МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

(Национальный Исследовательский Университет)

Институт№9: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**ПРАКТИКУМ**

По циклу дисциплин «Информатика»

2 семестр

Тема:

«Сортировка и поиск»

|  |  |
| --- | --- |
| Группа: | М8О-106Б-22 |
| Студент: | Абдисаламов Э. |
| Преподаватель: | Дубинин А. В. |
| Оценка: |  |
| Дата: |  |

Москва, 2023

Содержание

[Введение 4](#_Toc131537686)

[1. Теоретическая часть 5](#_Toc131537687)

[1.1 Матрица 5](#_Toc131537688)

[1.2 Разреженные матрицы 6](#_Toc131537696)

[1.3 Вектор как структура данных 8](#_Toc131537697)

[2. Практическая часть 9](#_Toc131537698)

[2.1 Задание 9](#_Toc131537699)

[2.2. Вариант схемы размещения матрицы 10](#_Toc131537702)

[2.3 Вариант преобразований: 10](#_Toc131537703)

[2.4 Использованные переменные и функции 10](#_Toc131537704)

[2.5 Протокол 11](#_Toc131537706)

[2.6 Оценка сложности функций 12](#_Toc131537707)

[2.7 Входные и выходные данные. Тесты 12](#_Toc131537708)

[Заключение 17](#_Toc131537709)

[Список использованных источников 18](#_Toc131537710)

# Введение

# Сортировка и поиск – два фундаментальных понятия в информатике, важность и применяемость которых трудно переоценить. Сортировка позволяет упорядочить данные по определенному критерию, облегчая поиск и анализ информации. Алгоритмы сортировки являются неотъемлемой частью успешного решения задач в различных областях. Поиск, в свою очередь, помогает находить информацию в заданном наборе данных, экономя время и ресурсы.

# Эти две операции широко применяются в современном информационном мире. От поиска информации в Интернете до анализа больших объемов данных в бизнесе, сортировка и поиск позволяют нам управлять и обрабатывать информацию эффективно. Они используются в информационных системах, базах данных, алгоритмическом программировании, биоинформатике, финансовых системах, логистике, медицинских исследованиях и многих других областях.

# 1. Теоретическая часть

## 1.1 Быстрая сортировка Хоара

Быстрая сортировка, сортировка Хоара (англ. quicksort), часто называемая qsort (по имени в стандартной библиотеке языка Си) — алгоритм сортировки, разработанный английским информатиком Тони Хоаром во время своей работы в МГУ в 1960 году.

Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(n\*log n) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

Общая идея алгоритма состоит в следующем:

1. Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность (см. ниже).
2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие»[2].
3. Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие или равные опорного» и «равные и большие опорного»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Хоар разработал этот метод применительно к машинному переводу; словарь хранился на магнитной ленте, и сортировка слов обрабатываемого текста позволяла получить их переводы за один прогон ленты, без перемотки её назад. Алгоритм был придуман Хоаром во время его пребывания в Советском Союзе, где он обучался в Московском университете компьютерному переводу и занимался разработкой русско-английского разговорника.

Оценка сложности алгоритма

Ясно, что операция разделения массива на две части относительно опорного элемента занимает время O(n). Поскольку все операции разделения, проделываемые на одной глубине рекурсии, обрабатывают разные части исходного массива, размер которого постоянен, суммарно на каждом уровне рекурсии потребуется также O(n) операций. Следовательно, общая сложность алгоритма определяется лишь количеством разделений, то есть глубиной рекурсии. Глубина рекурсии, в свою очередь, зависит от сочетания входных данных и способа определения опорного элемента.

В среднем сложность алгоритма составляет O(nlogn), но в худшем случае может достигать O(). Это достигается, когда опорный элемент равен минимальному или максимальному элементу в массиве. Тогда массив делятся на 2 массива, размер одного 1, а другого n-1. Хуже то, что в таком случае глубина рекурсии при выполнении алгоритма достигнет n, что будет означать n-кратное сохранение адреса возврата и локальных переменных процедуры разделения массивов. Для больших значений n худший случай может привести к исчерпанию памяти (переполнению стека) во время работы программы.

Достоинства и недостатки

Достоинства:

1. Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
2. Алгоритм очень короткий: запомнив основные моменты, его легко написать «из головы», невелика константа при n log n.
3. С модификациями требует лишь O(log n) дополнительной памяти в виде стека (неулучшенный рекурсивный алгоритм — в худшем случае O(n) стека).
4. Хорошо сочетается с механизмами кэширования и виртуальной памяти.
5. Допускает естественное распараллеливание (сортировка выделенных подмассивов в параллельно выполняющихся подпроцессах).
6. Допускает эффективную модификацию для сортировки по нескольким ключам (в частности — алгоритм Седжвика для сортировки строк): благодаря тому, что в процессе разделения автоматически выделяется отрезок элементов, равных опорному, этот отрезок можно сразу же сортировать по следующему ключу.
7. Работает на связных списках и других структурах с последовательным доступом, допускающих эффективный проход как от начала к концу, так и от конца к началу.

Недостатки:

1. Сильно деградирует по скорости (до O(n^{2})) в худшем или близком к нему случае, что может случиться при неудачных входных данных.
2. Прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке переполнения стека, так как в худшем случае ей может потребоваться сделать O(n) вложенных рекурсивных вызовов.

3. Неустойчив.

## 1.2 Бинарный поиск

Бинарный поиск — тип поискового алгоритма, который последовательно делит пополам заранее отсортированный массив данных, чтобы обнаружить нужный элемент. Другие его названия — двоичный поиск, метод половинного деления, дихотомия.

Принцип работы алгоритма бинарного поиска

Основная последовательность действий алгоритма выглядит так:

1. Сортируем массив данных.
2. Делим его пополам и находим середину.
3. Сравниваем срединный элемент с заданным искомым элементом.
4. Если искомое число больше среднего — продолжаем поиск в правой части массива (если он отсортирован по возрастанию): делим ее пополам, повторяя пункт 3. Если же заданное число меньше — алгоритм продолжит поиск в левой части массива, снова возвращаясь к пункту 3.

2. Практическая часть

2.1 Задание

Составить программу на языке Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

Метод сортировки: Быстрая сортировка Хоара (не рекурсивный вариант)

Структура таблицы:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип ключа | Длина ключа байтах | Хранение данных и ключей | Минимальное число элемент таблицы |
| целое + литера | 9 | вместе | 17 |

2.2 Метод реализации

Так как метод сортировки не рекурсивный, то сделаем стек из структур, представляющих границы: левый и правый. Опорный элемент выбираем случайным образом из первого, последнего и серединного элемента.

Двоичный поиск не совсем обычный, он должен возвращать первое вхождение по порядку в таблице, а не по первенству нахождению. Такое реализуем так: после нахождения искомого элемента сохраняем этот элемент и что мы нашли, но вместо возвращение элемента, мы сужаем нижнюю границу и так до конца. Тогда сохранившийся элемент будет первым по порядку в таблице.

