Hong Kong Olympiad in Informatics 2021/22
Heat Event (Junior Group)
Official Solution

Statistics (N = 290)
Full mark = 45. Maximum = 40. Median = 11. Advance to Final = 14.5 marks or above.

Section A

<table>
<tr><th>Q</th><th>A</th><th>Explanation</th></tr>
<tr><td>1</td><td>F</td><td>Assume 60% of the doctors are smart people, and the 60% within are rich, then only 60%*60%=36% of the doctors are rich, which is less than 50% while satisfying the conditions. Thus the statement is false.</td></tr>
<tr><td>2</td><td>F</td><td>The first program prints “HKOI” for <code>a[0]*a[1]*a[2]*a[3]</code> times, but the second does <code>a[0]+a[1]+a[2]+a[3]</code> times, which may not be equal. The statement is false.</td></tr>
<tr><td>3</td><td>T</td><td>Before any handshake happens, the statement is true.
When a handshake happens, the parity in the number of hands shaken for the participants involved flips.
<table>
<tr><th colspan="2">Before</th><th colspan="2">After</th><th rowspan="2">Is the statement still true?</th></tr>
<tr><th>Parity of participant A</th><th>Parity of participant B</th><th>Parity of participant A</th><th>Parity of participant B</th></tr>
<tr><td>Odd</td><td>Odd</td><td>Even</td><td>Even</td><td>Yes</td></tr>
<tr><td>Odd</td><td>Even</td><td>Even</td><td>Odd</td><td>Yes</td></tr>
<tr><td>Even</td><td>Odd</td><td>Odd</td><td>Even</td><td>Yes</td></tr>
<tr><td>Even</td><td>Even</td><td>Odd</td><td>Odd</td><td>Yes</td></tr>
</table>
The statement is always true.</td></tr>
<tr><td>4</td><td>F</td><td>When the element to be determined is at the beginning of the array, linear search can operate with <code>1</code> comparison, which would be fewer than what binary search needs.</td></tr>
<tr><td>5</td><td>F</td><td>Calling <code>A()</code> and <code>B()</code> two times may not return the same result, it is possible for <code>A() > B()</code> and <code>A() <= B()</code> and therefore the statement is false.</td></tr>
<tr><td>6</td><td>A</td><td>2^{14} is the first power of 2 that is greater than 15000, therefore at least 2 bytes (16 bits) are required to represent all the citizen ID.</td></tr>
<tr><td>7</td><td>A</td><td><code>x</code> counts the sum of numbers divisible by 10 in range [1, 2022], in which the answer is 202.
<code>y</code> counts the sum of numbers divisible by 5 but not 10 in range [1, 2022], which is 202 as well.
<code>z</code> counts the rest of numbers, which is 2022-202-202=1618</td></tr>
<tr><td>8</td><td>B</td><td><code>d</code> is the smallest integer <code>n</code> such that <code>n(up-down) + down >= 150</code>. From that, it can be easily calculated that d is minimum for choice B.</td></tr>
<tr><td>9</td><td>A</td><td>Option B and C are subsets of option A i.e. if they are true, then option A is also true. Therefore option A are more likely to be true compared to option B and C.</td></tr>
<tr><td>10</td><td>A</td><td><code>P AND (NOT (Q AND R))</code>
<code>≡ P AND ((NOT Q) OR (NOT R))</code> (by De Morgan’s Law)
Expression i is equivalent..
<code>P AND (NOT (Q AND R))</code>
<code>≡ NOT((NOT P) OR (Q AND R))</code> (by De Morgan’s Law)
Expression ii is not equivalent.</td></tr>
</table>

Q	A	Explanation

11	B	For `x = 12`,
		`((x / 10 = 2) and (x > 15) or (x mod 3 = 0))`
		`=> ((1 = 2) and (12 > 15) or (0 = 0))`
		`=> (false and false or true)`
		`=> (false or true)`
		`=> true`
		`((x / 5 = 2) or (x < 13) and (x mod 5 = 1))`
		`=> ((2 = 2) or (12 < 13) and (2 = 1))`
		`=> (true or true and false)`
		`=> (true || false)`
		`=> true`
		Therefore the output is `YES YES`

