Истина слишком сложна, чтобы допустить что–либо, кроме приближений.

Джон фон Нейман

Алгоритмы и структуры данных–1

2024–2025 учебный год

SET 3. Домашняя работа

Стохастические алгоритмы. Сортировка линейных контейнеров

ноябрь

28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	
пн	вт	ср	чт	пт	сб	вс

создано командой преданных своему делу энтузиастов

Немного инструкций

Задания в рамках домашней работы SET 3 подразделяются на два блока:

1. *Блок A* «Аналитические задания» — задачи, связанные с разработкой и экспериментальным анализом стохастических алгоритмов, а также алгоритмов сортировки.

   > Решения заданий Блока A оформляются в письменном виде в любом удобном формате (LaTeX, текстовый документ, скан, изображение и др.) и загружаются для оценки в соответствующую форму раздела `SET 3. Домашняя работа` на странице курса в `LMS`. Обратите внимание на необходимость загрузки реализаций алгоритмов сортировки в соответствующие задачи `CODEFORCES`, которые помечены "i".

2. *Блок P* «Задания на разработку» — задачи, связанные с реализацией стохастических алгоритмов и алгоритмов сортировки.

   > Решения заданий Блока P загружаются в систему `CODEFORCES` и проходят автоматизированное тестирование. Для загрузки нужно перейти на `https://dsahse.contest.codeforces.com` и выбрать соответствующее соревнование. Доступ к соревнованию предоставлен по тем же учетным данным, что и к системе Яндекс.Контест.

Домашняя работа SET 3 содержит 2 обязательные задачи в Блоке A и 6 обязательных задач в блоке P. Баллы, которые можно получить за решение задач, распределены следующим образом:

Блок A			**Блок P**					
A1	A2	A3b	P1	P2	P3	P4	P5	P6
15	25	20	5	6	6	5	7	7

Задачи, помеченные "b" не являются обязательными — баллы за их решение относятся к *бонусным*. Подтверждение решений бонусных задач сопровождается обязательной *устной защитой*.

Важные даты

1. Домашняя работа SET 3 открыта с **14:30 11 ноября 2024 г**.
2. Прием решений на оценку завершается в **02:00 25 ноября 2024 г**.
3. Загруженное решение бонусной задачи A3b должно быть защищено до **6 декабря 2024 г**.

Содержание

Успехов!

Задача A1. Задача трёх кругов

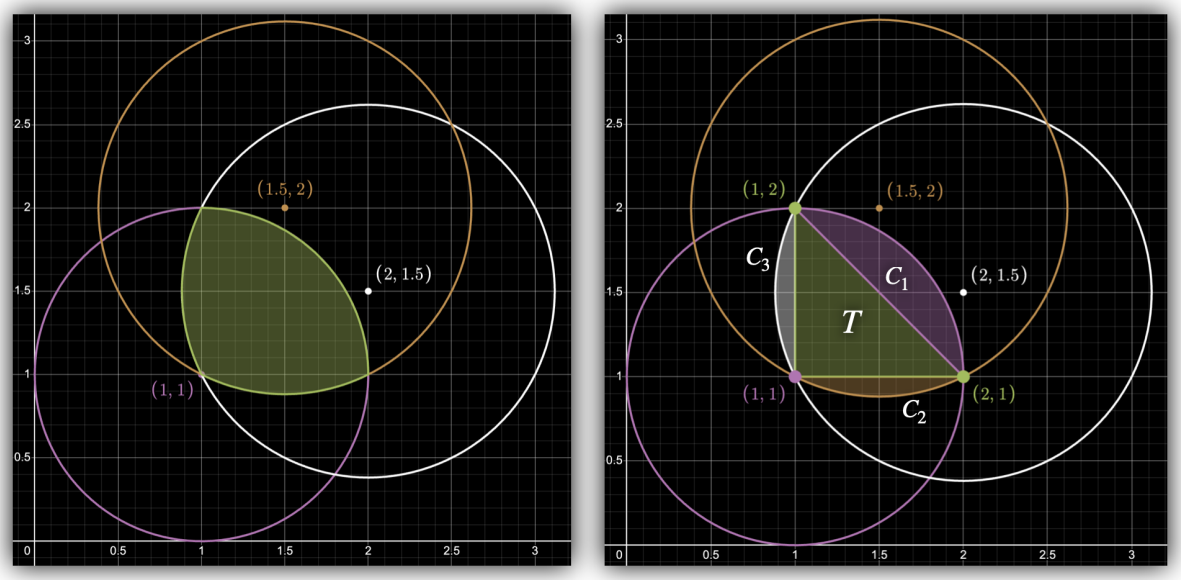
Среди множества возможных применений стохастического метода Монте-Карло особенно выделяется приближенная оценка площадей самых разных геометрических фигур, к которым в том числе относятся фигуры, образуемые пересечением кругов.