## 2.3 Протокол

* main.c:
* #include <stdio.h>
* #include <stdbool.h>
* #include <unistd.h>
* #include <stdlib.h>
* #include <time.h>
* #include "borders\_stack.h"
* #define max\_elem\_data\_size 1000
* #define max\_arr\_elem\_size 100
* typedef struct
* {
* int intg;
* char lit;
* } key;
* typedef struct
* {
* key key;
* char data[max\_elem\_data\_size];
* } elem;
* void print\_elem(elem \*el)
* {
* printf("{ %d, %c}: %s\n", el->key.intg, el->key.lit, el->data);
* }
* void print\_arr\_elem(elem \*arr, int arr\_size)
* {
* for (int i = 0; i < arr\_size; i++)
* {
* printf("%d %c : %s\n", arr[i].key.intg, arr[i].key.lit, arr[i].data);
* }
* }
* int cmp(key a, key b)
* {
* if (a.intg != b.intg)
* {
* return a.intg - b.intg;
* }
* return a.lit - b.lit;
* }
* elem calc\_pivot(elem \*arr, int i, int j)
* {
* srand(time(NULL));       // инициализация генератора случайных чисел
* int random = rand() % 3; // генерируем случайное число от 0 до 2
* elem i\_j\_hoarult;
* switch (random)
* {
* case 0:
* i\_j\_hoarult = arr[i];
* break;
* case 1:
* i\_j\_hoarult = arr[j / 2];
* break;
* case 2:
* i\_j\_hoarult = arr[j];
* break;
* }
* return i\_j\_hoarult;
* }
* void swap(elem \*arr, int i, int j)
* {
* elem c = arr[i];
* arr[i] = arr[j];
* arr[j] = c;
* }
* borders hoar\_partition(elem \*arr, int l, int r, elem b)
* {
* int i = l, j = r;
* do
* {
* while (cmp(arr[i].key, b.key) < 0)
* i++;
* while (cmp(arr[j].key, b.key) > 0)
* j--;
* if (i <= j)
* {
* swap(arr, i, j);
* i++;
* if (j > 0)
* j--;
* }
* } while (i <= j);
* borders i\_j\_hoar = {i, j};
* return i\_j\_hoar;
* }
* void hoar\_sort(elem \*arr, int size)
* {
* borders\_stack st;
* borders\_stack\_init(&st);
* // borders brdr = {0, size-1};
* // borders\_stack\_push(&st, brdr);
* borders\_stack\_push(&st, (borders){0, size - 1});
* while (!borders\_stack\_is\_empty(&st))
* {
* borders cur\_border = borders\_stack\_pop(&st);
* while (cur\_border.R > cur\_border.L)
* {
* elem pivot = calc\_pivot(arr, cur\_border.L, cur\_border.R);
* borders i\_j\_hoar = hoar\_partition(arr, cur\_border.L, cur\_border.R, pivot);
* if (i\_j\_hoar.L < cur\_border.R)
* {
* borders r\_half = {i\_j\_hoar.L, cur\_border.R};
* borders\_stack\_push(&st, r\_half);
* }
* cur\_border.R = i\_j\_hoar.R;
* }
* }
* }
* void mix\_arr\_elem(elem \*arr, int arr\_size)
* {
* srand(time(NULL)); // Инициализируем генератор случайных чисел
* for (int i = 0; i < arr\_size; i++)
* {
* int j = rand() % arr\_size;
* swap(arr, i, j);
* }
* }
* elem \*bin\_search(elem \*arr, int arr\_size, key srch\_elem\_key)
* {
* int i = 0, j = arr\_size - 1;
* bool found = false;
* elem \*found\_el = NULL;
* while (i <= j)
* {
* int mid\_idx = i + (j - i) / 2;
* if (cmp(arr[mid\_idx].key, srch\_elem\_key) == 0)
* {
* found = true;
* found\_el = &arr[mid\_idx];
* j = mid\_idx - 1;
* }
* if (cmp(arr[mid\_idx].key, srch\_elem\_key) > 0)
* j = mid\_idx - 1;
* if (cmp(arr[mid\_idx].key, srch\_elem\_key) < 0)
* i = mid\_idx + 1;
* }
* return found\_el;
* }
* int main()
* {
* elem arr[] = {
* {
* {1, 'd'},
* " 1.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢿⣿⣿⣿⣿⣿",
* },
* {
* {2, 'b'},
* " 2.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣤⣿⣿⣿⣿⣿",
* },
* {
* {2, 'z'},
* " 3.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⢿⠻⠟⠻⠟⠿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {3, 'f'},
* " 4.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠛⠉⠀⠁⣀⣀⠐⠒⠂⠆⣀⣀⠉⠛⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠛⢛⢉⡉⡉⢛⣛⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {4, 'f'},
* " 5.⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⠃⢀⠐⣠⠷⠛⠉⠉⠙⠲⣄⠀⡄⠈⠻⠆⠀⡘⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢛⠍⣀⢹⢤⣃⠹⣧⢢⣂⢙⠷⢌⠻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {5, 'f'},
* " 6.⣿⣿⣿⣿⣿⣿⣿⣿⠋⠀⠐⣠⡞⠁⠀⠀⠀⠀⠀⣀⣟⣀⡀⢁⡤⢤⣆⠑⡄⠘⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠘⠁⠜⣠⣫⢴⢺⡵⢥⣕⡯⢾⠦⢱⠳⡜⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {6, 'f'},
* " 7.⣿⣿⣿⣿⣿⣿⣿⠇⠀⡈⢀⡏⠀⠀⠀⣰⡶⠿⠏⠉⠉⠉⢹⡏⢀⣰⢿⡆⠰⠆⠈⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡁⠎⡰⣸⠡⢶⣏⢿⣸⠯⣎⣰⠟⠚⡀⠠⡇⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {7, 'f'},
* " 8.⣿⣿⣿⣿⣿⣿⡏⠀⢐⠀⣼⣇⠀⣠⠞⢁⡄⠒⠂⡉⡉⡉⠙⢿⣋⣁⡼⠃⠀⡅⢂⠹⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⡇⢰⣵⠱⢛⣷⢾⢿⢞⣋⢗⠐⣨⣾⠇⡨⢱⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {8, 'f'},
* " 9.⣿⣿⣿⣿⣿⣿⡁⠐⡊⢠⠀⣨⠟⢁⡴⠁⡀⠒⠂⠂⣀⡰⢦⡀⢨⠡⠀⠐⢂⠐⡁⠘⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣧⢠⠘⠊⣆⠟⣢⣋⡫⠛⠁⣐⣴⠟⣡⠆⠊⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {9, 'f'},
* "10.⣿⣿⣿⣿⣿⣿⠀⠐⡆⠀⠁⠀⢀⡞⠀⠂⠂⠢⠄⡂⢸⣆⢀⡟⠀⢀⡶⠶⢤⡀⠑⣈⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡘⢶⣲⠈⣌⢀⣀⠀⠀⠈⢼⢐⠒⠐⢠⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {10, 'f'},
* "11.⣿⣿⣿⣿⣿⣿⡅⠀⢳⡀⠀⠀⣈⣤⠶⠒⠲⣆⢀⣡⡀⠉⡉⠀⡀⢾⡀⢀⢠⡇⠐⢠⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣌⡛⠶⠤⢭⠭⠭⠥⠾⣘⣩⣴⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {12, 'f'},
* "12.⣿⣿⣿⣿⣿⣿⣗⡀⠠⢈⠁⣰⠏⠀⠀⠐⠀⠈⣿⠀⢸⠀⣀⣤⣄⠈⠓⠖⠉⣀⠂⣼⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣶⣶⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {12, 'f'},
* "13.⣿⣿⣿⣿⣿⣿⣿⣦⠀⠨⠀⢻⡀⠀⠀⠀⢀⣴⠏⠛⢧⡞⠁⢀⠈⢷⠀⠐⡀⠁⣼⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {12, 'f'},
* "14.⣿⣿⣿⣿⣿⣿⣿⣿⣷⡐⢠⠀⠙⠲⠶⠖⠋⠁⠄⡀⠸⣇⠀⠀⣠⡟⠀⠂⣁⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⣛⣙⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {12, 'f'},
* "15.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣦⣌⠐⠒⠄⣀⠁⠉⠄⠡⠄⠈⢙⠛⠁⢀⢤⣺⣽⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢡⠈⠿⢿⣿⣿⣿⣿⣿⣿",
* },
* {
* {12, 'f'},
* "16.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣤⣄⣈⢉⡁⣁⢉⡁⣤⣴⣮⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠟⣛⡛⡛⠘⠀⠡⠩⠿⠝⢻⣿⣿⣿",
* },
* {
* {12, 'f'},
* "17.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣦⡄⢐⠀⢀⡀⠄⠺⢿⣿⣿⣿⣿",
* },
* {
* {17, 'f'},
* "18.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢠⠔⢠⣴⣄⣨⣔⠄⣿⣿⣿⣿",
