **[40 4]**
-	**v1 = S.pop() = 40** **v2 = S.pop() = 4** **S.push(v2-v1)**	**[4]** **[]** **[–36]**

(i) *Stack elements*

What task is being performed in this question?

- Converting an infix expression to a postfix expression.
- **Evaluating a postfix expression, i.e. calculate and arrive at a single result.**

Each task requires different elements to be stored temporarily:

- Conversion - (Brackets and) operators stored on the Stack, but NOT operands.
- Evaluation - **Operands** (numbers) stored on the Stack, but **NOT operators.**

(ii) *Binary postfix operators, or...?*

Which of the following infix expressions is the equivalent of `72 6 30 +` postfix:

- `(72 + 6 + 30)` ☻ `...`
- **`72 ☻ (6 + 30) ...`**

where ☻ is an operator that appears later in the postfix expression.

The answer affects how many operands (numbers) are popped in each operation.

(iii) *Stack --> LIFO*

Watch the orientation of the stack while writing its contents. The top of the Stack is to be drawn facing the LEFT side.

Now if the Stack contains `[v1 v2 v3]`, and we want to perform operation ☻, what should the algorithm compute? (Think of what the original expression would look like.)

- `v1 ☻ v2`
- **`v2 ☻ v1`**

If we use the other expression, operators that are non-commutative will produce the wrong result.

Q14. Answer and Comments

(Comments by Prof Tan)

```
public static String generate(int n) {
   if (n == 1)
      return "a";
   else
      return generate(n-1) + (char)(96+n) + generate(n-1);
}
```

This is supposed to be a give-away question as the given hint already suggested to students how to break the problem into sub-problems. But it turned out that the main problem many students faced was to convert an integer to its corresponding character.

A simple casting (char) will change an integer (ASCII) to its corresponding character. There is no need to use any toString() method to convert it to String as the generate() method returns a String and hence during the concatenation (+), the character will automatically be widen to a string.

Other common errors:

- Many students wrote if (n = 1) when it should be if (n == 1).
- Writing return 'a' (a character) instead of "a" (a string).
- Many students used System.out.print() statement to print out the string. Hence their method does not return any value. They did not realise that the method is supposed to return a String, according to the given method header.

Q15. Answers and Comments

(Comments by Ivan)

(a) **O(n^2)** (b) **O(n)** (c) **O(n log n)**

(i) *Big-O notation*

If an algorithm performs $\frac{n}{2}$ + 4 basic operations, the algorithm runs in ____ time:

- n
- **O(n)**
- o(n)
- More than one of the above

Don't forget, the term "Big-O" already tells you that the 'O' has to be in uppercase.

(ii) *Loops over constants* as in parts (a) and (c)

In theory, if some operation runs k times (O(k) algorithm), but k is independent of the input/problem size n, then the algorithm runs in O(1) time by the definition of a function being "bounded from above" by another.

Therefore, when analyzing the time complexity of a code fragment theoretically, you can "remove" the nested loop that iterates a constant number of times from your analysis.

You are also expected to know that:

- O(10n) can be simplified to O(n), discarding coefficients.
- O(n log n + n) can be simplified to O(n log n), discarding lower-order terms.

(iii) *What are you analyzing?*

Don't lose sight of what your goal is: Finding out roughly how many times some operation is executed, dependent on some input/problem size. Statements `int x = n * n`, `int y = x * n`, and `int z = y/n` do NOT affect how many times an operation is executed.

Q15c.

The number of times the nested loop over i iterates is **independent of the control variable** of the outer loop j.

Therefore, it may be easier to "ignore" the loop over i, just analyzing the time complexity of the outer loop j, and then multiplying the time complexity of the loop over i to the result.

This is similar to how you can easily evaluate 5x + 3x + 4x, by factorizing x out, adding (5 + 3 + 4), then multiplying the result by x once again.

Q15b.

