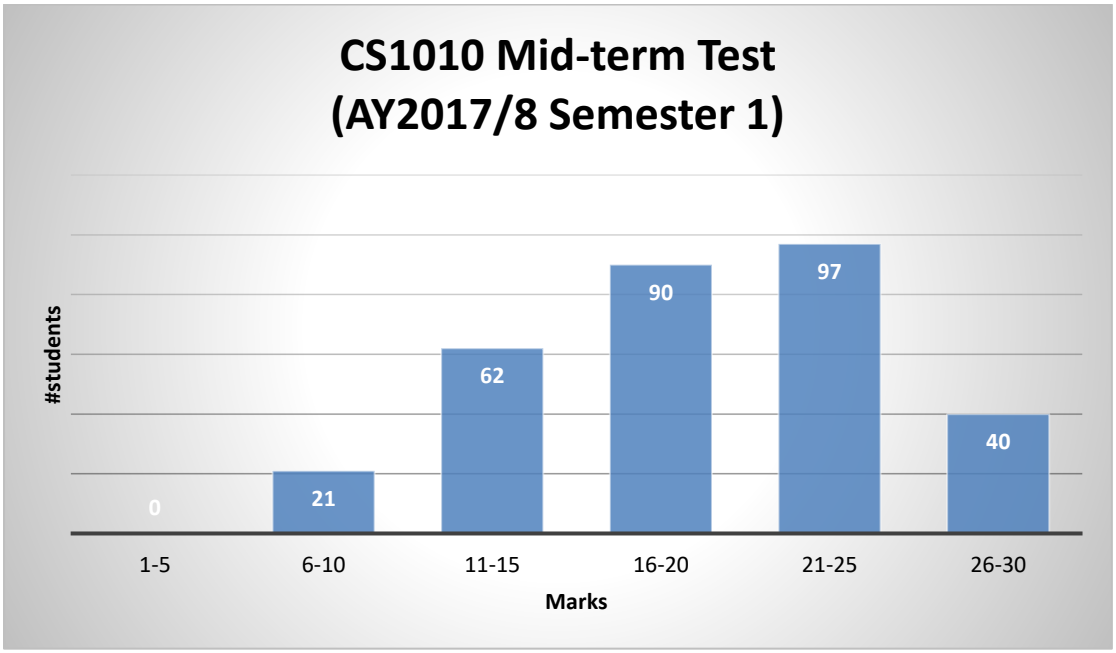
Comments on CS1010 Mid-term Test (AY2017/8 Semester 1)

1. Summary of results

310 students sat for the test on 6 October 2017. The mean is 19.24 out of 30 marks, or 64.1%. This mean is a normal one.

The distribution of the marks, in brackets of 5 marks, is shown in the chart below.



You may check your marks on the IVLE gradebook and report any error to me (tantc@comp.nus.edu.sg) within one week. We will return your answer sheets to you at your next discussion session.

2. Comments on MCQs

The table below shows the percentage of students who chose the correct answers, and of those who chose the most popular wrong answers:

	Q2	Q3	Q4	Q5
%students who chose the correct answer	D (77.7%)	C (60.6%)	B (51.9%) Hardest	E (61.0%)
%students who chose the most popular wrong answer	E (14.5%)	A (23.5%)	A (29.4%)	B (14.8%)

Q2: The first 3 passes of Bubble Sort algorithm on the given array is shown below.

Original array:	19	20	34	71	5	88	12	3
After first pass:	19	20	34	5	71	12	3	88
After second pass:	19	20	5	34	12	3	71	88
After third pass:	19	5	20	12	3	34	71	88

Q3: Option (i): Cannot be compiled, as the value in case must be a discrete constant value determined at compile time.

Option (ii): points incremented only once for all values of *situp*, due to the 'break' statements.

Option (iii): Without the 'break' statement, the fall-through effect increments variable *points* the correct number of times to get the right value.

Hence the answer is C.

