Hong Kong Olympiad in Informatics 2019/20
Heat Event (Senior Group)
Official Solution

Statistics (N = 203)

Full mark = 42. Maximum = 40. Median = 12.5. Advance to Final = 13.5 marks or above.

Section A

Q	A	Explanation
1	D	Consider a sequence with '-' and '/' to represent the modification of n in the while loop : (e.g. : 50 = "/-///-/"). Note that '-' operations must come along with '/' operations and the last operation must be '/'. So, there should be 4C3 + 5C2 = 14 combinations with (4'/' 3'-') and (5'/' 2'-'). But as 56("//-//-/") and 52("///-/-/") are out of range, so the final answer is 14 - 2 = 12.
2	B	Trace the program carefully. fact(10, 4) = 10 * fact(6, 3) = 10 * 6 * fact(3, 2) = 10 * 6 * 3 * fact(1, 1) = 10 * 6 * 3 * 1 * fact(0, 0) = 10 * 6 * 3 * 1 * 1 = 180
3	C	Quick sort works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain. Merge sort first divides the unsorted list into n sublists, each containing one element (a list of one element is considered sorted).Then repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list. So only quick sort and merge sort apply divide and conquer.

4	C	Both function `f` and `g` would return a value when `n = 0` and stop recurring, because of short-circuit evaluation -
		• when first argument of the AND function evaluates to false, the overall value must be false.
		• and when the first argument of the OR function evaluates to true, the overall value must be true.
		`f(0) = true`, `g(0) = false`, and for n > 0, `f(n) = g(n - 1)`, and `g(n) = f(n - 1)`.
		`f(72) = g(71) = f(70) = ... = g(1) = f(0) = true`
		`g(107) = f(106) = g(105) = ... = g(1) = f(0) = true`

5 C

Consider the cases when Tom stands at the leftmost end of the line, as boys and girls must stand alternatively, the arrangement is as follows (T = Tom, G = Girl, B = Boy): TGBGBGBGBGBG, which is 5! * 6! = 86400.

When Tom stands at the rightmost end of the line, the arrangements are the reversed version of Tom standing at the leftmost end of the line. And thus, the answer is 86400*2 = 172800.

6 D

Trace the program carefully:

i	q[0]	q[1]	q[2]	q[3]
push(1)	1	0	0	0
push(4)	1	4	0	0
push(3)	1	4	3	0
push(2)	1	4	3	2

ii	q[0]	q[1]	q[2]	q[3]
push(4)	4	0	0	0
push(3)	3	4	0	0
push(2)	2	4	3	0
push(1)	1	4	3	2

iii	q[0]	q[1]	q[2]	q[3]
push(4)	4	0	0	0
push(2)	2	4	0	0
push(3)	2	4	3	0
push(1)	1	4	3	2

Since the values of q are the same for all i, ii and iii after push, the output will also be the same. The output is 1234 for options i, ii and iii.

7 D Consider the truth tables below:

(A OR B) OR (A XOR B)		
	A = false	A = true
B = false	false	true
B = true	true	true
(A OR B) XOR (A XOR B)		
	A = false	A = true
B = false	false	false
B = true	false	true
(A OR B) OR (A AND B)		
	A = false	A = true
B = false	false	true
B = true	true	true
(A OR B) XOR (A NOR B)		
	A = false	A = true
B = false	true	true
B = true	true	true

`A` $\otimes$ `B` is equivalent to `(A OR B) XOR (A NOR B)`.

8	D	A pair of Boolean expression is logically equivalent if they have the same truth table. The truth tables are as follows:

`((NOT a) AND b) OR (a AND (NOT b))`		
	a = false	a = true
b = false	false	true
b = true	true	false
`NOT (a = b)`		
	a = false	a = true
b = false	false	true
b = true	true	false
`NOT ((NOT a) = (NOT b))`		
	a = false	a = true
b = false	false	true
b = true	true	false

From the truth tables, all three Boolean expressions are logically equivalent.

