Задача A

Розглянемо випадок $p = q$.

Тоді зрозуміло, що всі початкові числа навколо сейфа будуть однакові та рівні S, і необхідно встановити такі числа, щоб їх сума дорівнювала S. Це можна робити довільним способом, наприклад встановити числа $1, 1, \ldots, 1, S - p + 1$.

Тепер маємо $p < q$.

Доведемо допоміжний факт: якщо a, b - взаємнопрості числа, то числа $a, 2a, 3a, \ldots, ba$ дають всі можливі різні залишки при діленні на b. Дійсно, нехай деякі два з них ka та ta при $t < k$ дають однакові залишки при діленні на b. Тоді їх різниця $(k - t)a$ ділиться на b. Однак оскільки a та b - взаємнопрості, тобто не мають спільних дільників більших за 1, то звідси випливає, що $k - t$ ділиться на b. Оскільки $0 < k - t < b$, то $k - t$ не може ділитися на b. Отримали суперечність, отже, $a, 2a, 3a, \ldots, ba$ дають всі можливі залишки при діленні на b.

Позначимо початкові числа навколо сейфа номерами $S_0, S_1, \ldots, S_{p-1}$, а кінцеві числа позначимо $a_0, a_1, \ldots, a_{p-1}$. Тоді $S_i = a_i + a_{i+1} + \ldots + a_{i+q-1}$(тут і далі для зручності будемо вважати, що $a_{kp+i} = a_i, S_{kp+i} = S_i$).

Розглянемо різницю двох послідовних сум $S_{i+1} - S_i = (a_i + a_{i+1} + \ldots + a_{i+q-1}) - (a_{i+1} + a_{i+2} + \ldots + a_{i+q}) = a_i - a_{i+q}$.

Нехай $d_i = a_i - a_0$.

Тоді $d_q = a_q - a_0 = S_1 - S_0$

$$d_{2q} = a_{2q} - a_0 = (a_{2q} - a_q) + (a_q - a_0) = S_{q+1} - S_q + d_q$$

…

$$d_{(p-1)q} = S_{(p-2)q+1} - S_{(p-2)q} + d_{(p-2)q}$$

За фактом доведеним вище можемо знайти всі числа d_i $(1 \le i \le p - 1)$.

Залишилося лише знайти a_0.

Запишемо $S_0 = a_0 + a_1 + \ldots + a_{q-1} = a_0 + (d_1 + a_0) + (d_2 + a_0) + \ldots + (d_{q-1} + a_0) = qa_0 + d_1 + d_2 + \ldots + d_{q-1}$.

Звідси $a_0 = \frac{S_0 - (d_1 + d_2 + \ldots + d_{q-1})}{q}$.

Тепер можемо знайти всі інші елементи з рівностей $a_i = d_i + a_0$.

Задача B

Переможцем у описаній грі є той гравець, на ході якого вперше стає доступним останнє непарне число. Якщо непарних чисел не було, то перемагає другий. Опишемо оптимальну стратегію для цього.

Нехай на і-ому кроці стало вперше доступним для зменшення останнє непарне число. Тоді гравець, який ходить зменшує його і всі решту непарних чисел на 1. В результаті всі числа (у тому числі й ще недоступні) стають парними. Інший гравець своїм ходом

вимушений зменшити хоча б одне з чисел, тому після його ходу буде хоча б одне непарне число. Знову зменшимо всі непарні числа на 1. Таким чином після кожного вашого ходу всі числа на столі залишатимуться парними. Зрозуміло, що оскільки кожен хід зменшує сумарну кількість сірників, то гра закінчиться за скінчену кількість кроків. Оскільки після вашого ходу всі числа є парними, а після ходу суперника є хоча б одне непарне, то ви не можете програти. Врешті-решт, настане момент, коли всі числа стануть нулями і ваш суперник програє.
Якщо всі числа з самого початку є парними, то другий гравець, знову ж таки, зможе після кожного ходу першого робити всі числа парними і перемогти з міркувань описаних вище.

Задача C

Розглянемо як можна було організувати повний перебір та отримати 25 балів за цю задачу:
Для кожних дверей переберемо їх кінцеву позицію. Перевіримо, чи виконується умова (усі заповнені секції закриті, інші відкриті, усі двері на різних позиціях, тощо). Порахуємо для кожних дверей різницю початкової та кінцевої позиції, спробуємо оновити відповідь. Слід зазначити, що цей алгоритм є коректним, оскільки в оптимальному наборі дій не буде перетинів дверей при пересуванні тому, що це додає зайві дії. Складність такого рішення дорівнює O(n ^ m).