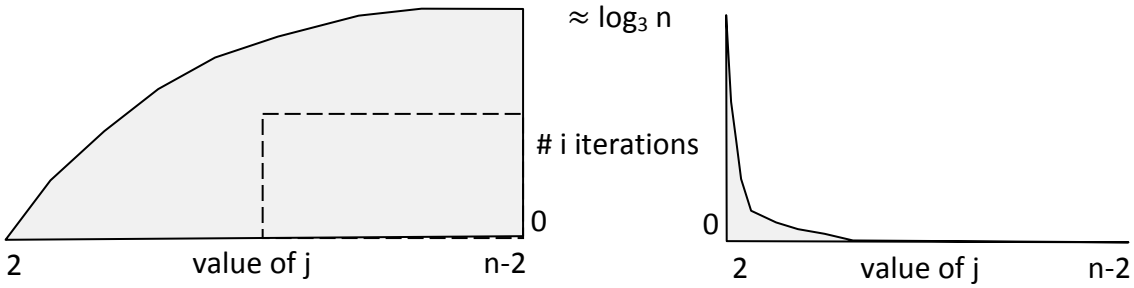
(i) O(log n) loop nested within O(n) loop, ***so result must be O(n log n)...!***
The nested loop over i is dependent on the control variable j of the outer loop. For different values of j, the inner loop may iterate different number of times.

The (outer) loop over j takes values 0, 2, 4, 6, 8, ..., n-6, n-4, n-2
(ii) $1 + \log_3 2 + \log_3 4 + ... + \log_3 (n-2)$, ***so result must be O(log(n!)) = O(n log n)...!***
(iii) $1 + \log_3 2 + \log_3 4 + ... + \log_3 (n-2)$, ***so result must be O((log n)^2), arithmetic series...!***

Firstly, the above series is **NOT an arithmetic series**. The next missing terms are **$\log_3 6$**, $\log_3 8$, **$\log_3 10$**, **$\log_3 12$**. There is no common difference between two consecutive terms. The difference keeps decreasing.

Analyzing the nested loop over i, you are NOT performing around O(log (n - j)) iterations for each j. Rather, you are performing around O(log n - log j) = O(log $\frac{n}{j}$) iterations for each j. The total number of operations is around
$\log_3 n + \log_3 \frac{n}{2} + \log_3 \frac{n}{4} + \log_3 \frac{n}{6} + \log_3 \frac{n}{8} + ... + \log_3 (\frac{n}{n-4}) + \log_3 (\frac{n}{n-2})$.

Another view of the problem



$1 + \log_3 2 + \log_3 4 + \log_3 6 + \log_3 8 +$
$\log_3 10 + \log_3 12 + ... + \log_3 (n-2)$

$\log_3 n + \log_3 \frac{n}{2} + \log_3 \frac{n}{4} + \log_3 \frac{n}{6} + \log_3 \frac{n}{8}$
$+ ... + \log_3 (\frac{n}{n-4}) + \log_3 (\frac{n}{n-2})$

We can easily conclude that the number of operations on the left is O(n log n), from the area of the rectangle. However, we can't say the same for the diagram on the right. There is **no rectangle or triangle that has area of O(n log n)**. The number of i iterations decreases too quickly. So O(n log n) may not be a tight bound for the time complexity of this code.

I hope those of you who *got the answer correct* **really** know why the time complexity is tightly bounded as such.

So, what's the solution?

The expression $\log_3 n + \log_3 \frac{n}{2} + \log_3 \frac{n}{4} + \log_3 \frac{n}{6} + \log_3 \frac{n}{8} + ... + \log_3 (\frac{n}{n-4}) + \log_3 (\frac{n}{n-2})$ is difficult to simplify. We have to **transform the problem** in order to count the number of operations easily. Following the illustration above, we can **find the area under the graph row by row**, instead of column by column.

Notice, after 1/3 of all outer loop iterations, for each j, the inner loop performs 1 iteration. After 1/9 of the outer loop iterations, each j, the inner loop performs at most 2 iterations. After 1/27 of the outer loop iterations, each j, the inner loop performs at most 3 iterations.
...

The last iteration, the inner loop performs around $\log_3 n$ iterations.

Instead of summing up "how many inner loop iterations are there" for each j, we can **sum the number of times** the x^{th} **inner loop iteration ran**, for each x in 1 to around $\log_3 n$. We should ask "how many values of j are there, in which the inner loop performs at least x iterations" instead.