9	B	Bun can always set the counter to a multiple of 8 after his round(regardless of Apple's choice). After reaching 992(multiple of 8), the counter must lie within 993-999 after Apple's round. Consequently, Bun can win the game.
10	A	i. Bun can only use the strategy above if and only if the initial value of the counter is a multiple of 8, otherwise Apple can use the strategy above instead. ii. Each time Apple used 0, Bun can use 0 in the next round to keep the counter to a multiple of 8. iii. Using the strategy above, for each number `x` in 1 to 7, the count of number `x` used by Apple must be the same as the count of number `(8-x)` used by Bun after each of Bun's rounds, so Bun can always keep the counter to a multiple of 8. So, the answer is i only.
11	D	The program outputs the number of 1s in the binary notation of `x`. Since $79622_{10} = 10011011100000110_2$, the answer is 8.
12	B	The program performs bubble sort on the odd index elements and even index elements of array `a` respectively. So only ii and iii must be true.

13 D

i. 65535 mod 3 = 0. Note that the range starts from 0, so the number of values % 3 that return 0 is 65535/3+1 = 21846. While that of 1 and 2 are 21845. The chance of returning 0 is higher than that of 1 and 2.

ii. (r()+r()+r()) mod 3 = (r() mod 3 + r() mod 3 + r() mod 3) mod 3.
From (i), we know that the probability function of r() mod 3 is not equally distributed. In fact, the probabilities of getting 0, 1, 2 are as follows,
P(0) = 21846 / 65535, P(1) = 21845 / 65535, P(2) = 21845 / 65535

For (r() mod 3 + r() mod 3 + r() mod 3) mod 3, the probabilities of getting 0, 1, 2 can be obtained by using the results above.

- P'(0) = P(0)P(0)P(0) + P(0)P(1)P(2) + P(0)P(2)P(1) + P(1)P(0)P(2) + P(1)P(1)P(1) + P(1)P(2)P(0) + P(2)P(0)P(1) + P(2)P(1)P(0) + P(2)P(2)P(2)
- P'(1) = P(0)P(0)P(1) + P(0)P(1)P(0) + P(0)P(2)P(2) + P(1)P(0)P(0) + P(1)P(1)P(2) + P(1)P(2)P(1) +P(2)P(0)P(2) + P(2)P(1)P(1) + P(2)P(2)P(0)
- P'(2) = P(0)P(0)P(2) + P(0)P(2)P(0) + P(0)P(1)P(1) + P(1)P(0)P(1)+ P(1)P(1)P(0) + P(1)P(2)P(2) + P(2)P(0)P(0) + P(2)P(1)P(2) + P(2)P(2)P(1)

Suppose P(0) = a, P(1) = P(2) = b,
$P'(0) = a^3 + 6ab^2 + 2b^3$, $P'(1) = P'(2) = 3a^2b + 3ab^2 + 3b^3$,
P'(0) > P'(1) = P'(2)
So, the chance of returning 0 is higher than that of 1 and 2.
Alternatively, considering r() that return an integer between 0 and 4 inclusively with equal probability would provide insights for finding that the chance of returning 0 is higher.

14 D

The possible range of (myrand(50) - 30) is [-30,19]. But after (mod 5), the range will become [-4,4], So the answer is 9.

15 B

Values of `i, x, y` after the `i`th iteration:

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
x	0	4	4	2	2	3	3	1	1	**5**
y	0	0	1	1	0	0	3	3	1	1

16 C

The push function pushes an element into the queue. The pop function outputs the first element in the queue and pops it. The queue size is 3. After the first 3 push, tail = head so the first pop outputs "Empty". Queue is a First-In-First-Out data structure, so the remaining outputs are "4", "8", "Empty".

17 A Calculate the number of different paths for every cell.

	1	2	3	4	5	6	7	8
1	1(A)	1	1	1	1	1	1	1
2	1	2	3	4	1	2	3	4
3	1	3	6	10	1	3	6	10
4	1	4	10	20	20	3	9	19
5	1	1	1	20	40	40	40	59
6	1	2	3	3	40	80	120	179
7	1	3	6	9	40	120	240	419
8	1	4	10	19	59	179	419	838(B)

There are 838 different paths.

18 A When `a[i]≠0`, `x=a[i]`, so `a[j] mod x` must be 0 when `j=i`. This sets `flag` to true and increases `res`. By tracing the program, it can be found that these values are not set to 0:

`a[0]` = 2, which sets 6(`a[2]`), 18(`a[5]`) and 50(`a[9]`) to 0

`a[1]` = 5, which sets 15(`a[4]`), 35(`a[7]`), and 45(`a[8]`) to 0

`a[3]` = 9

`a[6]` = 21

Alternatively, one may observe that the program outputs the number of elements in `a` that is not a multiple of any element before it. Only `2,5,9,21` meet this criterion, so `res = 4`.