* },
* {
* {18, 'f'},
* "19.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣧⣴⣾⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿",
* },
* {
* {19, 'f'},
* "20.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {20, 'f'},
* "21.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {21, 'f'},
* "22.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢿⢟⣛⢛⣛⣛⣛⠛⡻⠿⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {22, 'f'},
* "23.⣿⣿⣯⣿⣿⣿⣿⣿⣿⣿⣿⡿⠿⢿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠟⢡⡾⠝⡈⠅⢈⢀⡉⢉⡛⣷⡚⡑⠬⠻⢿⠿⢿⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {23, 'f'},
* "24.⣿⣿⣿⣿⣿⣿⣿⣿⢿⡋⠔⠈⣉⠽⢯⠽⢏⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢋⡴⢠⠆⠁⣀⠔⢈⠁⠆⠑⠀⠉⠡⠠⠘⠸⠀⠐⠈⠀⠤⠄⠄⢄⠠⠪⣙⠻⢿⣿⣿⣿⣿⣿",
* },
* {
* {24, 'f'},
* "25.⣿⣿⣿⣿⣿⣿⡟⢀⠌⠠⣱⠴⠨⠲⢹⠀⠢⢻⣾⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠟⠘⠠⠀⢄⠂⠄⠀⠀⠐⢐⢨⢀⡐⠠⣐⠐⡀⠦⢰⡰⣶⣿⣿⣿⣷⣶⣤⡠⡄⢰⢹⣿⣿⣿⣿",
* },
* {
* {25, 'f'},
* "26.⣿⣿⣿⣿⣿⣿⠇⠀⠀⠘⡡⠐⢀⢠⢞⠄⠀⠀⠹⠉⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⠛⢋⢀⠁⠀⠀⠀⢀⢀⣐⢼⣀⢆⡸⡄⢆⢑⠅⠸⢑⠰⢸⠃⢿⣿⣿⣿⣿⣿⣿⣿⡇⢐⠘⣿⣿⣿⣿",
* },
* {
* {26, 'f'},
* "27.⣿⣿⣿⣿⣿⣯⢀⠀⠈⢠⠋⠠⢱⢢⢠⠀⠂⠀⢠⠐⢸⣿⣿⣿⣿⣿⣿⣿⠿⡛⢯⢩⠀⢠⠀⢀⠘⢠⠀⠠⠃⣠⢲⡏⡮⠧⠴⣌⠳⣘⠈⠕⢀⢅⢊⡦⠈⢼⢸⣿⣿⣿⣿⣿⣿⣿⠃⠾⢀⣿⣿⣿⣿",
* },
* {
* {27, 'f'},
* "28.⣿⣿⣿⣿⣿⣿⠀⠐⠀⢀⠘⠬⠢⣱⠈⠠⠀⠀⡈⢀⣿⣿⣿⣿⣿⣿⣿⣿⢟⠉⢀⠤⢐⣴⣶⣷⣟⢸⢠⢀⠦⣱⡟⢖⠔⠣⢘⢚⢡⠐⠲⡰⡴⣡⠋⠐⡀⢸⣻⣿⣿⣿⣿⠿⡟⢅⢀⣼⣿⣿⣿⣿⣿",
* },
* {
* {28, 'f'},
* "29.⣿⣿⣿⣿⣿⣿⣷⠀⢠⢲⡠⠀⢉⡘⠠⠔⠰⢠⡾⣽⣿⣿⣿⣿⣿⣿⣿⢰⢋⢐⣥⣾⡿⣿⣽⣿⣿⡄⠈⡈⠙⠓⠱⠐⠲⠣⡅⠎⠲⠘⢈⢰⠈⣘⠈⠘⠱⠘⠸⠿⠋⠡⠒⢐⣨⣾⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {29, 'f'},
* "30.⣫⡉⣿⣿⣿⣿⣿⣷⣆⣨⣘⣃⠃⢠⣭⢈⡼⣯⣿⣿⣿⣿⣿⣿⣿⣿⣿⢀⠸⢸⢿⣷⣿⣿⣿⣿⣿⣧⠰⢤⠓⡙⠙⠻⠮⢱⣘⣐⠉⢊⠥⠴⠉⠁⠐⠀⠐⣐⣀⣤⣤⣦⣽⣤⣽⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {30, 'f'},
* "31.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣯⣽⣽⣿⣷⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⡔⠀⢈⠙⠛⠻⠛⠿⠿⠿⠎⡙⢚⡂⣠⠰⠺⠊⠈⠀⢐⡉⠑⢀⠀⢀⠀⠴⠡⢃⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⣛⠻",
* },
* {
* {31, 'f'},
* "32.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣶⣤⠬⢀⠈⠨⢤⣤⣡⠠⡄⡀⠀⡐⢐⡀⠀⣀⣐⣀⠠⠴⣋⠄⠯⣠⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣯",
* },
* {
* {32, 'f'},
* "33.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣧⣿⣋⣡⣒⣚⣸⣽⣥⣶⣦⣀⣀⠒⠞⠲⠒⠶⢌⣓⣨⣴⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {33, 'f'},
* "34.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {34, 'f'},
* "35.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {35, 'a'},
* "36.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {36, 'f'},
* "37.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢛⣵⢹⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {37, 'f'},
* "38.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢟⢔⢁⠜⡸⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {38, 'f'},
* "39.⣿⣿⣿⣿⣿⣿⡿⠟⠛⠙⢉⣙⠛⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣏⠩⠩⠭⠭⠑⠙⠐⠂⡂⢠⣿⣿⣿⣿⣿⣿⣿",
* },
* {
* {39, 'f'},
* "40.⣿⣿⣿⣿⡟⣩⠤⠀⠁⢠⡈⠉⣿⣶⣤⣽⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣿⣿⣮⢠⡬⠂⠐⠀⠙⢠⡁⠌⠙⢿⣿⣿⣿⣿",
* },
* {
* {40, 'f'},
* "41.⣿⣿⣿⠏⣴⠋⠀⠀⣤⣸⢕⠂⠀⣴⢮⣙⢷⣻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣼⣿⣿⣿⣿⣿⡾⣟⣿⣻⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢋⣐⣱⠤⢲⣺⢀⣰⣾⣶⣴⣾⣿⣿⣿⣿",
* },
* {{41, 'h'}, "42.⣿⣿⣿⠰⡇⢰⣤⡄⠉⠑⡡⠞⣶⣉⢶⠌⢁⠉⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢟⣋⢲⣱⣸⣤⣿⢻⡜⣙⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣨⣶⣿⣿⣀⢻⡄⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{42, 'h'}, "43.⣿⣿⣿⠈⣛⠈⠉⠴⢀⠘⡓⢞⣩⢟⢈⢰⢟⡔⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⠯⢦⡿⣏⡷⣿⢿⣾⢻⡝⡷⠿⠾⠹⣯⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣁⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{43, 'h'}, "44.⣿⣿⣿⣦⣷⠋⣈⣁⣵⣊⢲⢛⡱⠀⣠⠔⣛⣹⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⡿⢦⢂⡿⣱⣏⣞⡥⡏⠄⣩⣜⠊⠉⠽⠟⣈⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{44, 'h'}, "45.⣿⣿⣿⣿⣆⢛⣉⣙⢘⣂⣭⠔⠐⣑⣠⠾⣸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣹⡎⣾⢱⢷⡎⣿⢱⡎⣯⡻⣳⠎⢔⢮⢟⢣⡸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{45, 'h'}, "46.⣿⣿⣿⣿⣿⣷⣭⣛⠿⡷⣶⠿⡟⣯⣭⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣟⣮⡜⣫⢞⢾⣡⢷⠽⢿⠞⢐⣲⢈⢮⣾⢃⣷⢸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{46, 'h'}, "47.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣿⣷⣻⢙⠪⢷⡿⢬⣕⡁⣮⣿⠇⢠⢸⢟⣾⣿⢸⣿⣿⣿⣿⣯⠉⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{47, 'h'}, "48.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⢨⣜⢸⣮⠺⠌⢥⣼⡟⢱⣼⣜⢨⡾⢯⢃⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{48, 'h'}, "49.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣾⡿⣮⣍⠙⠰⢈⣟⠝⣴⣿⢉⣔⣯⣿⢹⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{49, 'h'}, "50.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣏⣠⡀⣌⢩⣜⠯⠷⠾⣙⣨⣴⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{50, 'h'}, "51.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣾⣿⣿⣿⣿⣿⣷⣤⣵⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{51, 'h'}, "52.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"},
* {{52, 'h'}, "53.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿"}};
* int arr\_size = sizeof(arr) / sizeof(elem);
* mix\_arr\_elem(arr, arr\_size);
* printf("Неотсортированный вид:\n");
* print\_arr\_elem(arr, arr\_size);
* hoar\_sort(arr, arr\_size);
* printf("Oтсортированный вид:\n");
* print\_arr\_elem(arr, arr\_size);
* key srch\_elem\_key;
* while (1)
* {
* printf("Введите ключ искомого элмента(example: \"int char\"): ");
* int c = getchar();
* if (c == EOF)
* {
* putchar('\n');
* break;
* }
* ungetc(c, stdin);
* int res\_scan = scanf("%d %c", &srch\_elem\_key.intg, &srch\_elem\_key.lit);
* fflush(stdin);
* while (getchar() != '\n')
* ;
* if (res\_scan != 2)
* {
* printf("Ключ введена неправильно\nCперва числовой потом литерный ключ\n");
* continue;
* }
* elem \*srch\_elem = bin\_search(arr, arr\_size, srch\_elem\_key);
* if (srch\_elem == NULL)
* printf("Элемент с таким ключом не найден: %d %c\n", srch\_elem\_key.intg, srch\_elem\_key.lit);
* else
* printf("{%d %c}: %s\n", srch\_elem->key.intg, srch\_elem->key.lit, srch\_elem->data);
* }
* return 0;
* }

