Computer Science

---

1. Что такое сортировка, зачем она нужна, какие виды сортировок бывают, какие у них свойства

Сортировка – это алгоритм упорядочивания элементов по какому-либо признаку. Сортировка необходима для обеспечения удобства поиска информации. Например, у нас есть список из 1000 людей разного возраста. В данный момент нам может быть необходимо найти 20 самых младших людей, можно проходить каждый раз по всему массиву данных о людях, находить самого младшего, записывать в отдельную структуру данных, потом проходить вновь, искать следующего младшего, проверяя, что такого уже не записано в нашу результирующую структуру данных. Это не очень удобно. В отсортированных данных по возрасту нам нужно было бы просто отобрать первые 20 (или последние – в случае убывающей сортировки) человек и все. К тому же это в данный момент нам нужно было отобрать 20 самых младших людей, а спустя пять минут будет нужно найти 4 старших человека или людей определенного возраста. Поиск информации в беспорядочно расположенных данных трудоемок. В отсортированных данных поиск информации происходит гораздо быстрее, тем более для отсортированных данных могут применяться специальные виды поиска (например, бинарный поиск), которые ищут информацию с еще большей производительностью.

Основные свойства сортировок:

- **время работы** – это свойство зачастую считается наиболее важным. Оценивается худшее, среднее и лучшее время работы алгоритма. У большинства алгоритмов временные оценки бывают O(n*log n), O(n^2);
- **дополнительная память** – свойство сортировки, показывающее, сколько дополнительной памяти требуется алгоритму. Сюда входят дополнительный массив, переменные, затраты на стек вызовов. Обычно доп. затраты памяти составляют O(1), O(log n), O(n);
- **устойчивость** – устойчивой называется сортировка, не меняющая порядок объектов с одинаковыми ключами. Ключ – поле элемента, по которому проводим сортировку;
- **количество обменов** – этот параметр может быть важен в том случае, если обмениваемые объекты имеют большой размер, т.к. при большом количестве обменов время работы сильно увеличивается;

Некоторые виды сортировок:

- **Пузырьковая сортировка**
  - время работы: лучшее – O(n), среднее – O(n^2), худшее – O(n^2);
  - затраты памяти – O(1);
  - устойчивая;
  - количество обменов – обычно O(n^2) обменов;
- **Сортировка выбором**
  - время работы: лучшее – O(n^2), среднее – O(n^2), худшее – O(n^2);
  - затраты памяти – O(1);
  - неустойчивая;
  - количество обменов – обычно O(n) обменов;
- **Сортировка вставками**
  - время работы: лучшее – O(n), среднее – O(n^2), худшее – O(n^2);
  - затраты памяти – O(1);
  - устойчивая;
  - количество обменов – обычно O(n^2) обменов;
- **Быстрая сортировка**
  - время работы: лучшее – O(n*log n), среднее – O(n*log n), худшее – O(n^2);
  - затраты памяти – O(log n);
  - неустойчивая;
  - количество обменов – обычно O(n*log n) обменов;
- **Сортировка слиянием**
  - время работы: лучшее – O(n*log n), среднее – O(n*log n), худшее – O(n*log n);
  - затраты памяти – O(n);

- устойчивая;
- количество обменов – обычно O(n*log n) обменов;

- **Поразрядная сортировка**
  - время работы: лучшее – O(nk), среднее – O(nk), худшее – O(nk);
  - затраты памяти – O(n + k);
  - устойчивая;
  - количество обменов – обычно O(nk) обменов;

---

2. Бинарный поиск, принцип и краткое описание алгоритма

Ответ дан с использованием термина «множество» для упрощения пояснения, поиск же может применяться и к другим структурам данных.

Бинарный поиск – алгоритм поиска объекта по заданному признаку в множестве объектов, упорядоченных по тому же признаку.

Принцип. На каждом шаге множество объектов делится на 2 части и в работе остается та часть, где находится искомый объект, деление множества на 2 части продолжается до тех пор, пока элемент не будет найден, либо пока не будет установлено, что его нет в заданном множестве.