12	D	Assume team 1 won 4 matches, it has to win over every other teams. Thus it is impossible for team 2 to win over team 1 and the other teams to get another 4 points.
13	C	Sequence of operation for the required output: {1, 1, 1, 1, 2, 2, 1, 1, 2, 2, 2, 1, 2, 2}
14	A	Follow the trace table: `i` | 1 | 2 | 3 `a` after operation | `{1, 1, 1, 1, 5, 2, 2, 6}` | `{1, 1, 1, 1, 2, 2, 2, 6}` | `{1, 1, 1, 1, 2, 2, 2, 6}` The array remains the same for the rest of the operations, thus the answer is `1 2`
15	B	The number of inversions in the sequence is 5, at least 5 swaps are required.
16	B	Follow the trace table. `i` | 1 | 2 | 3 | 4 `x % 4` | 1 | 3 | 3 | 3 `y % 4` | 0 | 0 | 0 | 0 `x` afterward | 103 | 107 | 111 | 115 `y` afterward | 104 | 108 | 112 | 116 Value of x and y can then be deduced, which is 103+99*4=499 and 100+100*4=500 respectively.
17	D	`a[3] = 2`: false `a[6] mod 2 = 1`: true `a[4] = a[7]`: false Thus x = 2 + 4 = 6
18	D	All possible ways: 12123, 12132, 12312, 12313, 12323, 13123, 13132, 13213, 13212, 13232 Fill in 1, 2 and 3 with R, B and B, there is 3 ways to do so. Thus the number of ways is 10*3=30

19	C	ii: `(a xor b) = 0`
		`≡ (a xor b) xor b = 0 xor b`
		`≡ a = b`
		iii: when `a != b`, there must be bit differences between them.
		On those bits, their or value would be 1 while their and value is 0, which would return false on the comparison, equivalent to i.
		iv: When `b = a + 1, (2a + 1) / 2 = a` would be true while `a = b` is false.

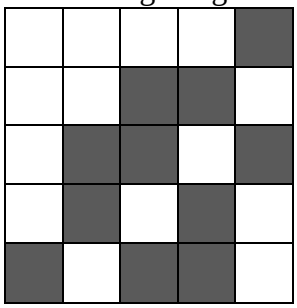
Q	A	Explanation
20	C	The problem can be converted into "number of ways to tile 1x10 grid with 1x1 and 1x2 tiles", and can be found using dynamic programming: `f[n] = f[n - 1] + f[n - 2]`, with base case `f[0] = 1` and `f[1] = 1`.

i	0	1	2	3	4	5	6	7	8	9	10
f[i]	1	1	2	3	5	8	13	21	34	55	89

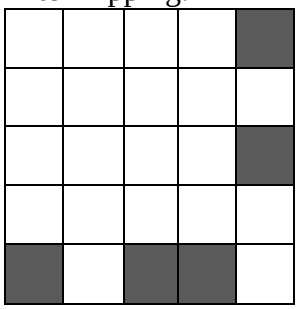
In fact, the answer is the 11^{th} Fibonacci number.

21 B Observe that if a grid can be produced, they can both achieve the same sets of grids. Using this, we can check whether 2 grids are in the same set of configuration by achieving grids that are easy to produce, in which we may choose to flip the leftmost and uppermost 4x4 grids into white cells.

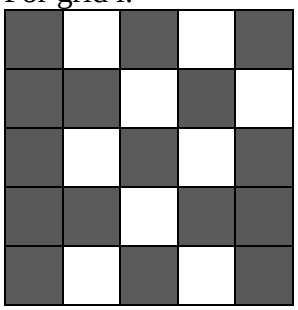
For the original grid:



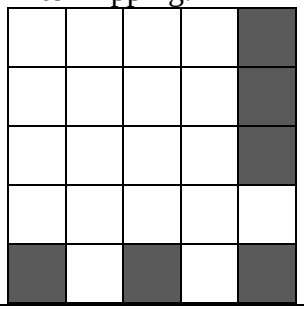
After flipping:

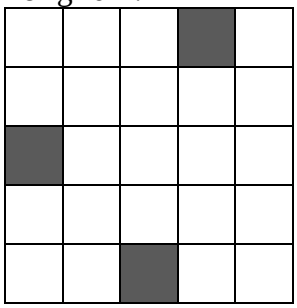
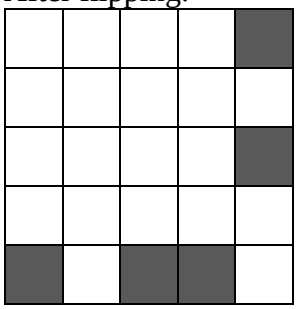


For grid i:



After flipping:



Q	A	Explanation
		For grid ii:  After flipping:  Only grid ii can reach the same configuration as the original grid, the answer is B.

