Задача A.

Ідея розв'язання.

В даній задачі потрібно було змінити порядок голосних літер зі зберіганням регістру в стрічці символів.

Даний метод носить назву “Два вказівники”. Ставимо один вказівник на початок стрічки, інший - на кінець стрічки. Перший та другий вказівник будемо зсувати вліво та вправо відповідно. Ітеруємося першим вказівником з ліва на право, доти поки він не буде вказувати на голосну літеру. Аналогічно діємо для іншого вказівника. Коли обидва вказівники вказують на голосні літери робимо обмін літер. Вище наведені дії повторюємо доти поки вказівники не зустрінуться. Також при кожному обміні робимо перевірку регістрів.

Складність алгоритму ***O(n^2).***

Задача B.

Ідея розв'язання.

Формально в даній задач потрібно було знайти кількість підмасивів, таких що сума кожного належала б проміжку ***[A, B].***

Нехай ***a[i] - i***-тий елемент вхідного масиву чисел.

Знайдемо префіксні суми - тобто побудуємо масив prefixSum[..],

такий що ***prefixSum[i] = a[0] + … + a[i] = prefixSum[i - 1] + a[i]***

Звернемо увагу на факт - маючи масив ***prefixSum*** - ми можемо швидко знайти суму на будь-якому підмасиві:

a[i] + a[i + 1] + … + a[j] = prefixSum[j] - prefixSum[i - 1]

Модифікуємо алгоритм ***Merge Sort*** для швидкого вирішення нашої задачі:

На кожній ітерації злиття двох відсортованих частин масиву ми будемо підраховувати відповідь.

Всі елементи з лівого відсортованого підмасиву ***prefixSum*** лежать раніше за всі елементи правого відсортованого підмасиву, тобто якщо ми візьмемо будь-який елемент з лівого відсортованого підмасиву та віднімемо його від будь-якого елементу з правого відсортованого підмасиву, то ми отримаємо суму на коректному підмасиві нашого вхідного масиву.

Будемо ітеруватися по лівому відсортованому підмасиву **prefixSum**, нехай **“i”** - вказівник ітератор. Для кожного зафіксованого елементу лівого відсортованого підмасива - **prefixSum[i]** - знайдемо два індекси які належать правому відсортованому підмасиву, такі що:

1) ***j*** - найменше значення вказівника при якому ***PrefixSum[j] - PrefixSum[i] >= A***
2) ***k*** - найбільше значення вказівника при якому ***PrefixSum[k] - PrefixSum[k] <= B***

Знайшовши значення вище вказаних вказівників ми додаємо до відповіді **k - j + 1** - кількість елементів з правого відсортованого підмасиву, що при різниці з зафіксованим ***i***-м елементом з лівого відсортованого підмасиву будуть утворювати суму, яка належатиме інтервалу ***[A, B]***.

Наступним кроком ми зсуваємо вказівник ***"i"*** на наступний елемент в лівому відсортованому підмасиві. І повторюємо все знову, але очевидно що нам не потрібно кожен раз після того як ми зафіксували новий елемент ***prefixSum[i]*** перебирати всі значення вказівників ***j*** та ***k,*** так як попередній елемент ***prefixSum[i - 1] < prefixSum[i]*** (тому що підмасиви вже відсортовані, за властивістю сортування злиттям).

Також не забуваємо виконувати злиття двох відсортованих підмасивів на кожній фазі.

Складність такого алгориму ***O(n log n).***

Задача C.

Ідея розв'язання.

Ставимо вказівник ***L*** на початок масиву значень гісторграми, а вказівник ***R*** - в кінець. На початку максимальна висота рівня води ***maxHeight = 0***. Оберемо менше значення з двох (на які вказують вказівники) і ми можемо припустити, що ділянка землі яка між ***L*** та ***R*** буде покрита водою як мінімум на висоту ***min(a[L], a[R])***. Нехай ***a[L] < a[R]***, тоді додамо до відповіді ***ans += (a[L] - maxHeight) * (R - L - 1)***, якщо ***maxHeight <= a[L]***. Після чого ми оновлюємо ***maxHeight***, та зсуваємо вказівник який вказував на менше значення. Також коли оновлений вказівник вказує на нове значення, треба відняти від відповіді перетин рівня води з землею. Нехай на попередньому кроці ми зсунули ***L***, тоді ***ans -= min(A[L], maxHeight)***. Всі дії аналогічні і для правого вказівника. Робимо це доти поки ***L <= R***.

Складність алгоритму ***O(n)***.

Задача D.

Ідея розв'язання.

Оскільки все, що за межами матриці має висоту землі 0, то вода буде виливатися з крайніх клітинок. Додамо значення висот клітинок які розташувалися по периметру в структуру даних **priority_queue** або **multiset**.

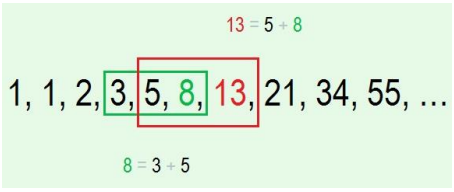
Виберемо та видалимо з черги/сету клітинку з найменшою висотою та розглянемо її сусідні клітинки, які ще не відвідані. Якщо висота сусідньої

клітинки менша за висоту витягнутої клітинки з черги/сету, то ми до відповіді додаємо їхню різницю. Додаємо сусідні клітинки в сет та відмічаємо їх як відвідані. Діємо так доти поки в сеті не залишиться клітинок.

Складність алгоритму **O(m * n * log(m + n)).**

Задача E. "Число 7"

Ідея розв'язання.



Числа Фібоначчі – це елементи нескінченої числової послідовності: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,…., в якій кожне наступне число рівне сумі двох попередніх. Розглянемо основні властивості чисел Фібоначчі, які пов'язані із подільністю. Найбільший спільний дільник двох чисел Фібоначчі рівний числу Фібоначчі з індексом, що рівний найбільшому спільному дільнику індексів, тобто номерів чисел у послідовності Фібоначчі:

$$ÍÑÄ\ \left(F_n, F_m\right) = F_{ÍÑÄ\ (n_n, m)}$$

Наслідками із цієї властивості чисел Фібоначчі:

1. F_n ділиться на F_m тоді і тільки тоді, коли n ділиться на m (виключення m=2);
2. F_n ділиться на $F_3 = 2$ (тобто є парним) тільки при n = 3k;
3. F_n ділиться на $F_4 = 3$ тільки при n = 4k;
4. F_n ділиться на $F_5 = 5$ тільки при n = 5k;
5. F_n ділиться на 7, якщо його номер ділиться тільки на 8;
6. F_n ділиться на 16, якщо його номер ділиться тільки на 12.
7. F_n є прости числом тільки для простих n (за виключенням n =4);
8. Два сусідніх числа послідовності Фібоначчі є взаємно простим.

На основі властивості № 8 побудований розв'язок вказаної задачі.
Слід зауважити, що зчитування даних у форматі C у багатьох випадках суттєво пришвидшує виконання програми, що є досить важливим моментом, який необхідно пам'ятати при розв'язання олімпіадних задач.