Для випадку, коли немає дверей в одному з рядів, можна написати таке жадібне розв'язання:
Якщо n не дорівнює m, то відповіді, очевидно, не існує. Інакше, однозначно відомо, які двері на якій позиції будуть стояти в кінці, адже їх відносний порядок не міняється (не можна зробити так, щоб одні двері помінялися місцями з іншими). Тоді виведемо суму різниць початкової і кінцевої позиції по кожній з дверей. Асимптотика розв'язання O(n).

Коли в другому ряду не більше двох дверей, то можна покращити попереднє розв'язання: Давайте переберемо ті позиції, на яких будуть стояти двері з другого ряду, та чи будуть на цих позиціях якісь двері з першого ряду. Тоді знову отримаємо однозначну розстановку дверей для першого ряду, порахуємо кількість необхідних дій за допомогою попереднього рішення та спробуємо оновити відповідь. Складність дорівнює $O(n^3)$.

Розглянемо 2 розв'язки, які працюють за $O(n^3)$ та вирішують повну задачу.

1. Будемо рахувати таку тривимірну динаміку: d[n][bal1][bal2] – мінімальна кількість дій, які потрібно виконати, щоб були виконані вимоги для перших n секцій. При чому залишалось bal1 "зайвих" дверей в 1му ряді та bal2 - у 2му (bal від слова баланс). Зауважимо, що bal1 і bal2 можуть бути від'ємними. Переходи динаміки: при переході від n до n+1 додаємо до значення динаміки модулі bal1 і bal2 (тому, що рівно стільки дверей буде пересунуто між n и n+1 секціями в якусь із сторін), до балансів потрібно додати відповідну кількість дверей на поточній секції. Далі потрібно розглянути випадки, потрібно чи ні закривати

поточну секцію та перерахувати баланси усіма можливими способами це зробити (максимум 3).

2. Випишемо в окремі вектори відсортовані номери позицій дверей у кожному з рядів і номери заповнених секцій. Будемо рахувати таку кубічну динаміку: d[n][n1][n2] – кількість дій, необхідних для того, щоб прерсунути перших n1 дверей 1го ряду та n2 дверей 2го ряду, щоб закрити перші n заповнених секцій. Розглянемо переходи динаміки: для кожної заповненої секції або беремо двері в одному із рядів, або одночасно в обох. Таким чином отримуємо 3 переходи. Слід зазначити, що відносний порядок дверей не може змінюватися за умовою задачі.

Для того, щоб отримати 100 балів за цю задачу необхідно оптимально реалізувати другу з динамік для попереднього розв'язку (вона є більш швидкою). Для цього потрібно помітити, що в усіх переходах кількість закритих заповнених секцій завжди збільшується на один. Тому достатньо просто зберігати у пам'яті лише 2 двовимірні масиви, що значно пришвидшує роботу програми.

Задача D

Щоб пройти першу групу тестів, достатньо написати повний перебір усіх можливих підвідрізків.

Розглянемо відрізок з найбільшим середнім значеням. Доведемо, що кількість елементів в ньому не більша трьох:

- Нехай відрізок має координати [L, R] (L < R)
- Якщо R - L + 1 <= 3, то твердження вірне
- Інакше поділимо цей відрізок на два менших: [L, (L + R) / 2] та [(L + R) / 2 + 1, R]; середнє значення хоча б на одному з них не менше, ніж на [L, R] (середнє значення об'єднання множин не може перевищувати максимум з середніх значень кожної множини, які в нього входять)

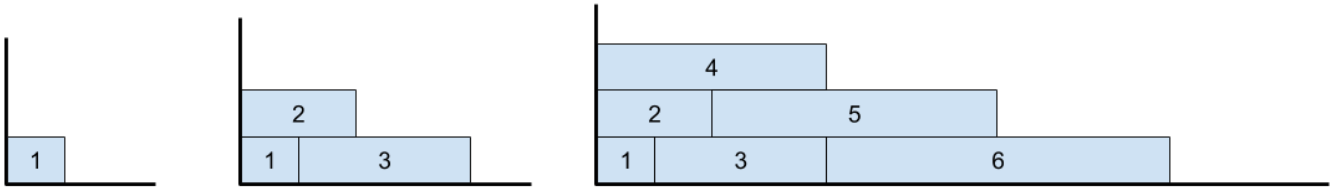
Тобто для проходження другої групи тестів необхідно просто розглянути усі можливі неперервні відрізки з двох та трьох елементів.