Алгоритм бинарного поиска (на отсортированной по возрастанию структуре данных):

1) задаем границы: левая граница = 0вой индекс; правая граница = максимальный индекс структуры данных;
2) берем элемент посередине между границами, сравниваем его с искомым;
3) оцениваем результат сравнения:
   - если искомое равно элементу сравнения, возвращаем индекс элемента, на этом работа алгоритма заканчивается;
   - если искомое больше элемента сравнения, то сужаем область поиска таким образом: левая граница = индекс элемента сравнения + 1;
   - если искомое меньше элемента сравнения, то сужаем область поиска таким образом: правая граница = индекс элемента сравнения - 1;
4) Повторяем шаги 1-3 до тех пор, пока правая граница не станет меньше левой;
5) Возвращаем -1 (до этого шага доходим только в случае, если элемент не был найден);

Время выполнения алгоритма – O(log n);

---

3. Quick Sort, принцип и краткое описание алгоритма

Ответ дан с использованием термина «массив» для упрощения пояснения, сортировка же может применяться и к другим структурам данных.

Принцип

Опирается на принцип «разделяй и властвуй». Выбирается опорный элемент, это может быть любой элемент. От выбора элемента не зависит корректность работы алгоритма, но в отдельных случаях выбор этого элемента может повысить эффективность работы алгоритма. Оставшиеся элементы сравниваются с опорным и переставляются так, чтобы массив представлял собой последовательность: элементы меньше опорного-равные опорному элементы-элементы больше опорного. Для частей «больше» и «меньше» опорного элемента рекурсивно выполняется та же последовательность операций, если размер этой части составляет больше 1 элемента.

На практике входные данные обычно делят не на 3, а на 2 части. Например, «меньше опорного элемента» и «больше или равны опорному элементу». В общем случае разделение на 2 части эффективнее.

Алгоритм

1) проверяется, что во входном массиве больше 1 элемента, в противном случае алгоритм завершает свое действие;
2) с помощью какой-то схемы разбиения (например, схема разбиения Хоара) элементы в массиве меняются местами и определяется опорный элемент;
3) массив разбивается на 2: «меньше опорного элемента» и «больше или равны опорному элементу»;

4) рекурсивно вызывается этот же алгоритм для частей массива, полученных в пункте 3;

Разбиение Хоара

Эта схема разбиения подразумевает использование 2 индексов (с начала массива и с конца массива), которые приближаются навстречу друг к другу, пока найдется пара элементов, где один элемент больше опорного и расположен перед ним, а другой меньше опорного и расположен после него. Эти элементы меняются местами. Поиск таких пар элементов и их обмен происходит до тех пор, пока индексы не пересекутся. Алгоритм возвращает последний индекс. Эта схема разбиения является одной из наиболее эффективных.

Свойства

- время работы: лучшее – O(n*log n), среднее – O(n*log n), худшее – O(n^2);
- затраты памяти – O(log n);
- неустойчивая;
- количество обменов – обычно O(n*log n) обменов;

Плюсы

- в среднем очень быстрая;
- требует небольшое количество дополнительной памяти;

Минусы

- зависит от данных, на плохих данных деградирует до O(n^2);
- может привести к ошибке переполнения стека;
- неустойчива;

---

4. Merge Sort, принцип и краткое описание алгоритма

Ответ дан с использованием термина «массив» для упрощения пояснения, сортировка же может применяться и к другим структурам данных.

Принцип

Основная идея заключается в том, чтобы разбить массив на максимально малые части, которые могут считаться отсортированными (массивы с одним элементом), а потом сливать их между собой в порядке, необходимом для сортировки.

Алгоритм

1) Если в массиве меньше 2 элементов, то он уже отсортирован, алгоритм завершает свою работу;
2) Массив разбивается на 2 части (примерно пополам), для которых рекурсивно вызывается тот же алгоритм сортировки;
3) После сортировки двух частей массива производится слияние двух упорядоченных массивов в один упорядоченный массив;

Процедура слияния:

1) объявляются счетчики индексов для результирующего массива и для двух сливаемых частей, счетчики инициализируются 0;
2) в цикле с условием (пока не дошли до конца какого-либо из сливаемых массивов) на каждом шаге берется меньший из двух первых (по индексу счетчиков) элементов сливаемых массивов и записывается в результирующий массив. Счетчики индексов результирующего массива и массива, из которого был взят элемент, увеличиваются на 1.
3) все оставшиеся элементы сливаемого массива, счетчик которого не дошел до конца массива, записываются в результирующий массив.

Свойства

- время работы: лучшее – O(n*log n), среднее – O(n*log n), худшее – O(n*log n);
- затраты памяти – O(n);
- устойчивая;
- количество обменов – обычно O(n*log n) обменов;

Плюсы

- проста для понимания;
- независимо от входных данных стабильное время работы;
- устойчивая;

Минусы

- нужно O(n) дополнительной памяти;
- в среднем на практике несколько уступает в скорости Quick Sort;

---

5. Insertion Sort, принцип и краткое описание алгоритма

Ответ дан с использованием термина «массив» для упрощения пояснения, сортировка же может применяться и к другим структурам данных.