borders\_stack.h

#pragma once

#include <stdbool.h>

typedef struct {

    int L;

    int R;

} borders;

typedef struct {

    borders \*buf;

    int head;

    int size;

    int cap;

} borders\_stack;

void borders\_stack\_init(borders\_stack \*v);

void borders\_stack\_destroy(borders\_stack \*v);

bool borders\_stack\_push(borders\_stack \*v, borders val);

borders borders\_stack\_pop(borders\_stack \*v);

bool borders\_stack\_is\_empty(borders\_stack \*v);

int borders\_stack\_get\_size(borders\_stack \*v);

borders borders\_stack\_top(borders\_stack \*v);

* borders\_stack.c:
* #include <stdbool.h>
* #include <malloc.h>
* #include "borders\_stack.h"
* #define MIN\_CAP 5 // > 0
* void borders\_stack\_init(borders\_stack \*v){
* v->size = 0;
* v->head = 0;
* v->cap = MIN\_CAP;
* v->buf = malloc(sizeof(borders) \* v->cap);
* }
* void borders\_stack\_destroy(borders\_stack \*v){
* free(v->buf);
* v->buf = NULL;
* v->head = 0;
* v->size = 0;
* v->cap = 0;
* }
* static bool increase(borders\_stack \*v){
* int new\_cap = 2 \* v->cap;
* borders\* tmp = realloc(v->buf, new\_cap \* sizeof(borders));
* if (tmp == NULL){
* return false;
* }
* v->buf = tmp;
* int old\_cap = v->cap;
* v->cap = new\_cap;
* if (v->head + v->size > old\_cap){
* for (int i = old\_cap; i >= v->head; i--){
* v->buf[i+old\_cap] = v->buf[i];
* }
* v->head += old\_cap;
* }
* return true;
* }
* static void decrease\_if\_possible(borders\_stack \*v){
* if (v->size < (v->cap/4)){
* int new\_cap = v->cap/2;
* if (new\_cap < MIN\_CAP){
* new\_cap = MIN\_CAP;
* }
* if (new\_cap >= v->cap){
* return ;
* }
* if ((v->head + v->size) > v->cap){
* for (int i = v->head - new\_cap; i < new\_cap; i++){
* v->buf[i] = v->buf[i + new\_cap];
* }
* v->head = v->head - new\_cap;
* } else if (v->head + v->size > new\_cap) {
* for (int i = 0; i < v->size; i++){
* v->buf[i] = v->buf[i + v->head];
* }
* v->head = 0;
* }
* borders \*tmp = realloc(v->buf, new\_cap \* sizeof(borders));
* v->buf = tmp;
* v->cap = new\_cap;
* }
* }
* bool borders\_stack\_push(borders\_stack \*v, borders val){
* if (v->size == v->cap){
* if(!increase(v)) {
* return false;
* printf("== false ==\n");
* }
* }
* v->buf[(v->size + v->head)%v->cap] = val;
* v->size++;
* return true;
* }
* borders borders\_stack\_pop(borders\_stack \*v){
* borders res = v->buf[(v->head+v->size-1)%v->cap];
* v->size--;
* decrease\_if\_possible(v);
* return res;
* }
* bool borders\_stack\_is\_empty(borders\_stack \*v){
* if (v->size == 0){
* return true;
* }
* return false;
* }
* int borders\_stack\_get\_size(borders\_stack \*v){
* return v->size;
* }
* borders borders\_stack\_top(borders\_stack \*v){
* return v->buf[(v->head+v->size-1)%v->cap];
* }