Для повного розв'язання необхідно оптимізувати попередню ідею за допомогою структур даних. Це можна зробити за допомогою дерева відрізків: в кожній вершині будемо зберігати найбільшу суму двох\трьох послідовних елементів, та перші\останні два елементи послідовності, що відповідають відрізку відповідної вершини.
При об'єднанні двох вершин необхідно обрати максимум з оптимальних відповідей довжини два та три, а також розглянути випадок, що максимум утворився на стику двох вершин (саме для цього необхідно явно зберігати перші та останні два елементи відповідного відрізка).

Така реализація виконує порядку O((N + M) * log(N)) операцій.

Задача А

Покажемо, що пірамідка з k рівнів містить якнайменше $\frac{k(k+1)}{2}$ кубиків. Дійсно, k-й рівень складається хоча б з одного кубика, $k-1$-й рівень складається хоча б з двох кубиків (щоб кубик з k-го рівня лежав хоча б на двох), $k-2$-й рівень складається хоча б з трьох кубиків (щоб кожен з кубиків з $k-1$-го рівня лежав хоча б на двох) і так далі. Таким чином перший рівень буде містити хоча б k кубиків. Отже сумарна кількість кубиків в пірамідці з k рівнів не менше, ніж $1+2+\ldots+k=\frac{k(k+1)}{2}$.



Покажемо, як для кожного k можна побудувати пірамідку з рівно $\frac{k(k+1)}{2}$ кубиків:

Тобто, пірамідку з $k+1$ рівня можна побудувати з пірамідки з k рівнів шляхом додавання на кожен рівень, починаючи з найнижчого, по одному кубику в порядку зменшення довжини.

Інтуїтивно зрозуміло, що така конструкція завжди буде пірамідкою.

Доведемо це формально.

Нехай -й знизу рівень в конструкції з k рівнів має довжину $L_{k,i}$ та закінчується кубиком з довжиною $last_{k,i}$. Тоді доведемо, що для конструкції з k рівнів, побудованою за описаним вище алгоритмом виконується таке твердження: для кожного i $(2 \le i \le k)$ $1 \le L_{k,i+1} - L_{k,i} < last_{k,i}$

Доведемо це твердження за індукцією.

Для $k=2$ твердження виконується.

Нехай воно виконується для $k=t$, доведемо для $k=t+1$.

Оскільки кубики, що додаються до сусідніх рівнів відрізняються за довжиною на один, то різниця $last_{t+1,i+1} - last_{t,i} = 1$

Також $L_{t+1,i} = L_{t,i} + last_{t+1,i}$

Тоді $L_{t+1,i+1} - L_{t+1,i} = L_{t,i+1} + last_{t+1,i} - L_{t,i} - last_{t,i} = L_{t,i+1} - L_{t,i} + last_{t+1,i} - last_{t,i}$

З одного боку $L_{t,i+1} - L_{t,i} + last_{t+1,i} - last_{t,i} \ge 1 + 1 > 1$

З іншого боку $L_{t,i+1} - L_{t,i} + last_{t+1,i} - last_{t,i} < last_{t,i} + last_{t+1,i} - last_{t,i} = last_{t+1,i}$

