Hong Kong Olympiad in Informatics 2020/21
Heat Event (Senior Group)
Official Solution

Statistics (N = 184)
Full mark = 45. Maximum = 43. Median = 13. Advance to Finals = 13.5 marks or above.

Section A

Q	A	Explanation
1	C	`m mod n` (C/C++: `m % n`) is equivalent to `m - (m div n) * n` (C/C++: `m - (m / n) * n`) in most programming languages (including Pascal and C/C++). Thus, we have `a = -4, b = -1, c = 2, d = -4 × (-1 + 2) = -4`.
2	D	Clearly `a[0]` doesn't affect the outcome of the code. Since the initial value of `answer` (0) is smaller than every element in the array, the output will always be 0.
3	A	Note that the values `x` and `y` are independent of each other, so we may calculate them separately. Simulating the first iterations of the loop, we may notice that values of `x` follow the pattern `0->3->7->3->7`... and the values of `y` follow the pattern `0->2->5->7->2->5->7->`...
4	B	By De Morgan's Law, `NOT (A AND (NOT B OR C))` is equivalent to `NOT A OR NOT (NOT B OR C)`, which is equivalent to `NOT A OR (B AND NOT C)`.
5	B	If Charlie goes, then Bob goes, which in turn implies that Alice will go. Therefore B is the correct answer.
6	B	It is obvious that `a[i]` stores the *i-th* Fibonacci number. Note that $F_n = F_{n-1} + F_{n-2} < 2F_{n-1} < 2^n$ and $F_n > 2F_{n-2} > 2^{\frac{n}{2}}$.As a signed 32-bit integer can only store values in $[-2^{31}, 2^{31} - 1]$, the only possible option would be B.
7	B	A directed graph has an Eulerian trail if and only if at most one vertex has (out-degree - in-degree) = 1, at most one vertex has (in-degree - out-degree) = 1, every other vertex has equal in-degree and out-degree, and all of its vertices with nonzero degree belong to a single connected component of the underlying undirected graph. (i) The graph contains a vertex with in-degree - out-degree = 1 - 3 = -2 (ii) The graph satisfies the conditions described above. (iii) The graph contains a vertex with in-degree - out-degree = 3 - 1 = 2
8	D	We can treat each connected component as a node. In this condensed graph with 3 nodes, there are 3 ways to add 2 edges such that the graph becomes connected. However, each added edge corresponds to one of $2 \cdot 2 = 4$ways to connect two connected components in the original graph. Therefore, there are $3 \cdot 4 \cdot 4 = 48$ ways in total.
9	A	As 31 is `11111` in binary, the function `f(n)` computes the sum of integers smaller or equal to `n` such that its 5 rightmost bits are all '1's.

<table>
<tr><td></td><td></td><td>As 125 is <code>1111101</code> in binary, we may deduce that the only possible bits before the 5 ‘1’s are “00”, “01”, “10”. The desired sum is therefore, <code>0011111</code> + <code>0111111</code> + <code>1011111</code> = 189 in base 10.</td></tr>
<tr><td>10</td><td>D</td><td>Let <code>g[n]</code> be the value of <code>f(n)</code>. We can translate the recursive function to <code>g[n] = n + g[n - 1] + g[n - 2]</code>, where <code>g[0] = -1 and g[1] = 2</code>.

<code>g[2] = 2 + g[1] + g[0] = 2 + 2 + (-1) = 3</code>.</td></tr>
<tr><td>11</td><td>D</td><td><code>g[7]</code> can be found by computing the earlier values one by one.