2.4 Входные и выходные данные. Тесты

quibex@STARK:~/labs/kp9$ make

gcc -c borders\_stack.c -o borders\_stack.o

gcc -c lesson.c -o lesson.o

gcc -c main.c -o main.o

gcc borders\_stack.o lesson.o main.o -o kp9

quibex@STARK:~/labs/kp9$ ./kp9

Неотсортированный вид:

40 f : 41.⣿⣿⣿⠏⣴⠋⠀⠀⣤⣸⢕⠂⠀⣴⢮⣙⢷⣻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣼⣿⣿⣿⣿⣿⡾⣟⣿⣻⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢋⣐⣱⠤⢲⣺⢀⣰⣾⣶⣴⣾⣿⣿⣿⣿

19 f : 20.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

12 f : 12.⣿⣿⣿⣿⣿⣿⣗⡀⠠⢈⠁⣰⠏⠀⠀⠐⠀⠈⣿⠀⢸⠀⣀⣤⣄⠈⠓⠖⠉⣀⠂⣼⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣶⣶⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

7 f : 8.⣿⣿⣿⣿⣿⣿⡏⠀⢐⠀⣼⣇⠀⣠⠞⢁⡄⠒⠂⡉⡉⡉⠙⢿⣋⣁⡼⠃⠀⡅⢂⠹⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⡇⢰⣵⠱⢛⣷⢾⢿⢞⣋⢗⠐⣨⣾⠇⡨⢱⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

45 h : 46.⣿⣿⣿⣿⣿⣷⣭⣛⠿⡷⣶⠿⡟⣯⣭⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣟⣮⡜⣫⢞⢾⣡⢷⠽⢿⠞⢐⣲⢈⢮⣾⢃⣷⢸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

26 f : 27.⣿⣿⣿⣿⣿⣯⢀⠀⠈⢠⠋⠠⢱⢢⢠⠀⠂⠀⢠⠐⢸⣿⣿⣿⣿⣿⣿⣿⠿⡛⢯⢩⠀⢠⠀⢀⠘⢠⠀⠠⠃⣠⢲⡏⡮⠧⠴⣌⠳⣘⠈⠕⢀⢅⢊⡦⠈⢼⢸⣿⣿⣿⣿⣿⣿⣿⠃⠾⢀⣿⣿⣿⣿

21 f : 22.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢿⢟⣛⢛⣛⣛⣛⠛⡻⠿⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

2 z : 3.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⢿⠻⠟⠻⠟⠿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

52 h : 53.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

24 f : 25.⣿⣿⣿⣿⣿⣿⡟⢀⠌⠠⣱⠴⠨⠲⢹⠀⠢⢻⣾⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠟⠘⠠⠀⢄⠂⠄⠀⠀⠐⢐⢨⢀⡐⠠⣐⠐⡀⠦⢰⡰⣶⣿⣿⣿⣷⣶⣤⡠⡄⢰⢹⣿⣿⣿⣿

44 h : 45.⣿⣿⣿⣿⣆⢛⣉⣙⢘⣂⣭⠔⠐⣑⣠⠾⣸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣹⡎⣾⢱⢷⡎⣿⢱⡎⣯⡻⣳⠎⢔⢮⢟⢣⡸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

3 f : 4.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠛⠉⠀⠁⣀⣀⠐⠒⠂⠆⣀⣀⠉⠛⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠛⢛⢉⡉⡉⢛⣛⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

20 f : 21.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

35 a : 36.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

31 f : 32.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣶⣤⠬⢀⠈⠨⢤⣤⣡⠠⡄⡀⠀⡐⢐⡀⠀⣀⣐⣀⠠⠴⣋⠄⠯⣠⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣯

12 f : 17.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣦⡄⢐⠀⢀⡀⠄⠺⢿⣿⣿⣿⣿

34 f : 35.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

36 f : 37.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢛⣵⢹⣿⣿⣿⣿⣿⣿⣿

18 f : 19.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣧⣴⣾⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿

4 f : 5.⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⠃⢀⠐⣠⠷⠛⠉⠉⠙⠲⣄⠀⡄⠈⠻⠆⠀⡘⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢛⠍⣀⢹⢤⣃⠹⣧⢢⣂⢙⠷⢌⠻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

1 d : 1.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢿⣿⣿⣿⣿⣿

38 f : 39.⣿⣿⣿⣿⣿⣿⡿⠟⠛⠙⢉⣙⠛⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣏⠩⠩⠭⠭⠑⠙⠐⠂⡂⢠⣿⣿⣿⣿⣿⣿⣿

30 f : 31.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣯⣽⣽⣿⣷⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⡔⠀⢈⠙⠛⠻⠛⠿⠿⠿⠎⡙⢚⡂⣠⠰⠺⠊⠈⠀⢐⡉⠑⢀⠀⢀⠀⠴⠡⢃⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⣛⠻

8 f : 9.⣿⣿⣿⣿⣿⣿⡁⠐⡊⢠⠀⣨⠟⢁⡴⠁⡀⠒⠂⠂⣀⡰⢦⡀⢨⠡⠀⠐⢂⠐⡁⠘⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣧⢠⠘⠊⣆⠟⣢⣋⡫⠛⠁⣐⣴⠟⣡⠆⠊⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

39 f : 40.⣿⣿⣿⣿⡟⣩⠤⠀⠁⢠⡈⠉⣿⣶⣤⣽⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣿⣿⣮⢠⡬⠂⠐⠀⠙⢠⡁⠌⠙⢿⣿⣿⣿⣿

10 f : 11.⣿⣿⣿⣿⣿⣿⡅⠀⢳⡀⠀⠀⣈⣤⠶⠒⠲⣆⢀⣡⡀⠉⡉⠀⡀⢾⡀⢀⢠⡇⠐⢠⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣌⡛⠶⠤⢭⠭⠭⠥⠾⣘⣩⣴⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

42 h : 43.⣿⣿⣿⠈⣛⠈⠉⠴⢀⠘⡓⢞⣩⢟⢈⢰⢟⡔⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⠯⢦⡿⣏⡷⣿⢿⣾⢻⡝⡷⠿⠾⠹⣯⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣁⣿⣿⣿⣿⣿⣿⣿⣿⣿

5 f : 6.⣿⣿⣿⣿⣿⣿⣿⣿⠋⠀⠐⣠⡞⠁⠀⠀⠀⠀⠀⣀⣟⣀⡀⢁⡤⢤⣆⠑⡄⠘⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠘⠁⠜⣠⣫⢴⢺⡵⢥⣕⡯⢾⠦⢱⠳⡜⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

43 h : 44.⣿⣿⣿⣦⣷⠋⣈⣁⣵⣊⢲⢛⡱⠀⣠⠔⣛⣹⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⡿⢦⢂⡿⣱⣏⣞⡥⡏⠄⣩⣜⠊⠉⠽⠟⣈⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