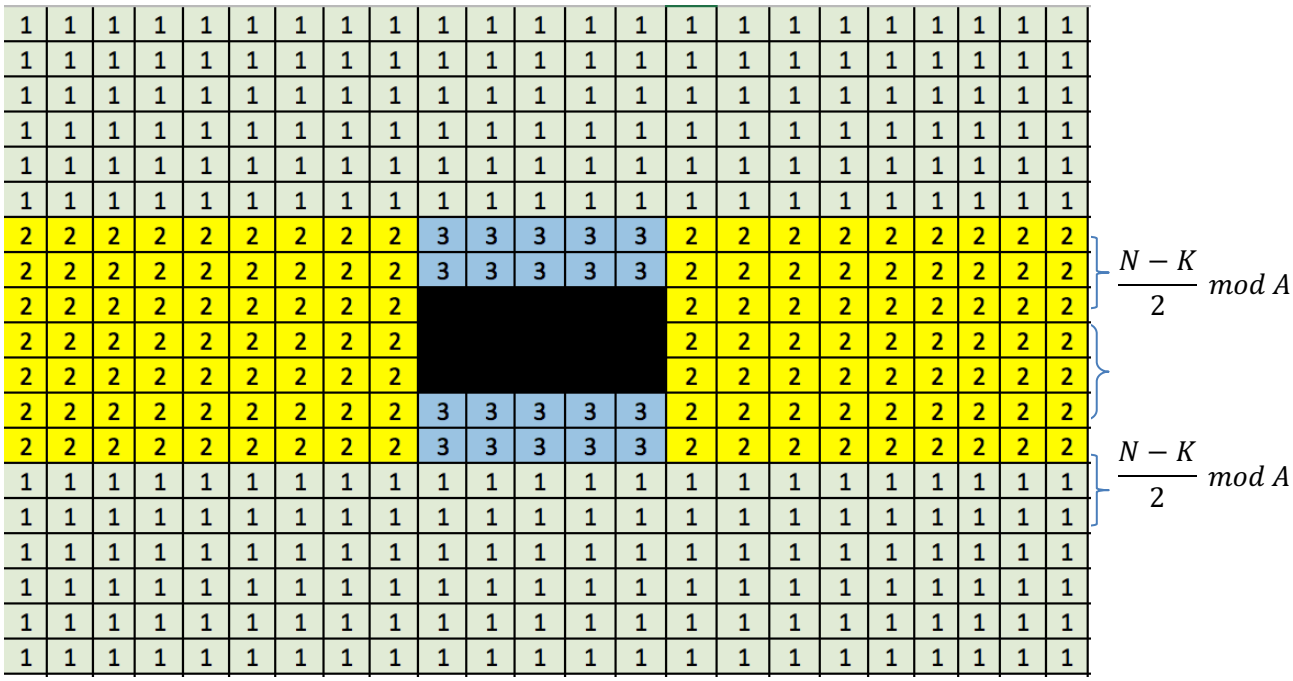
Таким чином твердження доведено.

З $1 \le L_{k,i+1} - L_{k,i}$випливає, що кожен наступний рівень в порядку знизу догори буде не більше, за попередній.

З $1 \le L_{k,i+1} - L_{k,i} < last_{k,i}$ випливає, що кожен новий кубик буде лежати рівно на двох кубиках з попереднього рівня (хоча б на 1 на кубику знизу ліворуч та хоча б на $last_{k,i} - (L_{k,i+1} - L_{k,i})$ на кубику знизу праворуч)
Отже така конструкція завжди є пірамідкою.

Задача B

Потрібно перебирати одну із сторін наліпки, потім, виходячи з периметру, знайти іншу. Для кожного фіксованого розміру A x B треба знайти скільки наліпок поміститься на кришці ноутбука розміром N x M та знайти максимум з них. Помітимо, що якщо б на кришці не було логотипу, то можна було б розмістити $\left\lfloor \frac{N}{A} \right\rfloor \cdot \left\lfloor \frac{M}{B} \right\rfloor$ наліпок. Нехай розмір логотипу K x L. Розіб'ємо кришку на 3 зони, як показано на малюнку:



Розглянемо таку стратегію розміщення наліпок заданого розміру. Спочатку розмістимо максимальну можливу кількість наліпок окремо у зоні 1. Потім незалежно розмістимо максимальну кількість наліпок у зоні 2. Стверджується, що це і буде шукана кількість. Доведемо цей факт. Припустимо, що у нас є деяке оптимальне розміщення наліпок. Розглянемо ті наліпки, що хоча б частково лежать у першій зоні. Помітимо, що ми можемо вертикально перемістимо усі ці прямокутники, щоб вони повністю лежали у ній. Це виплаває з того, що висота кожної з прямокутних частин першої зони ділиться на A. Тоді зона 3 залишиться зовсім порожньою, бо будь-який прямокутник який зачіпає цю зону, також частково знаходиться у першій зоні, а усі такі ми вже прибрали. Тому у цьому

розміщенні нема прямокутників які лежать у декількох зонах одночасно. Застосовуючи формулу кількості наліпок у прямокутнику отримуємо формулу:

$$2 \cdot \left\lfloor \frac{N-K}{2A} \right\rfloor \cdot \left\lfloor \frac{M}{B} \right\rfloor + 2 \cdot \left\lfloor \frac{2 \cdot \left(\left(\frac{N-K}{2} \right) \bmod A \right) + K}{A} \right\rfloor \cdot \left\lfloor \frac{M-L}{2B} \right\rfloor$$

Задача C

Описуючи ідею розв'язання, ми виходимо з припущення, що область дії полів лежить у межах таблиці. Те, що слід робити, коли область дії того чи іншого поля виходить за межі таблиці, описано наприкінці розбору.