<table><tr><td><code>n</code></td><td><code>g[n]</code></td></tr><tr><td>3</td><td>$3 + 3 + 2 = 8$</td></tr><tr><td>4</td><td>$4 + 8 + 3 = 15$</td></tr><tr><td>5</td><td>$5 + 15 + 8 = 28$</td></tr><tr><td>6</td><td>$6 + 28 + 15 = 49$</td></tr><tr><td>7</td><td>$7 + 49 + 28 = 84$</td></tr></table></td></tr>
<tr><td>12</td><td>B</td><td>It is obvious that a_1 is 1, (a_5, a_9) is (9, 8) or (8, 9), and a_4 is greater than all other elements, so it must be 7.
For the 5 remaining numbers <code>{2, 3, 4, 5, 6}</code>, we may split them into 2 unordered sets of sizes 2 and 3 and assign them to (a_2, a_3) and (a_6, a_7, a_8). The total number of combinations is therefore $2 \cdot (5C2) = 20$.</td></tr>
<tr><td>13</td><td>B</td><td>The total number of nodes is maximised when every non-leaf node has exactly two children. In this case, the tree has $2 \times 2020 - 1 = 4039$nodes. Such a tree can be easily modified to contain one fewer node while keeping the number of leaf nodes constant.</td></tr>
<tr><td>14</td><td>B</td><td>Let <code>way[i][j]</code> represent the number of ways to reach the cell on the i^{th}column and j^{th} row. We should initialize <code>way[1][j]</code> to 1 for all <code>j</code>. To compute <code>way[i][j]</code>, we add the number of ways to reach the cells that can directly reach the current cell. The completed table of values is shown below:

<table><tr><td>85</td><td>75</td><td>56</td><td>30</td></tr><tr><td>10</td><td>19</td><td>26</td><td>30</td></tr><tr><td>10</td><td>9</td><td>7</td><td>4</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>
The answer is the sum of <code>way[i][4]</code> for all <code>i</code>: $85 + 75 + 56 + 30 = 246$.</td></tr>
<tr><td>15</td><td>A</td><td><code>f(x)</code> returns the number of digits + the number of ‘0’ bits in the binary representation of <code>x</code>. As 88 is <code>1011000</code> in binary, the answer is$7 + 4 = 11$.</td></tr>
</table>

<table>
<tr><td>16</td><td>C</td><td>In order to maximize the number of edges while ensuring the bipartite graph is disconnected, there should be exactly two connected components. (If there are more than two connected components, we can increase the total number of edges by adding an edge between any two connected components.) Thus, we only need to consider the four cases below. To maximize the number of edges in a (bipartite) connected component, we should divide the vertices into two sets whose sizes are as close as possible.

<table><tr><td>Number of vertices in component 1</td><td>Number. of vertices in component 2</td><td>Number of edges</td></tr><tr><td>1</td><td>7</td><td>$0 + 3 \cdot 4 = 12$</td></tr><tr><td>2</td><td>6</td><td>$1 \cdot 1 + 3 \cdot 3 = 10$</td></tr><tr><td>3</td><td>5</td><td>$1 \cdot 2 + 2 \cdot 3 = 8$</td></tr><tr><td>4</td><td>4</td><td>$2 \cdot 2 + 2 \cdot 2 = 8$</td></tr></table>
Hence, the answer is 12.</td></tr>
<tr><td>17</td><td>D</td><td>Let W be the event such that the first player wins. We may set up the equation P(W) = $\frac{5}{8}$ + $\frac{3}{8}$ × (1 - P(W)). Solving the equation gives P(W) = $\frac{8}{11}$.</td></tr>
<tr><td>18</td><td>B</td><td>The program is trying to implement a queue. The maximum capacity of the queue is only 10 but we are trying to push 1 to 15 into it. As a result, the final values in <code>q</code> would be {11, 12, 13, 14, 15, 6, 7, 8, 9, 10} and the tail would point to index 5. The desired sum is therefore$11 + 12 + 13 + 14 + 15 = 65$.</td></tr>
<tr><td>19</td><td>A</td><td>Since a substring is a contiguous segment of characters, the three vowels must appear consecutively in the string <code>EXHAUSTION</code>. We consider the three possible cases separately:

<table><tr><td>Vowels</td><td>Constraints of substring</td><td>Number of substrings</td></tr><tr><td><code>EAU</code></td><td>must start at <code>E</code> and can end at <code>U</code>, <code>S</code> or <code>T</code></td><td>$1 \times 3 = 3$</td></tr><tr><td><code>AUI</code></td><td>can start at <code>X</code>, <code>H</code> or <code>A</code> and must end at <code>I</code></td><td>$3 \times 1 = 3$</td></tr><tr><td><code>UIO</code></td><td>must start at <code>U</code> and can end at <code>O</code> or <code>N</code></td><td>$1 \times 2 = 2$</td></tr></table>
The total number of substrings is therefore $3 + 3 + 2 = 8$.</td></tr>
<tr><td>20</td><td>B</td><td>We may solve this problem by dry running the code. Note that <code>a</code> and <code>b</code> are global variables. Therefore, only the value of <code>a</code> set in the deepest call of <code>f</code> is used.</td></tr>
<tr><td>21</td><td>B</td><td>The program counts the number of ‘1’ bits in the 5 rightmost bits in the binary representation of 56. As 56 is 110100 in binary, the answer is 2.</td></tr>
<tr><td>22</td><td>A</td><td>Consider the situation where Alice and Bob each flips n coins. There are three possible outcomes: Bob has more heads, Alice has more heads or they have the same number of heads. Let p be the probability that Bob has more heads. By symmetry, p is also the probability that Alice has more heads. Now we return to the original problem - there are</td></tr>
</table>

		two ways in which Bob can end up with more heads: 1. Alice and Bob have the same number of heads before Bob's last flip and Bob flips a head on his last flip 2. Bob has more heads before his last flip Hence, the answer is $(1 - 2p) \times \frac{1}{2} + p = \frac{1}{2}$.
23	B	The program implements the merge part in the merge sort algorithm, but the two sequences in the inputs provided are not necessarily sorted. The outputs are as follows: (i) `2 3 1 4 5 7 8 6` (ii) `2 3 4 1 5 6 8 7` (iii) `2 3 1 4 5 7 8 6`
24	C	If we consider the remaining elements after deletion, we may notice that the problem is identical to counting the number of non-empty increasing subsequences. This is a standard dynamic programming problem. Let `a` be the array and $f(x)$ be the number of increasing subsequences ending at `a[x]`. The recurrence relation of $f(x)$ is $f(x) = 1 + \sum_{y<x,\, a[y] < a[x]} f(y)$. The answer is therefore $\sum_{x=1}^{6} f(x) = 1 + 2 + 2 + 6 + 6 + 12 = 29.$
25	C	The top view dictates that there are at least 5 blocks and we can easily construct a valid configuration with 5 blocks. By considering the columns of the top and front views, the maximum possible number of blocks is $2 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = 7$. Hence, the answer is $7 - 5 = 2$.

Section B

Answer and Explanation			
	Pascal	**C**	**C++**
A	`D, E, F, G`		
	A and C must be discovered before E and F, so they cannot be the 6th discovered clue. G requires E and D, which requires B to be discovered, so B also cannot be the 6th clue. The following 4 examples show possible sequences in which D / E / F / G is the 6th clue: D: A->C->E->F->B->D->G E: A->C->B->D->F->E->G F: A->C->B->D->E->F->G G: A->C->B->D->E->G->F		
B1	`x+1`	`x+1`	
B2	`i*i*i-i`	`i*i*i-i`	
	Notice that $f(x) = \sum_{k=1}^{x} k(k+1)(k+2) = \sum_{k=0}^{x+1} k(k-1)(k+1) = \sum_{k=0}^{x+1} (k^3 - k)$. (Note: for $k < 2$, the corresponding term is 0.)		
C	`((i mod 6=0)or(i mod`	`(i%6==0||i%8==0)&&i%24!=0 //`	