Q4: Calling **binarySearch(values, 9, 1)** gives the following sequences of *left*, *left* and *mid*:

0, 8, 4 → 0, 4, 2 → 0, 2, 1 → 0, 1, 0 → key found at index 0

Calling **binarySearch(values, 9, 9)** gives the following sequences of *left*, *left* and *mid*:

0, 8, 4 → 4, 8, 6 → 6, 8, 7 → 7, 8, 7 → 7, 8, 7 → 7, 8, 7 → …

Hence the answer is B.

Q5: The function **mystery()** checks the elements in the lower triangular portion with their respective mirror images along the main diagonal. If any of elements in the lower triangular portion is smaller than its mirror image, the function returns 1, otherwise it returns 0.

For the matrices in options (ii) and (iii), every element in the lower triangular portion is larger than or equal to its mirror image. Hence the answer is E.

3. Comments on Questions 6 – 11.

Q6: Most students got this right. This is the best attempted question with an average mark of 1.83 (out of 2).

Q7. The average mark for this question is only 1.01 (out of 2). Many students were not aware that the pairs of elements are swapped twice, hence resulting in the original array.

Q8. The average mark for this question is 1.77 (out of 2).

First, we determine the return value of f2(100).

f2(100) → 50

Next, we determine the return value of f1(200, 50)

f1(200, 50):

f2(50) → 25 which is assigned to variable *a* in f1.

a + *b* → 25 + 50 → 75 is returned

The returned value 75 is assigned to the variable *a* in the main function.

The variable *b* in the main function remains unchanged.

Hence the output is: **75 200**.

Q9. This is the hardest question with an average mark of only 0.26 (out of 2). Many students got this question wrong.

Let's trace the first iteration of the outer loop i = 0. The inner loop variable *j* runs from 1 to 7:

j = 1: arr[0] = (arr[0] + arr[arr[1]])%8 → arr[0] = (1 + arr[2])%8 → arr[0] = (1 + 3)%8 → arr[0] = 4

j = 2: arr[0] = (arr[0] + arr[arr[2]])%8 → arr[0] = (4 + arr[3])%8 → arr[0] = (4 + 4)% 8 → arr[0] = 0

j = 3: arr[0] = (arr[0] + arr[arr[3]])%8 → arr[0] = (0 + arr[4])%8 → arr[0] = (0 + 5)% 8 → arr[0] = 5

j = 4: arr[0] = (arr[0] + arr[arr[4]])%8 → arr[0] = (5 + arr[5])%8 → arr[0] = (5 + 6)% 8 → arr[0] = 3

j = 5: arr[0] = (arr[0] + arr[arr[5]])%8 → arr[0] = (3 + arr[6])%8 → arr[0] = (3 + 7)% 8 → arr[0] = 2

j = 6: arr[0] = (arr[0] + arr[arr[6]])%8 → arr[0] = (2 + arr[7])%8 → arr[0] = (2 + 0)% 8 → arr[0] = 2

j = 7: arr[0] = (arr[0] + arr[arr[7]])%8 → arr[0] = (2 + arr[0])%8 → arr[0] = (2 + 2)% 8 → arr[0] = 4

Hence the final value of arr[0] is **4**. Do the same for the rest of the array.

The final result is: **4 4 1 5 0 2 3 0**.

One of the DLs has derived the short-cut for this question. You can try to figure it out – it might take a bit of time.

Q10. The given function **process_array(int arr[], int size)** basically takes an integer array *arr* with *size* elements and computes the cumulative sums.

For example, given this array with 6 elements:

5 3 -2 6 -4 3

The resulting array is:

5 8 6 12 8 11

where arr[1] is the sum of the first 2 original values; arr[2] is the sum of the first 3 original values, etc.

The given function re-computes the cumulative sums by repeating the computation in the inner for loop. This is a bad algorithm. The function could be improved by removing the inner for loop, by just adding up the elements progressively from left to right:

```
// Precond: size >=0
void process_array(int arr[], int size) {
   int i;

   for (i=1; i<size; i++) {
      arr[i] = arr[i] + arr[i-1];
   }
}
```

The average mark is 1.17 (out of 4). Some students simply removed the variable *sum*:

```
// Precond: size >=0
void process_array(int arr[], int size) {
   int i, j;

   for (i=size-1; i>=0; i--) {
      for (j=0; j<i; j++) {
         arr[i] = arr[i] + arr[j];
      }
   }
}
```