Розглянемо спершу випадок, коли таблиця складається з одного рядка (тобто має розмір 1 × *M*). Запровадимо додаткову лінійну таблицю *T* розміру 1 × *M*, яку спершу заповнено нулями. Додамо по одиниці до кожної клітинки, що є лівим краєм поля і правим краєм поля, а від значення кожної клітинки, що є центром поля, віднімемо по 2 (якщо одна клітинка є краєм або центром кількох полів, відповідні значення додаються/віднімаються). Після цього сумарні сили полів для кожної клітинки відновлюються за один прохід зліва направо: у кожен момент часу, проходячи клітинку *m* (1 ≤ *m* ≤ *M*), ми пам'ятаємо суму S_m всіх значень у таблиці *T* від першого до поточного, а також F_m — сумарну силу полів у поточному полі. Сумарна сила полів у наступному полі дорівнює $F_{m+1} = S_m + F_m$.

Для випадку кількох рядків (*N* × *M*) ми здійснимо ту саму операцію окремо для кожного рядка. Залишається лише зрозуміти, як ефективно додавати одиниці до відповідних клітинок на діагоналях, що є межами полів, та віднімати двійки на центральній вертикалі поля.

Якщо в клітинці (*i*, *j*) розташовано центр поля сили *x*, то відняти двійку від усіх клітинок на відповідному стовпці від (*i* − *x*, *j*) до (*i* + *x*, *j*) можна таким чином: узяти додаткову таблицю розміру *N* × *M*, спочатку заповнену нулями; відняти від її клітинки (*i* − *x*, *j*) двійку, а до значення в клітинці (*i* + *x*, *j*) додати два. Після здійснення такої операції для всіх полів слід пройтися зверху донизу по всій таблиці і значення кожної клітини замінити на суму значень в усіх клітинах даного стовпця, що розташовані не нижче від даної. Цілком аналогічно можна додати одиницю до діагональних меж полів: розставляємо одиниці зверху, мінус одиниці знизу діагоналей і за повний лінійний прохід таблиці міняємо значення у клітинках на суми, але вже не на стовпці, а по діагоналях. Для цього нам знадобиться не одна, а дві додаткові таблиці (і, відповідно, два проходи для них): одна для сум по діагоналях, що йдуть зліва направо, інша — для діагональ справа наліво.

Нарешті, розберемося, що робити, якщо поля виходять за межі таблиці. Найпростіший вихід, який, однак, дозволяє набрати повний бал у випадку ефективної реалізації, —

розглядати розширену таблицю, зліва, справа, згори та знизу від основної частини якої розташовано смуги, що мають ширину/висоту, яка дорівнює сумі довжини та ширини основної частини таблиці. Якщо поле виходить навіть за межі такої таблиці, це означає, що воно повністю покриває основну частину таблиці, тому його силу можна зменшити до такої величини, щоб воно досі покривало основну частину таблиці, але вміщалося у розширену, і запам'ятати значення, на яке ми зменшили силу поля, щоб при виведенні відповіді до кожного значення цю величину знову додати.

Задача D

Повний перебір може мати наступний вигляд:

Для кожної хвилини маємо множину чисел. Перебираємо порядок розсадження всіх людей, та моделюємо процес, описаний в умові. Асимптотика рішення дорівнює O(n * (n + m)! * n).

Для того, щоб набрати більшу кількість балів, треба помітити, що порядок розсадження гостей неважливий. Щоб довести це, давайте спробуємо поміняти місцями будь-яких двох сусідніх людей у послідовності, і побачити, що нічого не зміниться.

Довести той факт, що зміна порядку сусідів не змінює ні вартість, ні множину заповнених позицій, можна наступним чином:

Припустимо, ми поміняли місцями людей які будуть сідати **x** та **x + 1** по порядку. Очевидно, що для людей з іншими номерами нічого не зміниться. Не зміниться й сумарна вартість розсадки цих двох людей. Припустимо, що до того, як ми міняли їх місцями, перша людина хотіла сісти на місце з номером L1, а сіла на місце R1. Друга, відповідно, L2 та R2. Вартість такої розсадки -- це сумарна довжина цих відрізків, мінус два. Припустимо, що відрізки не перетинаються, тоді ці числа залишаться незмінними. Інакше, помітимо, що в результаті заповнені позиції та сумарна довжина відрізків теж не зміниться. Отже, порядок двох сусідніх людей неважливий.