Постановка задачи

Нам даны три окружности:

- с центром в точке **(1, 1)** и радиусом **1**,
- с центром в точке **(1.5, 2)** и радиусом $\sqrt{5}\,/\,2$ и
- с центром в точке **(2, 1.5)** и радиусом $\sqrt{5}\,/\,2$.

В рамках этой задачи требуется вычислить приближенное значение площади фигуры, образованной в результате пересечения соответствующих кругов (см. зеленую область на левом рисунке ниже). Кроме того, необходимо оценить, насколько приближенная оценка площади *отклоняется* от ее точного значения в зависимости от параметров работы алгоритма Монте-Карло.



Точное вычисление площади пересечения кругов

Вывод общей формулы для вычисления площади пересечения кругов — весьма сложная задача, поэтому для удобства вычислений мы разобьем целевую фигуру на прямоугольный треугольник и три круговых *сегмента*, как показано выше на правом рисунке, а именно на:

- прямоугольный треугольник T с вершинами в точках **(1, 1)**, **(1, 2)** и **(2, 1)**;
- сегмент C_1, ограниченный гипотенузой T и кругом с центром в **(1, 1)**;
- сегменты C_2 и C_3, ограниченные катетами T и кругами с центрами в **(1.5, 2)** и **(2, 1.5)**.

Площадь кругового сегмента равна $\dfrac{\theta - \sin(\theta)}{2} \cdot r^2$, где r — радиус, а θ — величина соответствующего центрального угла (в радианах).

Нетрудно заметить, что площади круговых сегментов C_2 и C_3 совпадают. Поэтому общая площадь фигуры, образованной пересечением заданных кругов, составит $S = S_T + S_{C_1} + 2S_{C_2} = S_T + S_{C_1} + 2S_{C_3}$. Рассмотрим вычисление площади каждого компонента S в отдельности:

1. Площадь прямоугольного треугольника T с единичными катетами равна 0.5.

2. Центральный угол, который образует сегмент C_1, составляет $90° — \pi / 2$ радиан. Тогда: $S_{C_1} = \dfrac{\pi / 2 - \sin(\pi / 2)}{2} \cdot 1^2 = 0.25 \cdot \pi - 0.5$.

3. Синус центрального угла, который образует сегмент C_2 (C_3), составляет 0.8. Его можно найти по теореме косинусов для треугольника с вершинами **(1, 1)**, **(1.5, 2)** и **(2, 1)**. Тогда: $2 \cdot S_{C_2} = 2 \cdot S_{C_3} = 2 \cdot \dfrac{\arcsin(0.8) - 0.8}{2} \cdot \left(\dfrac{\sqrt{5}}{2}\right)^2 = 1.25 \cdot \arcsin(0.8) - 1$.

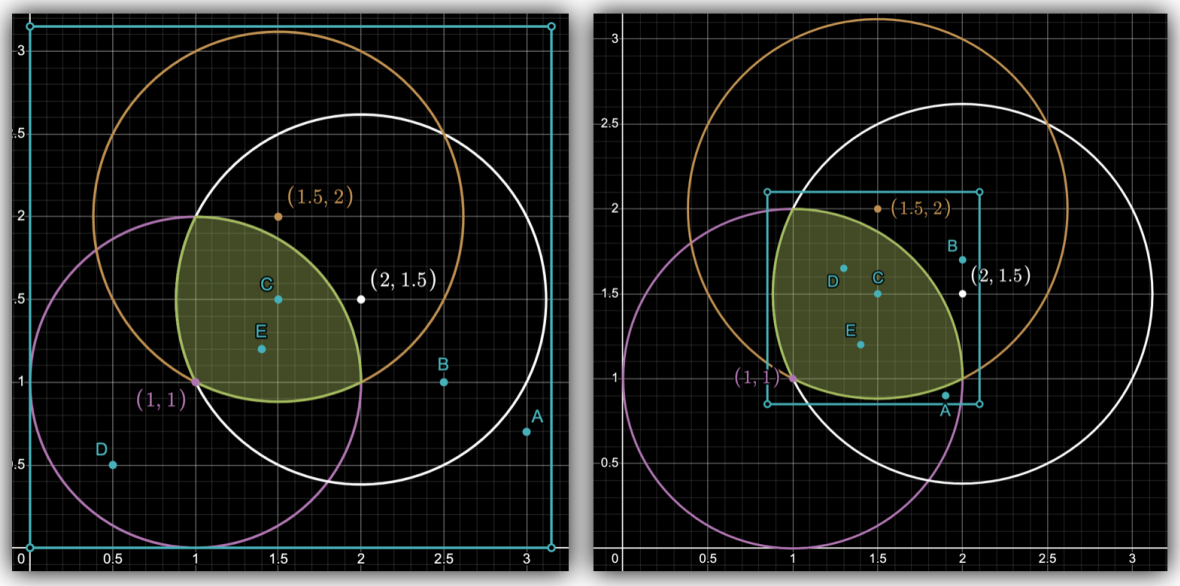
Итак, точная площадь фигуры пересечения трех заданных окружностей составляет:

$$\boxed{S = S_T + S_{C_1} + 2S_{C_2} = 0.5 + 0.25 \cdot \pi - 0.5 + 1.25 \cdot \arcsin(0.8) - 1 = 0.25 \cdot \pi + 1.25 \cdot \arcsin(0.8) - 1}$$

Приближенное вычисление площади пересечения кругов

Вычислить оценку площади фигуры, образованной пересечением трех заданных кругов, с помощью метода Монте-Карло можно как минимум двумя способами:

- путем случайной генерации точек в *широкой* прямоугольной области, которая охватывает все три круга полностью (см. на левом рисунке ниже) и
- путем случайной генерации точек в *узкой* прямоугольной области, которая более «плотно» ограничивает пересечение трех кругов (см. на правом рисунке ниже).



В обоих случаях, приближенная оценка отношения площади S целевой фигуры пересечения трех кругов к площади S_{rec} прямоугольной области составит $S / S_{rec} \approx M / N$, где N — общее число сгенерированных точек в рассматриваемой прямоугольной области, а M — число точек, которые попадают внутрь и на границу фигуры пересечения трех кругов. Таким образом, приближенная оценка площади пересечения трех кругов составит $\widetilde{S} = (M / N) \cdot S_{rec}$.

1. Реализуйте алгоритм Монте-Карло на основе случайной генерации точек в заданной прямоугольной области для приближенного вычисления площади пересечения трех кругов, заданных координатами центров и радиусами.

2. Проведите экспериментальные замеры точности вычисления площади фигуры, *рассмотренной в задаче*, в зависимости от масштаба прямоугольной области для случайной генерации точек, а также от количества случайно сгенерированных точек N, которое изменяется от **100** до **100000** с шагом **500**. Представьте результаты проведенных экспериментов в следующем виде:
   - график(-и) первого типа, которые отображают, как меняется приближенное значение площади в зависимости от указанных параметров алгоритма;
   - график(-и) второго типа, которые отображают, как меняется величина относительного отклонения приближенного значения площади от ее точной оценки в зависимости от указанных параметров алгоритма.
3. Опишите полученные вами результаты и сформулируйте содержательные выводы.

Язык программирования, который должен использоваться при реализации алгоритмов, — `C++`. Ограничений на используемые средства обработки и визуализации эмпирических данных нет. Помимо графиков и пояснений, приложите:

1. ID своей посылки по задаче `A1i` в системе `CodeForces` с реализацией алгоритма Монте-Карло.
2. Ссылку на публичный репозиторий с исходными данными, полученными в результате экспериментальных замеров.

Система оценки

1. 5 баллов Реализация алгоритма Монте-Карло для приближенного вычисления площади фигуры, образованной пересечением трех кругов.
2. 6 баллов Представление экспериментальных результатов работы алгоритма в зависимости от параметров его работы.
3. 4 балла Сравнительный анализ полученных эмпирических данных.

Обратите внимание, что загрузка реализации алгоритма в задачу `A1i` является *необходимым* условием для получения оценки по другим критериям.

Задача A1i. Задача трех кругов — реализация

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Эта подзадача *сопутствует* основной задаче `A1`.

Загрузите реализацию алгоритма Монте-Карло оценки площади фигуры, образованной пересечением трех кругов, которые заданы координатами центров и радиусами соответствующих окружностей.

Формат входных данных

Три строки, каждая содержит по три вещественных числа x_i, y_i, r_i, где $0 \leqslant x_i \leqslant 1$ и $0 \leqslant y_i \leqslant 1$ — координаты центра окружности, а r_i — ее радиус.

Формат выходных данных

Выведите одно число — приближенную оценку площади пересечения трех заданных кругов с абсолютной погрешностью, не превышающей 0.01.