<table>
<tr><td></td><td>8=0))and(i mod 24<>0) // ((i mod 6=0)or(i mod 8=0))and not((i mod 6=0)and(i mod 8=0)) // (i mod 6=0)and(i mod8<>0)or(i mod 6<>0)and(i mod 8=0) // (i mod 6=0)xor(i mod 8=0)</td><td>(i%6==0||i%8==0)&&!(i%6==0&&i%8==0)) // i%6==0&&i%8!=0||i%6!=0&&i%8==0 // i%6==0^i%8==0</td></tr>
<tr><td></td><td colspan="2">An integer is divisible by 6 and 8 if and only if it is divisible by their lowest common multiple 24.</td></tr>
<tr><td>D</td><td>n div 6+n div 8-n div 24*2</td><td>n/6+n/8-n/24*2</td></tr>
<tr><td></td><td colspan="2">n/6+n/8 double counts multiples of 24 so we need to subtract n/24 twice</td></tr>
<tr><td>E1</td><td>f</td><td>f</td></tr>
<tr><td>E2</td><td>i=5</td><td>i==5</td></tr>
<tr><td></td><td colspan="2">Note that the exact ASCII values are irrelevant as we are only concerned with the differences between characters. Without loss of generality, assume that ‘a’ is 0 so ‘z’ is 25. Hence, hkoi corresponds to 7, 10, 14, 8 respectively. The differences between these numbers are 3, 4, -6. Note that the first two numbers are positive, so the change of sign should occur when i equals 5. Since i starts from 2, v should be initialized as ‘f’ such that the first printed character is ‘f’ + 2 = ‘h’.</td></tr>
<tr><td>F</td><td colspan="2">17</td></tr>
<tr><td></td><td colspan="2">As 25 is 11001 in binary, f(25) = 11001 xor 1100 xor 110 xor 11 xor 1 = 17.</td></tr>
<tr><td>G</td><td colspan="2">21</td></tr>
<tr><td></td><td colspan="2">Note that the bit at position x from the left is 1 if and only if there is an odd number of ‘1’ bits on its left (including the bit at position x). The answer is very easy to construct with the above observation.</td></tr>
<tr><td>H</td><td colspan="2">2143</td></tr>
<tr><td></td><td colspan="2">Claim: For the bit positioned x (index starts at 0) from the right, there are 2^5 integers between 0 and 63 such that bit x is present after the function f is applied.

Proof:
Consider some integer y between 0 and 63 such that f(y) does not contain bit x, we can show that it is possible to map y into a unique value z such that f(z) contains bit x.
Case 1: x is odd
It is obvious you may toggle (0->1, 1->0) all bits from position x to 5 to find z
Case 2: x is even
We may toggle the leftmost bit to obtain z.
Therefore, the desired number of integers is $\frac{2^6}{2}$.
With the above claim, the answer is easily</td></tr>
</table>