Завдяки послідовним обмінам двох сусідів ми можемо досягти будь-якого порядку, отже, усі вони мають однакову сумарну засмученість.

Використовуючи цей факт, розглянемо рішення для випадку всіх **T = 1**. З описаних вище міркувань ми можемо вважати, що якщо є розсадження та його вартість для хвилини з номером **x,** то коли ми зможемо швидко зрозуміти, яке місце буде займати гість, що прийде о хвилині з номером **x + 1**, ми зможемо перерахувати розсадження та вартість для цієї хвилини, а отже, і розв'язувати задачу.

Щоб навчитися підтримувати інформацію про зайняті позиції та вартість розсадження, давайте заведемо масив **A**, в якому **A[x]** буде дорівнювати **0**, якщо позиція **x** зайнята, та

1, якщо ні. Над цим масивом побудуємо дерево відрізків, що зберігає з суму на відрізку. Тоді ми зможемо шукати найпершу позицію масиву з заданою сумою на префіксі очевидним спуском по дереву (детальніше можна прочитати у статті http://e-maxx.ru/algo/segment_tree), а отже і **k**-ту вільну позицію у поточній розсадці. Тоді щоб знайти позицію, яку займе людина з улюбленим числом **x**, можна порахувати суму на префіксі масиву **A** до позиції **x - 1** (нехай вона буде дорівнювати **k**), і знайти (**k + 1**) вільну позицію розсадки, після чого поновити масив та дерево.

Цю ідею можна використати, щоб написати очевидне рішення за O(n * (n + m) * log(n + m)), кожної хвилині розсаджуючи людей заново.

Автор пропонує наступне рішення:

Спробуємо для кожної хвилини знайти оптимальну розсадку. Можна помітити, що множина присутніх гостей для двох послідовних хвилин відрізняються рівно однією людиною, і спробувати використати цей факт.

Спочатку трохи інакше поглянемо на задачу. Для кожної людини запам’ятаємо хвилини, у які він буде присутній на вечірці -- це деякий відрізок **[L; R]**.
Тепер давайте розглянемо деякий відрізок часу, хвилини з номерами від **X** до **Y**. Тоді відносно цього відрізку часу усіх людей можна поділити на три групи.

- Люди, які не будуть присутні в цей час зовсім.
- Люди, які будуть присутні протягом всього часу.
- Люди, які будуть лише на деякому відрізку цього часу

Побачимо, що людей третього типу буде не більше, ніж **Y - X + 1**, адже таким людям в одну з хвилин відрізку треба або прийти на вечірку, або залишити її.

Зробимо наступне:

- Людей першого типу не будемо ніяк враховувати.
- Розсадимо в довільному порядку усіх людей другого типу (як в підзадачі, де всі T = 1).
- Розділимо відрізок на дві частини, і рекурсивно розв’яжемо задачу окремо для кожного з підвідрізків, використовуючи **лише** людей третього типу та поточну (для відрізку **[X; Y]**) розсадку.
- При виході з рекурсії не забудемо всіх людей, яких ми розсадили на цьому кроці, прибрати з нашої розсадки, та поновити поточну сумарну вартість. Тоді ми точно будемо знати, що на кожному кроці ми підтримуємо в масиві **A** розсадженими тільки тих людей, які **завжди** присутні у цей відрізок часу.
- Очевидно, що якщо в рекурсії для відрізку **[X; X]** ми маємо мінімальну вартість всіх людей, які присутні у цей момент часу, то ми маємо відповідь для хвилини з номером **X**.

Викликаючи рекурсію для відрізку **[1; N]**, і кожного разу поділяючи поточний відрізок навпіл, ми отримаємо глибину рекурсії приблизно **log(N)**. На кожному рівні рекурсії ці

відрізки не перетинаються, тому їх сумарна довжина рівно **N (**для кожного рівня рекурсії). Оскільки для кожного відрізку кількість людей, що присутні не весь час, не більше, ніж довжина відрізка, то в сумі ми виконаємо **O(N * log(N))** операцій вигляду “посадити гостя”, та стільки ж операцій “прибрати гостя”. Кожна операція виконується за **log(N)**, отже сумарна асимптотика рішення складає $\mathbf{O(N * log(N)^2)}$.