Примеры

стандартный ввод	стандартный вывод
0.0 0.0 1.0 1.0 1.0 1.0 0.0 1.0 1.0	0.44380799999999998029
0.3 0.3 1 0.3 0.3 0.8 0.3 0.3 0.9	2.01463200000000020040

Замечание

В этой задаче вам нужно внимательно подойти к выбору общего числа генерируемых точек, ориентируясь на значения, приведенные в примере, и требуемую погрешность.

Задача A2. Анализ MERGE+INSERTION SORT

Известно, что алгоритм `MERGE SORT` является одним из оптимальных алгоритмов сортировки на основе сравнений элементов и имеет сложность $O(n \cdot \log n)$, а алгоритм `INSERTION SORT` имеет сложность $O(n^2)$. Однако на массивах сравнительно небольшого размера `INSERTION SORT` сортирует быстрее `MERGE SORT` ввиду лучшей оценки скрытой константы, а также снижения накладных расходов, которые связаны с рекурсией.

В рамках этой задачи требуется провести экспериментальное исследование двух реализаций алгоритма `MERGE SORT`:

- стандартной рекурсивной (с выделением дополнительной памяти) и
- гибридной, которая на массивах малого размера переключается на `INSERTION SORT`.

Язык программирования, который должен использоваться при реализации алгоритмов и проведении замеров времени их работы, — `C++`. Ограничений на используемые средства обработки и визуализации эмпирических данных нет.

Этап 1. Подготовка тестовых данных

Реализуйте класс `ArrayGenerator` для генерации тестовых массивов, заполненных целыми числами, со следующими характеристиками:

1. Массивы, которые заполнены случайными значениями в некотором диапазоне.
2. Массивы, которые отсортированы в обратном порядке по невозрастанию.
3. Массивы, которые «почти» отсортированы. Их можно получить, обменяв местами небольшое количество пар элементов в полностью отсортированном массиве.

В рамках задачи зафиксируем следующие параметры для подготовки тестовых данных:

- размеры массивов — от **500** до **10000** с шагом **100** и
- диапазон случайных значений — от **0** до **6000**.

Для удобства подготовки тестовых данных сгенерируйте для каждого случая массив максимальной длины (**10000**), из которого выбирайте подмассив необходимого размера (**500**, **600**, **700**, ... элементов).

Этап 2. Эмпирический анализ стандартного алгоритма `MERGE SORT`

Проведите замеры времени работы стандартной реализации алгоритма `MERGE SORT` и представьте результаты в виде трех (групп) графиков для каждой категории тестовых данных. Замеры времени работы удобнее всего производить с помощью встроенной библиотеки `std::chrono`. Например:

```
auto start = std::chrono::high_resolution_clock::now();
mergeSort(A, l, r);
auto elapsed = std::chrono::high_resolution_clock::now() - start;
long long msec = std::chrono::duration_cast<std::chrono::milliseconds>(elapsed).count();
```

При выполнении эмпирического анализа обратите внимание на:

- минимизацию влияния работы других программ и сетевого подключения,
- необходимость многократных замеров времени работы и усреднения результатов — количество замеров и способ усреднения остаётся на ваше усмотрение,
- выбор единиц измерения времени — единицы измерения слишком большого масштаба (например, секунды) приведут к получению плохо интерпретируемых результатов.

Этап 3. Эмпирический анализ гибридного алгоритма `MERGE+INSERTION SORT`

Проведите аналогичные предыдущему этапу замеры времени работы гибридной реализации алгоритма `MERGE+INSERTION SORT` и представьте результаты в виде трех (групп) графиков для каждой категории подготовленных тестовых данных.

Диапазон параметра переключения (`threshold`) на алгоритм `INSERTION SORT` остаётся полностью на ваше усмотрение. Например, можно рассмотреть переключение на `INSERTION SORT` для массивов, состоящих из 5, 10, 20, 30 и 50 элементов.

Функции эмпирического замера времени работы рассматриваемых алгоритмов сортировки реализуйте в отдельном классе `SortTester`.

Этап 4. Сравнительный анализ

Опишите полученные вами результаты и представьте *содержательные* выводы о сравнении временных затрат двух рассмотренных реализаций алгоритма сортировки слиянием. При выполнении сравнительного анализа, среди прочего, можно обратить внимание на поиск порогового значения параметра переключения в гибридном алгоритме `MERGE+INSERTION SORT`, начиная с которого он работает медленнее стандартной реализации `MERGE SORT`.