<table>
<tr><td></td><td>$f(64) - f(0) + \sum_{x=0}^{5} 2^x \cdot 2^5 = 127 + 63 \cdot 2^5 = 2143.$</td></tr>
<tr><td rowspan="2">I</td><td>3</td></tr>
<tr><td>From below, we can work out the values of b and c using $f(0)$, $f(1)$and $f(-1)$. Then, we have $a \equiv f(1) - b - c \pmod 3$. Thus, at most 3 queries are needed. Note that since f returns the remainder of $ax^2 + bx + c$ when divided by 3, we won't gain any extra information by querying integers that are congruent modulo 3. Now we will show that 2 queries are not sufficient:
<table>
<tr><td>Queries</td><td>(a, b, c)</td><td>(a, b, c)</td><td>Query results</td></tr>
<tr><td>$f(0)$, $f(1)$</td><td>(0, 1, 0)</td><td>(1, 0, 0)</td><td>0, 1</td></tr>
<tr><td>$f(0)$, $f(2)$</td><td>(0, 2, 0)</td><td>(1, 0, 0)</td><td>0, 1</td></tr>
<tr><td>$f(1)$, $f(2)$</td><td>(0, 1, 1)</td><td>(1, 1, 0)</td><td>2, 0</td></tr>
</table>
For each possible pair of queries, there exist two tuples (a, b, c) with distinct values of a that return the same query result.</td></tr>
<tr><td rowspan="2">J</td><td>2</td></tr>
<tr><td>Note that $f(1) - f(-1) \equiv a + b + c - a - (-b) - c \equiv 2b \pmod 3$. As 2 is self-inverse, $b \equiv 2 \cdot (f(1) - f(-1)) \pmod 3$. Thus, at most 2 queries are needed. Since f returns the remainder of $ax^2 + bx + c$ when divided by 3, we won't gain any extra information by querying integers that are congruent modulo 3. Now we will show that 1 query is not sufficient:
<table>
<tr><td>Query</td><td>(a, b, c)</td><td>(a, b, c)</td><td>Query result</td></tr>
<tr><td>$f(0)$</td><td>(0, 1, 0)</td><td>(1, 0, 0)</td><td>0</td></tr>
<tr><td>$f(1)$</td><td>(0, 1, 0)</td><td>(1, 0, 0)</td><td>1</td></tr>
<tr><td>$f(2)$</td><td>(0, 2, 0)</td><td>(1, 0, 0)</td><td>1</td></tr>
</table>
For each possible query, there exist two tuples (a, b, c) with distinct values of a that return the same query result.</td></tr>
<tr><td rowspan="2">K</td><td>1</td></tr>
<tr><td>$f(0)$ returns the value of c</td></tr>
<tr><td>L1</td><td><code>c[i-1]</code></td><td><code>c[i-1] // j</code></td></tr>
<tr><td>L2</td><td><code>c[i]-1</code></td><td><code>c[i]-1</code></td></tr>
<tr><td>L3</td><td><code>b[j]:=i</code></td><td><code>b[j]=i</code></td></tr>
</table>

<table>
<tr><td></td><td colspan="2"><code>c[i]</code> represents the number of input integers that are smaller than or equal to <code>i</code>, so the number of occurrences of <code>i</code> is <code>c[i] - c[i - 1]</code>. Looping from <code>c[i - 1]</code> to <code>c[i] - 1</code> is exactly <code>c[i] - c[i - 1]</code> times. Since the program outputs the elements of <code>b</code>, we need to store our sorted numbers into <code>b</code>.

Note that this is actually an implementation of the counting sort algorithm.</td></tr>
<tr><td>M1</td><td><code>(s[1] or s[2])and not s[3]</code></td><td><code>(s[1]||s[2])&&!s[3] // s[1]&&s[3]==0||s[2]&&s[3]==0</code></td></tr>
<tr><td>M2</td><td><code>(s[3] or s[4])and not s[5]</code></td><td><code>(s[3]||s[4])&&!s[5] // s[3]&&s[5]==0||s[4]&&s[5]==0</code></td></tr>
<tr><td>M3</td><td><code>(s[5] or s[6])and not s[7]</code></td><td><code>(s[5]||s[6])&&!s[7] // s[5]&&s[7]==0||s[6]&&s[7]==0</code></td></tr>
<tr><td></td><td colspan="2">For the robot to go RIGHT, <code>s[3]</code> should be empty, otherwise the robot will be blocked.
Also the robot is to follow the boundary, so at least one of <code>s[1]</code> and <code>s[2]</code> should be true.
When <code>s[1]</code> is true, the robot “goes straight”;
when <code>s[2]</code> is true and <code>s[1]</code> is false, the robot “turns around a corner”.

Similarly for DOWN and LEFT.
Note that in the given board setting, no two of M1, M2, M3 can be true at the same time.
In other words, there are no width-1 “corridors” or “dead ends”.</td></tr>
</table>