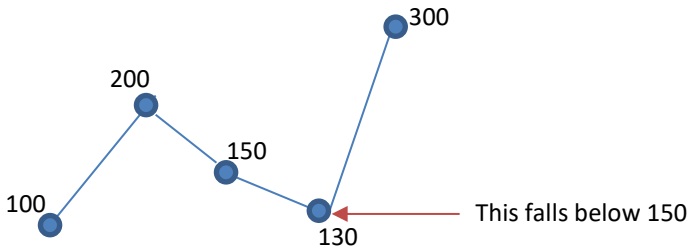
However, this is basically the same nested-loop algorithm. Note that the condition for the inner loop must be changed “j < i” for it to work. Some students wrote “j <= i" which that is wrong, as the value of arr[i] will be added twice.

Q11. The average mark for part (a) is 2.55 (out of 3) and for part (b) is 2.87 (out of 5).

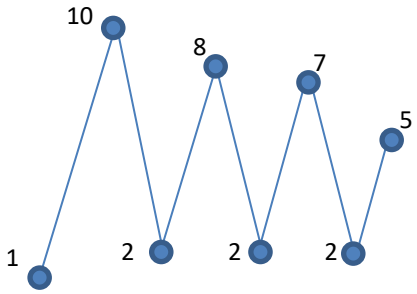
Part (a) is to test students on understanding problem statement. Most students got the first two answers right, but some got the third wrong.

A good way is to visualise the data.

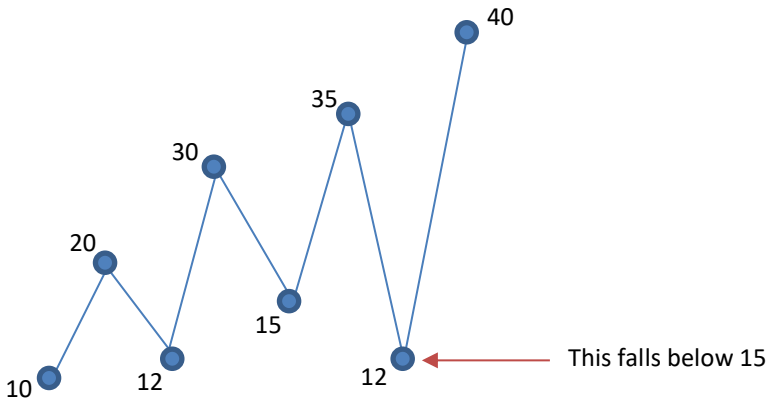
(i) int prices[] = { 100, 200, 150, 130, 300 }; → Answer: **False** (not favourable)



(ii) int prices[] = { 1, 10, 2, 8 , 2, 7, 2, 5 }; → Answer: **True** (favourable)



(iii) int prices[] = { 10, 20, 12, 30, 15, 35, 12, 40 }; → Answer: **False** (not favourable)



After going through part (a), you will come to realise that the trigger point is when you get a "down-slope" (a price fall), you check if the new price is below the current low limit – if it is, right way conclude that the stock is unfavourable, and if it is not, that this new price will be our new low limit from now on.

Some students somehow concluded that an unfavourable stock must have two consecutive price falls, as in (i), but it can be seen from (iii) that this is not true.

Other common mistakes include missing out check the last value, and also missing out checking that case where the second value is smaller than the first.

Some students assume that all prices are non-negative. This assumption is not necessary.

The code is rather short. Two versions are shown below.

```
// Precond: size > 0
int favourable(int arr[], int size) {
   int i, low = arr[0];

   for (i=1; i<size; i++) {
      if (arr[i] < low) {
         return 0;
      }
      if (arr[i] < arr[i-1]) {
         low = arr[i];
      }
   }
   return 1;
}
```

```
// Precond: size > 0
int favourable(int arr[], int size) {
   int i, fall_index = 0;

   for (i=1; i<size; i++) {
      if (arr[i] < arr[fall_index]) {
         return 0;
      }
      if (arr[i] < arr[i-1]) {
         fall_index = i;
      }
   }
   return 1;
}
```

Aaron Tan
9 October 2017