Помимо графиков и пояснений, приложите:

1. Реализацию класса `ArrayGenerator` и `SortTester`.
2. ID своей посылки по задаче `A2i` в системе `CodeForces` с реализацией `MERGE+INSERTION SORT`.
3. Ссылку на публичный репозиторий с исходными данными, полученными в результате эмпирических замеров времени работы алгоритмов.

Система оценки

1. 5 баллов Реализация гибридного алгоритма сортировки `MERGE+INSERTION SORT`.
2. 6 баллов Реализация внутренней инфраструктуры для экспериментального анализа — классы `ArrayGenerator` и `SortTester`.
3. 7 баллов Представление эмпирических замеров времени работы рассматриваемых алгоритмов.
4. 7 баллов Сравнительный анализ полученных эмпирических данных.

Обратите внимание, что загрузка реализации алгоритма `MERGE+INSERTION SORT` в задачу `A2i` является *необходимым* условием для получения оценки по другим критериям.

Задача A2i. MERGE+INSERTION SORT — реализация

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	5 секунд
Ограничение по памяти:	256 мегабайт

Эта подзадача *сопутствует* основной задаче `A2`.

Загрузите реализацию гибридного алгоритма `MERGE+INSERTION SORT` для сортировки целочисленного массива.

Параметр переключения на `INSERTION SORT` принимаем равным 15.

Формат входных данных

В первой строке входных данных содержится одно целое число n ($0 \leqslant n \leqslant 10^6$) — размер массива.

Во второй строке записано n целых чисел, каждое из которых по модулю не превосходит 10^9.

Формат выходных данных

Выведите элементы отсортированного массива через пробел.

Система оценки

Подзадача	Характеристика массивов	Необходимые подзадачи	Информация о проверке
0	тест из условия	–	полная
1	случайные	0	первая ошибка
2	отсортированные	0–1	первая ошибка
3	обратно отсортированные	0–2	первая ошибка

Пример

стандартный ввод	стандартный вывод
3 1 7 9	1 7 9

Замечание

Эта задача нацелена *только на проверку работоспособности* разработанного алгоритма.

Задача A3b. Анализ INTROSORT

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Интроспективная сортировка (`INTROSORT`) — это один из лучших среди известных алгоритмов сортировки, который сочетает в себе достоинства трех алгоритмов `QUICK SORT`, `HEAP SORT` и `INSERTION SORT`, основанных на сравнениях. Известно, что худший случай времени работы алгоритма `QUICK SORT` — $O(N^2)$. Для того чтобы избежать его возникновения, в процессе сортировки выполняется переключение на `HEAP SORT` и `INSERTION SORT` по следующим условиям:

1. Если текущая глубина рекурсии достигла $2 \cdot \log_2 N$, то выполняется переключение с `QUICK SORT` на `HEAP SORT`.

2. Если в сортируемой части массива осталось менее 16 элементов, то выполняется переключение с `QUICK SORT` на `INSERTION SORT`.

Во-первых, использование `HEAP SORT` обуславливается сокращением стека рекурсивных вызовов алгоритма `QUICK SORT`, размер которого в худшем случае может достигать $O(N)$. Во-вторых, `INSERTION SORT` является наиболее оптимальным алгоритмом сортировки массивов небольшого размера. Указанные выше значения параметров установлены в результате многочисленных исследований.

В рамках этой задачи требуется провести экспериментальное исследование двух реализаций алгоритма `QUICK SORT`:

- стандартной рекурсивной (со случайным выбором опорного элемента) и
- гибридной, которая в соответствии с представленными выше правилами переключается на `HEAP SORT` и `INSERTION SORT`.

Язык программирования, который должен использоваться при реализации алгоритмов и замеров времени их работы, — `C++`. Ограничений на используемые средства обработки и визуализации эмпирических данных нет.

Этап 1. Подготовка тестовых данных

Используйте те же тестовые данные, которые были подготовлены при решении задачи A2.

Этап 2. Эмпирический анализ стандартного алгоритма `QUICK SORT`

Проведите замеры времени работы стандартной рекурсивной реализации алгоритма быстрой сортировки со случайным выбором опорного элемента и представьте результаты в виде трех (групп) графиков для каждой категории подготовленных тестовых данных.

Этап 3. Эмпирический анализ гибридного алгоритма `QUICK+HEAP+INSERTION SORT`

Проведите аналогичные предыдущему этапу замеры времени работы гибридной реализации алгоритма `QUICK+HEAP+INSERTION SORT` и представьте результаты в виде трех (групп) графиков для каждой категории подготовленных тестовых данных.