22	A	Rephrase the for loops into
		`for i := 0 to 9 do`
		`  for j := i to 7 do`
		`  for k := j to 7 do`
		`  cnt := cnt + 1`
		The answer is 8 + 2(7) + 3(6) + 4(5) +5(4) + 6(3) + 7(2) + 8(1) = 120

23	B	Through enumeration we can easily find that {1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60} are products from 1, 2, 3, 4, 5 that are within [1, 100]. However for {4, 12, 20, 60}, if we input them into the program they will be divided by 2 instead of 4, causing the output to be 2. The answer is therefore 11.
24	D	Answer = 3(picking 2) + 10(picking 3) + 12(picking 4) + 6(picking 5) + 1(picking 6) = 32
25	B	P(Bob winning) = P(Alice lost on third step) / 3 + P(Alice lost on second step) / (3*4) + P(Alice lost on first step) / (3*4*4) = (3/64) / 3 + (3/16) / 12 + (3/4) / 48 = 3/64

Section B

<table>
<tr><th colspan="4">Answer and Explanation</th></tr>
<tr><th></th><th>Pascal</th><th>C</th><th>C++</th></tr>
<tr><td>A</td><td><code>(a[i]+9)mod 26</code></td><td colspan="2"><code>(a[i]+9)%26 // a[i]+i%2*26-17 // a[i]%17+i%2*9</code></td></tr>
<tr><td></td><td colspan="3">If we map A ~ Z to 0 ~ 25, to turn “RIVER” into “ARENA”, we need to go through: 17 → 0, 8 → 17, 21 → 4, 4 → 13. The new number is always congruent to the original number plus 9 (modulo 26).</td></tr>
<tr><td>B1</td><td><code>i:=n-1 downto 0 do</code></td><td colspan="2"><code>i = n - 1; i >= 0; i--</code></td></tr>
<tr><td></td><td colspan="3">The coins are sorted in ascending order and we want to use the coins with the larger values first, so we should loop from <code>n - 1</code> to <code>0</code></td></tr>
<tr><td>B2</td><td colspan="3"><code>m >= c[i]</code></td></tr>
<tr><td>B3</td><td colspan="3"><code>m - c[i]</code></td></tr>
<tr><td></td><td colspan="3">We store the remaining money with <code>m</code>, and each time we deduct the coin value from <code>m</code> so that we can keep track of how much money we still have to pay.</td></tr>
<tr><td>C1/
C2</td><td colspan="3"><code>(3, 4) // (5, 6) // (5, 7) // (5, 8)</code></td></tr>
<tr><td></td><td colspan="3">When <code>c[1]</code> = 3 and <code>c[2]</code> = 4, Alice’s algorithm uses 4 coins but there exists a construction using only 3 coins (3 cents * 2 + 4 cents * 2)
When <code>c[1]</code> = 5 and <code>c[2]</code> = 6, Alice’s algorithm uses 5 coins but there exists a construction using only 2 coins (5 cents * 2).
When <code>c[1]</code> = 5 and <code>c[2]</code> = 7, Alice’s algorithm uses 4 coins but there exists a construction using only 2 coins (5 cents * 2).
When <code>c[1]</code> = 5 and <code>c[2]</code> = 8, Alice’s algorithm uses 3 coins but there exists a construction using only 2 coins (5 cents * 2).</td></tr>
<tr><td>D</td><td colspan="3"><code>2, 4</code></td></tr>
<tr><td></td><td colspan="3">Counter examples where Alice’s greedy algorithm fail in systems 1, 3, and 5:
System 1. m = 24, her algorithm uses 5 coins, optimal solution 3 coins (8 cents * 3).
System 3. m = 16, her algorithm uses 3 coins, optimal solution 2 coins (8 cents * 2).
System 5. m = 80, her algorithm uses 10 coins, optimal solution 2 coins (40 cents * 2).</td></tr>
<tr><td>E</td><td colspan="3"><code>x*m+y*n-2*x*y</code></td></tr>
<tr><td></td><td colspan="3">A bulb is related to the button of its row and the button of its column, and will be turned on if exactly 1 of the 2 buttons are pressed and turned off if neither or both buttons are pressed. There are m cells in a row and x rows with their buttons pressed, so there are x * m lightbulbs with the button of its row pressed. Also, x * y cells have both buttons pressed, so there are x * m - x * y light bulbs with only the button of its row pressed but not the button of its column. Similarly, there are exactly y * n - x * y light bulbs with only the button of its column pressed but not the button of its row. So in total, (x * m - x * y) + (y * n - x * y) = x * m + y * n - 2 * x * y light bulbs are ultimately on.</td></tr>
</table>