19 B The possible scores of each round:

Round	Score
0	0:0
1	1:0 or 0:1
2	2:0 or 0:2
3	2:1 or 1:2 or 3:0 or 0:3
4	3:1 or 1:3
5	3:2 or 2:3
6	3:3

For round 1 and round 3, the probability of getting the score listed on the table from the previous round is 1. For the remaining rounds, the probability is 1 / 2. So, the final answer is $(1 / 2)^4 = 1 / 16$.

20 D The expected time require for each strategy:

	Expected Time
Strategy A	10*0.1 + 20*0.2 + 35*0.2 + 75*0.5 = 49.5
Strategy B	40*0.5 + 55*0.2 + 65*0.2 + 75*0.1 = 51.5
Strategy C	10*0.1 + 50*0.5 + 65*0.2 + 75*0.2 = 54
Strategy D	10*0.2 + 25*0.2 + 65*0.5 + 75*0.1 = 47

As strategy D has the minimum expected time among all strategies, it is the answer.

21 C Note that the leftmost 3 vertices are interconnected, so their colours have to be different. Hence, there are 3! = 6 ways to fill in the left part.

Denote the central vertex as colour 1. The rightmost 3 vertices (from up to down) can be 2 1 2, 2 1 3, 2 3 2, 3 1 2, 3 1 3 and 3 2 3. Hence, there are 6 ways to fill in the right part.

Total number of combinations = 6 * 6 = 36

22 C The program outputs the number of 1s in the binary notation of `i`. Since $153_{10} = 10011001_2$, the answer is 4.

23 A Trace the program carefully:

	Each value in array b
i = 0	3 1 2 3 4 5 6 7 8 9
i = 1	3 1 2 3 4 5 6 7 8 9
i = 2	3 1 4 3 4 5 6 7 8 9
i = 3	3 1 4 1 4 5 6 7 8 9
i = 4	3 1 4 1 4 5 6 7 8 9
i = 5	3 1 4 1 4 7 6 7 8 9
i = 6	3 1 4 1 4 7 1 7 8 9
i = 7	3 1 4 1 4 7 1 8 8 9
i = 8	3 1 4 1 4 7 1 8 1 9
i = 9	3 1 4 1 4 7 1 8 1 1

As `f(x)` will return `x` when `b[x] = x` or return `f(b[x])` in other cases,
`f(b[4]) = 4`,
`f(b[8]) = f(b[1]) = 1`,
So the answer is 4 1.

24 A f(n) is the Euler's phi function. The only positive integers < pq that are not coprime with `pq` are kp for k = 1, 2, …, p - 1, and kq for k = 1, 2, …, q - 1, altogether p + q - 2 numbers. The list is exhaustive because the integers required should have common divisors (besides 1) with pq, which that common divisor can only be either p or q since p and q are distinct primes. Also, no integer exists in both lists because the smallest integer that is multiples of p and q is pq.
There are a total of pq - 1 positive integers smaller than pq, subtract the number of integers on the list above to get the number of coprime required.
f(pq) = (pq - 1) - (p + q - 2) = pq - p - q + 1 = (p - 1)(q - 1)

25 B The program finds the prime numbers between 2 and 10. If i is a prime number, a[i] = 0. Otherwise, a[i] = 1. The array p stores the prime numbers. There are 4 prime numbers between 2 and 10 so k = 4. Note that 4 is not a prime number so a[k] = 1.