Этап 4. Сравнительный анализ

Опишите полученные вами результаты и представьте *содержательные* выводы о сравнении временных затрат стандартной и гибридной реализации алгоритма быстрой сортировки.

Помимо графиков и пояснений, приложите:

1. ID посылки по задаче `A3i` в системе `CodeForces` с гибридной реализацией алгоритма `QUICK+HEAP+INSERTION SORT`.

2. Ссылку на публичный репозиторий с исходными данными, полученными в результате эмпирических замеров времени работы алгоритмов.

Система оценки

1. 7 баллов Реализация гибридного алгоритма сортировки `QUICK+HEAP+INSERTION SORT`.
2. 6 баллов Представление эмпирических замеров времени работы рассматриваемых алгоритмов.
3. 7 баллов Сравнительный анализ полученных эмпирических данных.

Обратите внимание, что загрузка реализации алгоритма `QUICK+HEAP+INSERTION SORT` в задачу `A3i` является *необходимым* условием для получения оценки по другим критериям.

Задача A3i. INTROSORT — реализация

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	5 секунд
Ограничение по памяти:	256 мегабайт

Эта задача *сопутствует* основной задаче A3b. Загрузите реализацию гибридного алгоритма `QUICK+HEAP+INSERTION SORT` для сортировки целочисленного массива A.

Напомним параметры работы этого алгоритма:

1. Переключение с `QUICK SORT` на `HEAP SORT` происходит при достижении глубины рекурсии, равной $2 \cdot \log_2 |A|$.
2. Переключение с `QUICK SORT` на `INSERTION SORT` происходит, когда сортируемая часть исходного массива содержит менее 16 элементов.

Формат входных данных

В первой строке входных данных содержится одно целое число n — размер массива, который не превосходит 10^6.

Во второй строке записано n целых чисел, каждое из которых по модулю не превосходит 10^9.

Формат выходных данных

Выведите элементы отсортированного массива через пробел.

Система оценки

Подзадача	Характеристика массивов	Необходимые подзадачи	Информация о проверке
0	тест из условия	–	полная
1	случайные	0	первая ошибка
2	отсортированные	0–1	первая ошибка
3	обратно отсортированные	0–2	первая ошибка

Пример

стандартный ввод	стандартный вывод
3 1 7 9	1 7 9

Замечание

Эта задача нацелена *только на проверку работоспособности* разработанного алгоритма.

Задача P1. Пирамидальная сортировка Ганменов

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

В мире, где люди сражаются на гигантских механических существах — Ганменах — *Симон* и его команда столкнулись с новой угрозой. Для того, чтобы эффективно координировать действия огромного флота Ганменов в предстоящей битве, на первом этапе необходимо упорядочить Ганменов по размеру. Симон обратился к вам, чтобы вы реализовали алгоритм сортировки `HEAP SORT`, который позволит оптимально распределить Ганменов по боевым порядкам.

Реализация `HEAP SORT` должна содержать три ключевые функции (A — одномерный целочисленный массив размеров Ганменов, i — целое положительное число):

1. `heapify(A, i)` — восстановление свойств max–кучи для элемента, стоящего по индексу i.
2. `buildMaxHeap(A)` — построение max–кучи из исходного целочисленного массива.
3. `heapSort(A)` — сортировка по неубыванию.

Формат входных данных

В первой строке входных данных содержится одно целое число n — размер массива, который не превосходит 10^6.

Во второй строке записано n целых чисел. Размеры Ганменов по модулю не превосходят 10^9.

Формат выходных данных

Выведите упорядоченные размеры Ганменов через пробел.

Система оценки

Подзадача	Баллы	Дополнительные ограничения	Необходимые подзадачи	Информация о проверке
0	–	тест из условия	–	полная
1	1	$n \leqslant 100$	0	первая ошибка
2	2	$n \leqslant 10^5$	0–1	первая ошибка
3	2	$n \leqslant 10^6$	0–2	первая ошибка

Примеры

стандартный ввод	стандартный вывод
1 913	913
6 555555 44444 3333 222 11 0	0 11 222 3333 44444 555555

Замечание

Посылки с реализациями других алгоритмов сортировки будут проигнорированы.

Задача P2. Сортировка звезд подсчетом

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Команде *Гуррен-Дан* предстоит отправиться в глубины космоса, чтобы выступить против *Антиспиральщиков* и их короля, *Лоргенома*. Механику *Лиирону Литтнеру* нужно проложить оптимальный маршрут сквозь звезды, которые должны быть отсортированы в порядке неубывания расстояния до главного корабля.