<table>
<tr><th colspan="4">Answer and Explanation</th></tr>
<tr><td>F1</td><td colspan="3">abs(a[j]-i) // abs(i-a[j])</td></tr>
<tr><td>F2</td><td colspan="3">temp</td></tr>
<tr><td rowspan="2">F3</td><td colspan="3">i</td></tr>
<tr><td colspan="3">We consider each possible value of x one by one using i. For each i, we compute the sum of absolute differences and save it in variable temp. If temp is smaller than minsum, which stores the minimum so far, we update minsum to be equal to temp and x to be equal to i.</td></tr>
<tr><td rowspan="2">G</td><td colspan="3">a[200]</td></tr>
<tr><td colspan="3">Considering a[200 - i] and a[200 + i] for i from 1 to 200, notice that if a[200 - i] <= x <= a[200 + i], then abs(a[200 - i] - x) + abs(a[200 + i] - x) = x - a[200 - i] + a[200 + i] - x = a[200 + i] - a[200 - i], otherwise, abs(a[200 - i] - x) + abs(a[200 + i] - x) > a[200 + i] - a[200 - i], and considering a[200], we have abs(a[200] - x) >= 0. Thus, the minimum sum is at least the sum of (a[200 + i] - a[200 - i]) for i from 1 to 200, which is achieved when x = a[200]. If x is not a[200], then abs(a[200] - x) cannot be 0, so the sum cannot be minimum.</td></tr>
<tr><td rowspan="2">H</td><td colspan="3">47</td></tr>
<tr><td colspan="3">The program will output ‘HKOI’ if the xor sum of 10 integers in the array is 118. Note that x xor x is always identical to 0 and x xor 0 is always identical to x. Thus, a[9] = 118 xor 118 xor a[9] = (a[0] xor a[1] xor … xor a[9]) xor 118 xor a[9] = (a[0] xor a[1] xor … xor a[8]) xor 118 = 1 xor 2 xor 12 xor 0 xor 58 xor 74 xor 64 xor 92 xor 58 xor 118 = 47.</td></tr>
<tr><td rowspan="2">I</td><td>22, end;dec(i); //
24, ;dec(i);end;</td><td>53, }i--; //
54, n--;i--; //
55, i--;} //
55, }else //
56, else i++;
56, else{i++};</td><td>86, }i--; //
87, n--;i--; //
88, i--;} //
88, }else //
89, else i++;
89, else{i++};</td></tr>
<tr><td colspan="3">The value of i is sometimes incorrect.</td></tr>
<tr><td rowspan="2">J</td><td>(ay=by)and((ax1<=bx1)and(ax2>=bx1)or(bx1<=ax1)and(bx2>=ax1)) //
(ay = by) and (ax1 <= bx2) and (bx1 <= ax2) // (ay = by) and not ((ax1 > bx2) or (bx1 > ax2))</td><td colspan="2">ay == by && (ax1 <= bx1 && ax2 >= bx1 || bx1 <= ax1 && bx2 >= ax1) //
ay == by && ax1 <= bx2 && bx1 <= ax2 //
ay == by && !(ax1 > bx2 || bx1 > ax2)</td></tr>
<tr><td colspan="3">If ay and by are not equal, a and b must be parallel lines with no points of intersection, so ay must be equal to by. Also, if $ax_2 < by_1$ or $ay_2 < bx_1$, the lines also have no points of intersection.</td></tr>
<tr><td rowspan="2">K</td><td colspan="3">f(by, ax1, ax2, by, bx1, bx2)</td></tr>
<tr><td colspan="3">The given code already returns false when $ay_1 > by$ or $ay_2 < by$, that is, when the line is above the rectangle or below the rectangle. Otherwise, considering only the region of rectangle a that is on the same line as b, it covers the line connecting (ax_1, by) and (ax_2, by), so we only have to check that.</td></tr>
</table>