Section B

<table>
<tr><th colspan="4">Answer and Explanation</th></tr>
<tr><td></td><td>Pascal</td><td>C</td><td>C++</td></tr>
<tr><td rowspan="2">A</td><td>(s[n]='0')and((n=1)or(s[n-1]='0'))</td><td colspan="2">s[n-1]=='0'&&(n==1||s[n-2]=='0')</td></tr>
<tr><td colspan="3">Please be noted that 0 is also a multiple of 100. So, checking if the integer is 0 or the last two digits of the integer are both 0 would be correct.</td></tr>
<tr><td rowspan="2">B</td><td>b[i]:=b[i]+b[i-1]</td><td>b[i]=b[i]+b[i-1]</td><td>b[i]=b[i]+b[i-1]</td></tr>
<tr><td colspan="3">Observe that b[1] = a[1], b[2] = a[2] - a[1], b[3] = a[3] - a[2]...
So adding b[i-1] to each b[i] from i = 1 to i = 8 will turn every b[i] to be a[i] again.</td></tr>
<tr><td rowspan="2">C</td><td>b[9-i]:=b[9-i]-b[8-i]</td><td>b[9-i]=b[9-i]-b[8-i]</td><td>b[9-i]=b[9-i]-b[8-i]</td></tr>
<tr><td colspan="3">As b[i] stores the value from a[1] to a[i], b[i] - b[i-1] would be a[i].
But starting the process from b[1] would not work as b[i-1] in b[i] - b[i-1] has been modified for other i.
Reversing the order of the process by starting from b[8] could prevent the problem.</td></tr>
<tr><td>D1</td><td colspan="3" rowspan="4">This question is cancelled.</td></tr>
<tr><td>D2</td></tr>
<tr><td>D3</td></tr>
<tr><td>E</td></tr>
<tr><td rowspan="2">F</td><td>3</td><td>3</td><td>3</td></tr>
<tr><td colspan="3">f(x) returns the number of factors x. The factors of 121 are 1,11 and 121, there are a total of 3 factors.</td></tr>
<tr><td rowspan="2">G</td><td>25</td><td>25</td><td>25</td></tr>
<tr><td colspan="3">The program outputs the number of integers between 1 and 10000 having 3 factors. The integers having 3 factors must be square of prime numbers, so the program outputs the number of prime numbers between 1 and 100, which is 25.</td></tr>
<tr><td>H1</td><td>n+1</td><td>n+1</td><td>n+1</td></tr>
<tr><td rowspan="2">H2</td><td>f(n-1) xor n</td><td>f(n-1)^n</td><td>f(n-1)^n</td></tr>
<tr><td colspan="3">By tracing g(n), the following is observed:
when n mod 4 = 0, g(n) = n
when n mod 4 = 1, g(n) = 1
when n mod 4 = 2, g(n) = n+1
when n mod 4 = 3, g(n) = 0
For H1, when n mod 4 = 2, returning n+1 will be same as g(n).
For H2, for other cases, as g(n) = g(n-1)^n, we can just simply return f(n-1)^n as the base case is defined.</td></tr>
</table>

<table>
<tr><td rowspan="2">I1</td><td>((a<=b)and(b<=c))or((c<=b)and(b<=a))</td><td>((a<=b&&b<=c)||(c<=b&&b<=a))</td><td>((a<=b&&b<=c)||(c<=b&&b<=a))</td></tr>
<tr><td>median(b,c,a) // median(c,a,b)</td><td>median(b,c,a) // median(c,a,b)</td><td>median(b,c,a) // median(c,a,b)</td></tr>
<tr><td rowspan="2">I2</td><td colspan="3">For I1, we just need to check whether b is in the middle of the three numbers.
We know that the function can return the median if the second parameter is the median.
For I2, what we need to do is to check a or c is the median. So, we can shuffle the numbers and call the function itself to check which one is the median. Remember to ensure that both a and c can be the second parameter in the future calls.</td></tr>
<tr><td>J</td><td>f(a-m)+f(b-m)+f(c-m)</td><td>f(a-m)+f(b-m)+f(c-m)</td><td>f(a-m)+f(b-m)+f(c-m)</td></tr>
<tr><td colspan="4">f(x) returns the absolute value of x. f(x-m) calculates the difference between x and m. One of f(a-m), f(b-m), f(c-m) is equal to 0 and the other two's sum will be equal to the range.</td></tr>
<tr><td rowspan="2">K</td><td>-480</td><td>-480</td><td>-480</td></tr>
<tr><td colspan="3">The error of the program is that it would treat the "+" sign as a digit precedent to the next number.
Given that you didn't memorize the ASCII code of "+", the digit that it represents can be figured out from the given example 100 + 1 = 51. Suppose x is the value "+" represents, solving 100 + 10x + 1 = 51, we will get x = -5.
The given expression 10 + 10 would evaluate to be 10 + -5 * 100 + 10 = -480</td></tr>
<tr><td>L1</td><td>26</td><td>56</td><td>85</td></tr>
<tr><td rowspan="2">L2</td><td>inc(i) end; //continue end;</td><td>i++; } // continue; }</td><td>i++; } // continue; }</td></tr>
<tr><td colspan="3">To fix the bug of the program, the program would simply need to go onto the next iteration whenever a "+" sign is met. This can be done by increasing the pointer (which is i) by 1, or using continue to break out of the current iteration and continue with the next.</td></tr>
</table>