Лиирон обратился к вам, чтобы вы помогли ему разработать алгоритм сортировки `COUNTING SORT`. Реализацию сортировки представьте в виде отдельной функции `countingSort(A)`, где A - это исходный массив целочисленных расстояний.

Формат входных данных

В первой строке входных данных содержится одно целое число n — размер массива, который не превосходит 10^7.

Во второй строке записано n целых чисел. Расстояния от главного корабля до звезд по модулю не превосходят 10^6.

Формат выходных данных

Выведите упорядоченные расстояния до звезд через пробел.

Система оценки

Подзадача	Баллы	Дополнительные ограничения	Необходимые подзадачи	Информация о проверке
0	–	тесты из условия	–	полная
1	1	$n \leqslant 10^4$	0	первая ошибка
2	2	$n \leqslant 10^5$	0–1	первая ошибка
3	3	$n \leqslant 10^7$	0–2	первая ошибка

Примеры

стандартный ввод	стандартный вывод
8 17 88 22 96 27 65 91 35	17 22 27 35 65 88 91 96
3 75 36 68	36 68 75

Замечание

Посылки с реализациями других алгоритмов сортировки будут проигнорированы.

Задача P3. Поразрядная сортировка зверолюдей

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Симон и его команда, после победы над *зверолюдьми* и возрождения цивилизации, решили адаптировать оставшихся представителей этой расы для жизни в человеческом обществе. Эффективная ассимиляция требует сортировки зверолюдей по их характеристикам в порядке неубывания.

Помогите Симону выполнить эту задачу с помощью алгоритма `RADIX SORT` по основанию 256. Реализацию сортировки представьте в виде отдельной функции `radixSort(A)`, где A - это исходный массив целочисленных характеристик зверолюдей.

Формат входных данных

В первой строке входных данных содержится одно целое число n — размер массива, который не превосходит 10^6.

Во второй строке записано n целых чисел. Целочисленные характеристики зверолюдей по модулю не превосходят 10^9.

Формат выходных данных

Выведите упорядоченные характеристики зверолюдей через пробел.

Система оценки

Подзадача	Баллы	Дополнительные ограничения	Необходимые подзадачи	Информация о проверке
0	–	тесты из условия	–	полная
1	1	$n \leqslant 100$	0	первая ошибка
2	2	$n \leqslant 10^5$	0–1	первая ошибка
3	3	$n \leqslant 10^6$	0–2	первая ошибка

Примеры

стандартный ввод	стандартный вывод
5 6 32 13 83 58	6 13 32 58 83
1 10	10

Замечание

Посылки с реализациями других алгоритмов сортировки будут проигнорированы.

Задача P4. И снова сортировка Ганменов

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

В команде Гуррен-Дан разработали новый способ описания характеристик Ганменов для их последующего распределения по боевым порядкам. Каждому Ганмену сопоставлена обыкновенная дробь p/q, где p и q — целые положительные числа.

Разработайте для Симона и его команды алгоритм *устойчивой* сортировки последовательности дробей по неубыванию — порядок Ганменов, которым соответствуют одинаковые дроби, должен остаться таким же, как и во входных данных.

Формат входных данных

Первая строка входных данных содержит одно целое число N ($1 \leqslant N \leqslant 10^5$).

Вторая строка содержит N дробей a_i. Каждая дробь представляет собой два целых числа от 1 до 10^4, записанных через знак /. В записи дроби пробела нет, дроби разделены между собой пробелами.

Формат выходных данных

Выведите тот же самый набор дробей, отсортированный по неубыванию их величины. Дроби выводить в том же формате, что и во вводе (два числа через / без пробелов). Сокращать дроби *не нужно*.

Система оценки

Подзадача	Баллы	Дополнительные ограничения	Необходимые подзадачи	Информация о проверке
0	–	тест из условия	–	полная
1	5	$n \leqslant 10^5$	0	первая ошибка