25 f : 26.⣿⣿⣿⣿⣿⣿⠇⠀⠀⠘⡡⠐⢀⢠⢞⠄⠀⠀⠹⠉⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⠛⢋⢀⠁⠀⠀⠀⢀⢀⣐⢼⣀⢆⡸⡄⢆⢑⠅⠸⢑⠰⢸⠃⢿⣿⣿⣿⣿⣿⣿⣿⡇⢐⠘⣿⣿⣿⣿

6 f : 7.⣿⣿⣿⣿⣿⣿⣿⠇⠀⡈⢀⡏⠀⠀⠀⣰⡶⠿⠏⠉⠉⠉⢹⡏⢀⣰⢿⡆⠰⠆⠈⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡁⠎⡰⣸⠡⢶⣏⢿⣸⠯⣎⣰⠟⠚⡀⠠⡇⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

12 f : 14.⣿⣿⣿⣿⣿⣿⣿⣿⣷⡐⢠⠀⠙⠲⠶⠖⠋⠁⠄⡀⠸⣇⠀⠀⣠⡟⠀⠂⣁⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⣛⣙⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣿⣿⣿⣿⣿⣿

32 f : 33.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣧⣿⣋⣡⣒⣚⣸⣽⣥⣶⣦⣀⣀⠒⠞⠲⠒⠶⢌⣓⣨⣴⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

12 f : 16.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣤⣄⣈⢉⡁⣁⢉⡁⣤⣴⣮⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠟⣛⡛⡛⠘⠀⠡⠩⠿⠝⢻⣿⣿⣿

51 h : 52.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

27 f : 28.⣿⣿⣿⣿⣿⣿⠀⠐⠀⢀⠘⠬⠢⣱⠈⠠⠀⠀⡈⢀⣿⣿⣿⣿⣿⣿⣿⣿⢟⠉⢀⠤⢐⣴⣶⣷⣟⢸⢠⢀⠦⣱⡟⢖⠔⠣⢘⢚⢡⠐⠲⡰⡴⣡⠋⠐⡀⢸⣻⣿⣿⣿⣿⠿⡟⢅⢀⣼⣿⣿⣿⣿⣿

46 h : 47.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣿⣷⣻⢙⠪⢷⡿⢬⣕⡁⣮⣿⠇⢠⢸⢟⣾⣿⢸⣿⣿⣿⣿⣯⠉⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

22 f : 23.⣿⣿⣯⣿⣿⣿⣿⣿⣿⣿⣿⡿⠿⢿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠟⢡⡾⠝⡈⠅⢈⢀⡉⢉⡛⣷⡚⡑⠬⠻⢿⠿⢿⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

23 f : 24.⣿⣿⣿⣿⣿⣿⣿⣿⢿⡋⠔⠈⣉⠽⢯⠽⢏⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢋⡴⢠⠆⠁⣀⠔⢈⠁⠆⠑⠀⠉⠡⠠⠘⠸⠀⠐⠈⠀⠤⠄⠄⢄⠠⠪⣙⠻⢿⣿⣿⣿⣿⣿

12 f : 15.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣦⣌⠐⠒⠄⣀⠁⠉⠄⠡⠄⠈⢙⠛⠁⢀⢤⣺⣽⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢡⠈⠿⢿⣿⣿⣿⣿⣿⣿

41 h : 42.⣿⣿⣿⠰⡇⢰⣤⡄⠉⠑⡡⠞⣶⣉⢶⠌⢁⠉⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢟⣋⢲⣱⣸⣤⣿⢻⡜⣙⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣨⣶⣿⣿⣀⢻⡄⣿⣿⣿⣿⣿⣿⣿⣿⣿

2 b : 2.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣤⣿⣿⣿⣿⣿

9 f : 10.⣿⣿⣿⣿⣿⣿⠀⠐⡆⠀⠁⠀⢀⡞⠀⠂⠂⠢⠄⡂⢸⣆⢀⡟⠀⢀⡶⠶⢤⡀⠑⣈⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡘⢶⣲⠈⣌⢀⣀⠀⠀⠈⢼⢐⠒⠐⢠⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

49 h : 50.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣏⣠⡀⣌⢩⣜⠯⠷⠾⣙⣨⣴⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

47 h : 48.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⢨⣜⢸⣮⠺⠌⢥⣼⡟⢱⣼⣜⢨⡾⢯⢃⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

28 f : 29.⣿⣿⣿⣿⣿⣿⣷⠀⢠⢲⡠⠀⢉⡘⠠⠔⠰⢠⡾⣽⣿⣿⣿⣿⣿⣿⣿⢰⢋⢐⣥⣾⡿⣿⣽⣿⣿⡄⠈⡈⠙⠓⠱⠐⠲⠣⡅⠎⠲⠘⢈⢰⠈⣘⠈⠘⠱⠘⠸⠿⠋⠡⠒⢐⣨⣾⣿⣿⣿⣿⣿⣿⣿

50 h : 51.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣾⣿⣿⣿⣿⣿⣷⣤⣵⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

33 f : 34.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

17 f : 18.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢠⠔⢠⣴⣄⣨⣔⠄⣿⣿⣿⣿

12 f : 13.⣿⣿⣿⣿⣿⣿⣿⣦⠀⠨⠀⢻⡀⠀⠀⠀⢀⣴⠏⠛⢧⡞⠁⢀⠈⢷⠀⠐⡀⠁⣼⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

37 f : 38.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢟⢔⢁⠜⡸⣿⣿⣿⣿⣿⣿⣿

29 f : 30.⣫⡉⣿⣿⣿⣿⣿⣷⣆⣨⣘⣃⠃⢠⣭⢈⡼⣯⣿⣿⣿⣿⣿⣿⣿⣿⣿⢀⠸⢸⢿⣷⣿⣿⣿⣿⣿⣧⠰⢤⠓⡙⠙⠻⠮⢱⣘⣐⠉⢊⠥⠴⠉⠁⠐⠀⠐⣐⣀⣤⣤⣦⣽⣤⣽⣿⣿⣿⣿⣿⣿⣿⣿

48 h : 49.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣾⡿⣮⣍⠙⠰⢈⣟⠝⣴⣿⢉⣔⣯⣿⢹⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

Oтсортированный вид:

1 d : 1.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢿⣿⣿⣿⣿⣿

2 b : 2.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣤⣿⣿⣿⣿⣿

2 z : 3.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⢿⠻⠟⠻⠟⠿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

3 f : 4.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠛⠉⠀⠁⣀⣀⠐⠒⠂⠆⣀⣀⠉⠛⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠛⢛⢉⡉⡉⢛⣛⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

4 f : 5.⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⠃⢀⠐⣠⠷⠛⠉⠉⠙⠲⣄⠀⡄⠈⠻⠆⠀⡘⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢛⠍⣀⢹⢤⣃⠹⣧⢢⣂⢙⠷⢌⠻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

5 f : 6.⣿⣿⣿⣿⣿⣿⣿⣿⠋⠀⠐⣠⡞⠁⠀⠀⠀⠀⠀⣀⣟⣀⡀⢁⡤⢤⣆⠑⡄⠘⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠘⠁⠜⣠⣫⢴⢺⡵⢥⣕⡯⢾⠦⢱⠳⡜⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