- For how many values of j does the inner loop iterate at least once? $\frac{1}{2}(n)$
- For how many values of j does the inner loop iterate at least twice? $\frac{1}{2}(\frac{n}{3})$
- For how many values of j does the inner loop iterate at least thrice? $\frac{1}{2}(\frac{n}{9})$
- For how many values of j does the inner loop iterate at least four times? $\frac{1}{2}(\frac{n}{27})$

  …
- For how many values of j does the inner loop iterate at least $\log_3 n + 1$ times? 1

The result is around $\frac{1}{2}(n + \frac{n}{3} + \frac{n}{9} + \frac{n}{27} + ...) = \frac{n}{2}(1 + \frac{1}{3} + \frac{1}{9} + \frac{1}{27} + ...) \approx \frac{n}{2}(\frac{1}{1-\frac{1}{3}}) = \frac{3n}{4}$, in O(n) time.

Q16. Answer and Comments

(Comments by Prof Tan)

```
guess ← 0
If (response == "try a bigger number") { // The number must be positive
    guess ← 1
    While (response == "Try a bigger number") {
        guess ← guess * 2   // make guesses 1, 2, 4, 8, 16, 32, …
    }
    // The above while loop takes O(log N)

    If (response == "You got it!") // End of guess
        Print guess; // Print the answer
    Else { // response == "Try a smaller number"
        // The number must be between guess/2 and guess
        // So use binary search to search between guess/2 and guess
        Print binarySearch(guess/2, guess) // this takes O(log N)
    }
}
Else { // The number must be negative
    Similar to above
}
```

Hence the overall complexity is O(log N).

The important things about this question are:

1. Must be able to handle both positive and negative numbers.
2. Time complexity must be O(log N).

Many students tried to find the upper and lower bound by guessing a random number (or MaxInt in some solution). This will cause the time complexity to be greater than O(log N). For example, if N = 16, log n is 4. If the first guess is 65536 (2^{16}), then your solution is actually O(N).

Q17. Answers and Comments

(Comments by Ivan)

(a)

0	1	2	3	4	5	6	7	8	9	10	11	12
36	14	6				32	19	46	45	71	23	

(b) **6, 7, 10, 2**

(c) **7, 8, 11, 3**

(i) *Probe sequence*

Probe sequence refers to the **index** of slots that are examined when finding a key. As quadratic probing is here used as the collision resolution method, the distance between two successive probes increases each time.

(ii) *Collision Resolution*

The output of the hash function is the home address. If the home address is occupied, there is a collision. Quadratic probing resolves the collision by attempting to insert the key in these indexes, in order:

home addr (home addr + 1) % 13 (**home addr** + 4) % 13 (**home addr** + 9) % 13 ...

A number of students added the distance of the next jump to the previous index probed. Some students added the distance to the home address, but % 12 (last index of the hash table) instead of 13. Both are incorrect.

Also, the collision resolution method affects the sequence in which we insert, delete, and search for keys. We can't insert keys using one collision resolution method and then search for keys using another.

Q18. Answers and Comments

(Comments by Prof Tan)

We are expecting pseudocode for this question.

(a)

```
(head, ListNode [] nodes)   // nodes is an array of 3 ListNode
```

There must be a way to return the references of the 3 parts after the partition as Java's parameters are **passed by value**. So, an array of 3 listNodes (or head + two listNodes, or any other data structure that stores the head nodes of the linkedlists) would be suitable.

(b)

```
parameter1:  nodes[0];
parameter2:  nodes[1];
parameter3:  nodes[2];
```

(c)

```
if (head == null) return null for all 3 array elements
Use the first node as pivot
ListNode curr = head;
while (curr != null) {
    if (curr.value < head.value) move node to the back of first list;
    else if (curr.value > head.value) move node to the back of second list;
    else move node to the last node of the middle list;
    curr = curr.next;
}
move first node to the front of middle list;
return;
```

The main mistakes made by students are:

- Most of the students forgot that Java's parameters are **passed by value** and so they did not cater for the references of the linkedlists to be sent back to the main quicksort() method.
- Did not ensure that the sorting algorithm is stable.
- Did not take care of the equal values which is part of the improvement we are looking for in this question.
- Many students just produced the Quick Sort algorithm on array.

Prepared: 8 May 2016