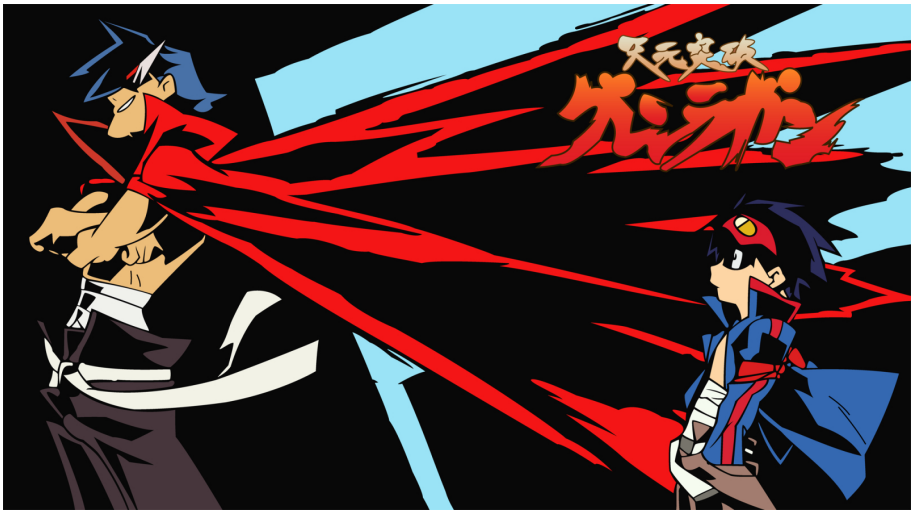
Пример

стандартный ввод	стандартный вывод
5 1/2 2/6 9/4 12/1 10/24	2/6 10/24 1/2 9/4 12/1

Задача P5. Симон и Камина умножают матрицы

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	3 секунды
Ограничение по памяти:	256 мегабайт

В свободное время Симон совершенствует не только технику управления Ганменов, но и математические навыки. Сейчас он учится перемножать большие бинарные квадратные матрицы в $\mathbb{F}_2$ — все арифметические операции производятся по модулю два.



Камина, его старший соратник, проверяет за ним каждый результат C перемножения двух матриц A и B размера $n \times n$ и утверждает, что в большинстве случаев Симон ошибается в вычислениях. Помогите им понять, кто прав.

Формат входных данных

На первой строке число n — размер матриц ($1 \leqslant n \leqslant 4000$).

Следующие три строки содержат описания матриц A, B и C.

Каждая матрица описывается строкой, содержащей n блоков размера $\lceil n/4 \rceil$ шестнадцатеричных цифр. Если записать цифры в двоичной записи в данном порядке от старших цифр к младшим и обрезать лишние цифры в конце строки, получится очередная строка матрицы. Например, матрица

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

представляется строкой «28 A8 68 78 D0 88».

Формат выходных данных

Выведите «YES», если $AB = C$, иначе «NO».

Система оценки

Подзадача	Баллы	Дополнительные ограничения	Необходимые подзадачи	Информация о проверке
0	–	тест из условия	–	полная
1	1	$n \leqslant 100$	0	первая ошибка
2	2	$n \leqslant 500$	0–1	первая ошибка
3	4	$n \leqslant 4000$	0–2	первая ошибка

Пример

стандартный ввод	стандартный вывод
6 28 A8 68 78 D0 88 80 40 20 10 08 04 28 A8 68 78 D0 88	YES

Замечание

Не забывайте, что для повышения вероятности получения корректного результата стохастические алгоритмы Монте–Карло запускаются несколько раз.

Задача P6. Симон расставляет ферзей

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Лиирон Литтнер, механик из команды Гуррен–дан, дал Симону очень непростую задачу — расставить N ферзей на большой шахматной доске $N \times N$ так, чтобы никакие два из них не били друг друга. Симон обратился к Вам, чтобы Вы помогли ему решить эту задачу.

Формат входных данных

В единственной строке находится число $5 \leqslant N \leqslant 200$ — размер шахматной доски.

Формат выходных данных

Выведите N чисел a_i — это номер горизонтали, на которую вы поставите ферзя, занимающего i–ую вертикаль. Нумерация горизонталей идёт снизу вверх, от 1 до N (как на обычной шахматной доске).

Система оценки

Подзадача	Баллы	Дополнительные ограничения	Необходимые подзадачи	Информация о проверке
0	–	тест из условия	–	полная
1	1	$n \leqslant 20$	0	первая ошибка
2	1.5	$n \leqslant 50$	0–1	первая ошибка
3	1	$n \leqslant 90$	0–2	первая ошибка
4	1.5	$n \leqslant 140$	0–3	первая ошибка
5	2	$n \leqslant 200$	0–4	первая ошибка

Примеры

стандартный ввод	стандартный вывод
5	5 2 4 1 3
8	3 6 4 1 8 5 7 2

Замечание

Возможно, Вам придётся перебрать несколько разных `seed` для генератора случайных чисел. Но не стоит делать на это ставку — количество попыток ограничено!

При этом, лучше не использовать время (и другие неконстантные `seed`) для инициализации генератора случайных чисел, чтобы результаты работы Вашей программы были воспроизводимы на сервере.