6 f : 7.⣿⣿⣿⣿⣿⣿⣿⠇⠀⡈⢀⡏⠀⠀⠀⣰⡶⠿⠏⠉⠉⠉⢹⡏⢀⣰⢿⡆⠰⠆⠈⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡁⠎⡰⣸⠡⢶⣏⢿⣸⠯⣎⣰⠟⠚⡀⠠⡇⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

7 f : 8.⣿⣿⣿⣿⣿⣿⡏⠀⢐⠀⣼⣇⠀⣠⠞⢁⡄⠒⠂⡉⡉⡉⠙⢿⣋⣁⡼⠃⠀⡅⢂⠹⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⡇⢰⣵⠱⢛⣷⢾⢿⢞⣋⢗⠐⣨⣾⠇⡨⢱⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

8 f : 9.⣿⣿⣿⣿⣿⣿⡁⠐⡊⢠⠀⣨⠟⢁⡴⠁⡀⠒⠂⠂⣀⡰⢦⡀⢨⠡⠀⠐⢂⠐⡁⠘⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣧⢠⠘⠊⣆⠟⣢⣋⡫⠛⠁⣐⣴⠟⣡⠆⠊⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

9 f : 10.⣿⣿⣿⣿⣿⣿⠀⠐⡆⠀⠁⠀⢀⡞⠀⠂⠂⠢⠄⡂⢸⣆⢀⡟⠀⢀⡶⠶⢤⡀⠑⣈⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡘⢶⣲⠈⣌⢀⣀⠀⠀⠈⢼⢐⠒⠐⢠⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

10 f : 11.⣿⣿⣿⣿⣿⣿⡅⠀⢳⡀⠀⠀⣈⣤⠶⠒⠲⣆⢀⣡⡀⠉⡉⠀⡀⢾⡀⢀⢠⡇⠐⢠⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣌⡛⠶⠤⢭⠭⠭⠥⠾⣘⣩⣴⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

12 f : 12.⣿⣿⣿⣿⣿⣿⣗⡀⠠⢈⠁⣰⠏⠀⠀⠐⠀⠈⣿⠀⢸⠀⣀⣤⣄⠈⠓⠖⠉⣀⠂⣼⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣶⣶⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

12 f : 15.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣦⣌⠐⠒⠄⣀⠁⠉⠄⠡⠄⠈⢙⠛⠁⢀⢤⣺⣽⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢡⠈⠿⢿⣿⣿⣿⣿⣿⣿

12 f : 16.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣤⣄⣈⢉⡁⣁⢉⡁⣤⣴⣮⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⠟⣛⡛⡛⠘⠀⠡⠩⠿⠝⢻⣿⣿⣿

12 f : 14.⣿⣿⣿⣿⣿⣿⣿⣿⣷⡐⢠⠀⠙⠲⠶⠖⠋⠁⠄⡀⠸⣇⠀⠀⣠⡟⠀⠂⣁⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⣛⣙⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣿⣿⣿⣿⣿⣿

12 f : 17.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣦⡄⢐⠀⢀⡀⠄⠺⢿⣿⣿⣿⣿

12 f : 13.⣿⣿⣿⣿⣿⣿⣿⣦⠀⠨⠀⢻⡀⠀⠀⠀⢀⣴⠏⠛⢧⡞⠁⢀⠈⢷⠀⠐⡀⠁⣼⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

17 f : 18.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢠⠔⢠⣴⣄⣨⣔⠄⣿⣿⣿⣿

18 f : 19.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣧⣴⣾⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿

19 f : 20.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

20 f : 21.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

21 f : 22.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢿⢟⣛⢛⣛⣛⣛⠛⡻⠿⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

22 f : 23.⣿⣿⣯⣿⣿⣿⣿⣿⣿⣿⣿⡿⠿⢿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠟⢡⡾⠝⡈⠅⢈⢀⡉⢉⡛⣷⡚⡑⠬⠻⢿⠿⢿⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

23 f : 24.⣿⣿⣿⣿⣿⣿⣿⣿⢿⡋⠔⠈⣉⠽⢯⠽⢏⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢋⡴⢠⠆⠁⣀⠔⢈⠁⠆⠑⠀⠉⠡⠠⠘⠸⠀⠐⠈⠀⠤⠄⠄⢄⠠⠪⣙⠻⢿⣿⣿⣿⣿⣿

24 f : 25.⣿⣿⣿⣿⣿⣿⡟⢀⠌⠠⣱⠴⠨⠲⢹⠀⠢⢻⣾⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠟⠘⠠⠀⢄⠂⠄⠀⠀⠐⢐⢨⢀⡐⠠⣐⠐⡀⠦⢰⡰⣶⣿⣿⣿⣷⣶⣤⡠⡄⢰⢹⣿⣿⣿⣿

25 f : 26.⣿⣿⣿⣿⣿⣿⠇⠀⠀⠘⡡⠐⢀⢠⢞⠄⠀⠀⠹⠉⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⠛⢋⢀⠁⠀⠀⠀⢀⢀⣐⢼⣀⢆⡸⡄⢆⢑⠅⠸⢑⠰⢸⠃⢿⣿⣿⣿⣿⣿⣿⣿⡇⢐⠘⣿⣿⣿⣿

26 f : 27.⣿⣿⣿⣿⣿⣯⢀⠀⠈⢠⠋⠠⢱⢢⢠⠀⠂⠀⢠⠐⢸⣿⣿⣿⣿⣿⣿⣿⠿⡛⢯⢩⠀⢠⠀⢀⠘⢠⠀⠠⠃⣠⢲⡏⡮⠧⠴⣌⠳⣘⠈⠕⢀⢅⢊⡦⠈⢼⢸⣿⣿⣿⣿⣿⣿⣿⠃⠾⢀⣿⣿⣿⣿

27 f : 28.⣿⣿⣿⣿⣿⣿⠀⠐⠀⢀⠘⠬⠢⣱⠈⠠⠀⠀⡈⢀⣿⣿⣿⣿⣿⣿⣿⣿⢟⠉⢀⠤⢐⣴⣶⣷⣟⢸⢠⢀⠦⣱⡟⢖⠔⠣⢘⢚⢡⠐⠲⡰⡴⣡⠋⠐⡀⢸⣻⣿⣿⣿⣿⠿⡟⢅⢀⣼⣿⣿⣿⣿⣿

28 f : 29.⣿⣿⣿⣿⣿⣿⣷⠀⢠⢲⡠⠀⢉⡘⠠⠔⠰⢠⡾⣽⣿⣿⣿⣿⣿⣿⣿⢰⢋⢐⣥⣾⡿⣿⣽⣿⣿⡄⠈⡈⠙⠓⠱⠐⠲⠣⡅⠎⠲⠘⢈⢰⠈⣘⠈⠘⠱⠘⠸⠿⠋⠡⠒⢐⣨⣾⣿⣿⣿⣿⣿⣿⣿

29 f : 30.⣫⡉⣿⣿⣿⣿⣿⣷⣆⣨⣘⣃⠃⢠⣭⢈⡼⣯⣿⣿⣿⣿⣿⣿⣿⣿⣿⢀⠸⢸⢿⣷⣿⣿⣿⣿⣿⣧⠰⢤⠓⡙⠙⠻⠮⢱⣘⣐⠉⢊⠥⠴⠉⠁⠐⠀⠐⣐⣀⣤⣤⣦⣽⣤⣽⣿⣿⣿⣿⣿⣿⣿⣿