Принцип

Есть часть массива (в роли этой части может выступать первый элемент в массиве), которая уже отсортирована, требуется вставить остальные элементы в отсортированную часть, сохранив упорядоченность.

Алгоритм

1) выбирается один из элементов входных данных и вставляется на нужную позицию в уже отсортированной части массива;
2) шаг 1 повторяется до тех пор, пока весь массив не будет отсортирован;

Обычно в качестве отсортированной части массива изначально выступает первый элемент (элемент с индексом 0), порядок выбора очередного элемента для вставки в отсортированную часть произволен, но обычно с целью достижения устойчивости алгоритма элементы вставляются по порядку их появления в исходном массиве.

Свойства

- время работы: лучшее – O(n), среднее – O(n^2), худшее – O(n^2);
- затраты памяти – O(1);
- устойчивая;
- количество обменов – обычно O(n^2) обменов;

Плюсы

- проста в реализации;
- не нужна дополнительная память;
- ускоряется на частично отсортированных массивах;
- хороша на малом количестве элементов (<100);
- устойчивая;

Минусы

- медленная в среднем и худшем случае;

---

6. Selection Sort, принцип и краткое описание алгоритма

Ответ дан с использованием термина «массив» для упрощения пояснения, сортировка же может применяться и к другим структурам данных.

Принцип

На каждом i-ом шаге алгоритма находится минимальный элемент в неотсортированной части массива и меняется местами с i-ым элементом массива.

Алгоритм

на каждом i-ом шаге алгоритма (всего n шагов алгоритма):

1) находим индекс минимального элемента среди всех неотсортированных элементов;
2) меняем местами элемент с найденным индексом и i-ый элемент;

Свойства

- время работы: лучшее – O(n^2), среднее – O(n^2), худшее – O(n^2);
- затраты памяти – O(1);
- неустойчивая;
- количество обменов – обычно O(n) обменов;

Плюсы

- проста в реализации;
- не нужна дополнительная память;
- не больше O(n) обменов;

Минусы

- не ускоряется на частично отсортированных массивах;
- неустойчивая;
- медленная;

---

7. Radix Sort, принцип и краткое описание алгоритма

Ответ дан с использованием термина «массив» для упрощения пояснения, сортировка же может применяться и к другим структурам данных.

Поразрядная сортировка может быть двух типов LSD (least significant digit) и MSD (most significant digit): отличие в том, с какого разряда начинается анализ, т.е. с меньшего или большего соответственно. В ответе рассматриваю LSD версию поразрядной сортировки.

Принцип

Алгоритм представляет собой цикл по номеру разряда от младшего к старшему. На каждой итерации в результирующем массиве числа упорядочиваются по разряду с помощью какой-либо устойчивой сортировки. Чаще всего в качестве вспомогательной сортировки используется сортировка подсчетом (Counting Sort).

Алгоритм

1) Находится максимальный элемент среди элементов массива для определения количества разрядов;
2) В цикле по номеру разряда сортируется результирующий массив по разряду с помощью сортировки подсчетом;

Сортировка подсчетом (Counting Sort)

Обозначения: входной массив **ar**, вспомогательный массив **count** для счетчика, массив **res** для результата.

1) заполняем массив count 0ми;
2) для каждого ar[i] увеличиваем count[a[i]] на 1;
3) подсчитываем количество элементов меньше или равных j-ому индексу для каждого элемента. Для этого каждый count[j], начиная с j = 1, увеличиваем на count[j - 1]. Таким образом, в последней ячейке будет находиться количество элементов равное количеству элементов в массиве.
4) В цикле (по всем элементам входного массива) входной массив читается с конца, значение count[ar[i]] уменьшается на 1 и в res[count[ar[i]]] записывается a[i]. Проход по массиву с конца обеспечивает устойчивость сортировки.

Свойства

- время работы: лучшее – O(nk), среднее – O(nk), худшее – O(nk);
- затраты памяти – O(n + k);
- устойчивая;
- количество обменов – обычно O(nk) обменов;

Плюсы

- линейная сложность алгоритма;
- независимо от входных данных стабильное время работы;
- наиболее быстрая сортировка для массива, отсортированного в обратную сторону;
- хороша для сортировки строк;

Минусы

- нужно O(n + k) дополнительной памяти;
- по умолчанию не может обрабатывать отрицательные числа;
- по умолчанию не может обрабатывать числа с плавающей точкой;