30 f : 31.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣯⣽⣽⣿⣷⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⡔⠀⢈⠙⠛⠻⠛⠿⠿⠿⠎⡙⢚⡂⣠⠰⠺⠊⠈⠀⢐⡉⠑⢀⠀⢀⠀⠴⠡⢃⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡟⣛⠻

31 f : 32.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣶⣤⠬⢀⠈⠨⢤⣤⣡⠠⡄⡀⠀⡐⢐⡀⠀⣀⣐⣀⠠⠴⣋⠄⠯⣠⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣯

32 f : 33.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣧⣿⣋⣡⣒⣚⣸⣽⣥⣶⣦⣀⣀⠒⠞⠲⠒⠶⢌⣓⣨⣴⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

33 f : 34.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

34 f : 35.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

35 a : 36.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

36 f : 37.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢛⣵⢹⣿⣿⣿⣿⣿⣿⣿

37 f : 38.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢟⢔⢁⠜⡸⣿⣿⣿⣿⣿⣿⣿

38 f : 39.⣿⣿⣿⣿⣿⣿⡿⠟⠛⠙⢉⣙⠛⠻⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣏⠩⠩⠭⠭⠑⠙⠐⠂⡂⢠⣿⣿⣿⣿⣿⣿⣿

39 f : 40.⣿⣿⣿⣿⡟⣩⠤⠀⠁⢠⡈⠉⣿⣶⣤⣽⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣿⣿⣮⢠⡬⠂⠐⠀⠙⢠⡁⠌⠙⢿⣿⣿⣿⣿

40 f : 41.⣿⣿⣿⠏⣴⠋⠀⠀⣤⣸⢕⠂⠀⣴⢮⣙⢷⣻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣼⣿⣿⣿⣿⣿⡾⣟⣿⣻⠿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢋⣐⣱⠤⢲⣺⢀⣰⣾⣶⣴⣾⣿⣿⣿⣿

41 h : 42.⣿⣿⣿⠰⡇⢰⣤⡄⠉⠑⡡⠞⣶⣉⢶⠌⢁⠉⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢟⣋⢲⣱⣸⣤⣿⢻⡜⣙⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣨⣶⣿⣿⣀⢻⡄⣿⣿⣿⣿⣿⣿⣿⣿⣿

42 h : 43.⣿⣿⣿⠈⣛⠈⠉⠴⢀⠘⡓⢞⣩⢟⢈⢰⢟⡔⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⠯⢦⡿⣏⡷⣿⢿⣾⢻⡝⡷⠿⠾⠹⣯⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣷⣁⣿⣿⣿⣿⣿⣿⣿⣿⣿

43 h : 44.⣿⣿⣿⣦⣷⠋⣈⣁⣵⣊⢲⢛⡱⠀⣠⠔⣛⣹⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⡿⢦⢂⡿⣱⣏⣞⡥⡏⠄⣩⣜⠊⠉⠽⠟⣈⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

44 h : 45.⣿⣿⣿⣿⣆⢛⣉⣙⢘⣂⣭⠔⠐⣑⣠⠾⣸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣹⡎⣾⢱⢷⡎⣿⢱⡎⣯⡻⣳⠎⢔⢮⢟⢣⡸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

45 h : 46.⣿⣿⣿⣿⣿⣷⣭⣛⠿⡷⣶⠿⡟⣯⣭⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣟⣮⡜⣫⢞⢾⣡⢷⠽⢿⠞⢐⣲⢈⢮⣾⢃⣷⢸⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

46 h : 47.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣯⣿⣷⣻⢙⠪⢷⡿⢬⣕⡁⣮⣿⠇⢠⢸⢟⣾⣿⢸⣿⣿⣿⣿⣯⠉⢻⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

47 h : 48.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⢨⣜⢸⣮⠺⠌⢥⣼⡟⢱⣼⣜⢨⡾⢯⢃⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

48 h : 49.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣾⡿⣮⣍⠙⠰⢈⣟⠝⣴⣿⢉⣔⣯⣿⢹⣾⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

49 h : 50.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣏⣠⡀⣌⢩⣜⠯⠷⠾⣙⣨⣴⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

50 h : 51.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣾⣿⣿⣿⣿⣿⣷⣤⣵⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

51 h : 52.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

52 h : 53.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

Введите ключ искомого элмента(example: "int char"): 1 d

{1 d}: 1.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡿⢿⣿⣿⣿⣿⣿

Введите ключ искомого элмента(example: "int char"): 2 z

{2 z}: 3.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⢿⠻⠟⠻⠟⠿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

Введите ключ искомого элмента(example: "int char"): 52 h

{52 h}: 53.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

Введите ключ искомого элмента(example: "int char"): 4434 d

Элемент с таким ключом не найден: 4434 d

Введите ключ искомого элмента(example: "int char"): 3 e

Элемент с таким ключом не найден: 3 e

Введите ключ искомого элмента(example: "int char"): 12 f

{12 f}: 12.⣿⣿⣿⣿⣿⣿⣗⡀⠠⢈⠁⣰⠏⠀⠀⠐⠀⠈⣿⠀⢸⠀⣀⣤⣄⠈⠓⠖⠉⣀⠂⣼⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣶⣶⣶⣶⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

Введите ключ искомого элмента(example: "int char"): 12 a

Элемент с таким ключом не найден: 12 a

Введите ключ искомого элмента(example: "int char"): 2 z

{2 z}: 3.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⠿⢿⠻⠟⠻⠟⠿⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⢿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

Введите ключ искомого элмента(example: "int char"): 32 24

Элемент с таким ключом не найден: 32 2

Введите ключ искомого элмента(example: "int char"): 49 h

{49 h}: 50.⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣽⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣏⣠⡀⣌⢩⣜⠯⠷⠾⣙⣨⣴⡿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿

Введите ключ искомого элмента(example: "int char"):

# Заключение

В данной работе был реализован алгоритм быстрой сортировки Хоара, примененный с использованием стека вместо рекурсии. Это позволило эффективно сортировать массив данных, минимизируя затраты на память и повышая производительность.

В данной реализации была использована структура данных стека для хранения информации о подмассивах, которые ещё не были отсортированы. Вместо рекурсивного вызова функции, информация о каждом подмассиве добавлялась в стек, и цикл обрабатывал эти подмассивы до тех пор, пока стек не опустел. Таким образом, был достигнут итеративный подход к алгоритму сортировки Хоара.

Бинарный поиск является эффективным алгоритмом поиска в отсортированном массиве данных. Реализация бинарного поиска в таблице по ключу позволяет быстро находить нужные элементы и облегчает процесс поиска в больших объемах данных. Этот алгоритм может быть полезен, например, при поиске информации в базе данных или в упорядоченном списке.

Таким образом, в данной работе были успешно реализованы алгоритм быстрой сортировки Хоара с использованием стека вместо рекурсии и бинарный поиск в таблице по ключу. Эти алгоритмы позволяют эффективно сортировать и искать данные, обеспечивая оптимальную производительность и минимальное потребление ресурсов.

# Список использованных источников

1. Левитин А. В. Глава 4. Метод декомпозиции: Быстрая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 174—179. — 576 с. — ISBN 978-5-8459-0987-9
2. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 7. Быстрая сортировка // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — С. 198—219. — ISBN 5-8459-0857-4.