Лекция 11.
Сортировки

Временная сложность алгоритмов

- Для решения одной и той же задачи можно использовать разные алгоритмы
- Разные алгоритмы отличаются друг от друга по производительности и объему требуемой памяти
- Важной характеристикой алгоритма является **временная сложность**

Временная сложность алгоритмов

- **Временная сложность** – это количество **элементарных операций**, требуемых в ходе выполнения алгоритма
- Элементарными операциями считаются сравнения, присваивания и арифметические операции

Пример – линейный поиск

- Рассмотрим обычный линейный поиск в массиве – мы проходимся по всем элементам от нулевого до последнего и сравниваем с искомым значением
- Пусть длина массива N
- Тогда в худшем случае нам понадобится N сравнений, т.е. временная сложность линейного поиска равна N

Временная сложность алгоритма

- Временная сложность:
  - Вычисляется через длину входных данных. Например, через длину массива или строки - N
  - Ее смотрят только с точностью до порядка
  - Смотрят только для худшего случая

Обозначение временной сложности

- Обычно временную сложность смотрят только до порядка
- Пусть, например, она равна $3N^2 + N$
- Тогда откидывают все коэффициенты и оставляют только главный член, который вносит самый большой вклад на бесконечности
- В данном случае получится N^2
- Записывается это так: O(N^2)

Худший случай

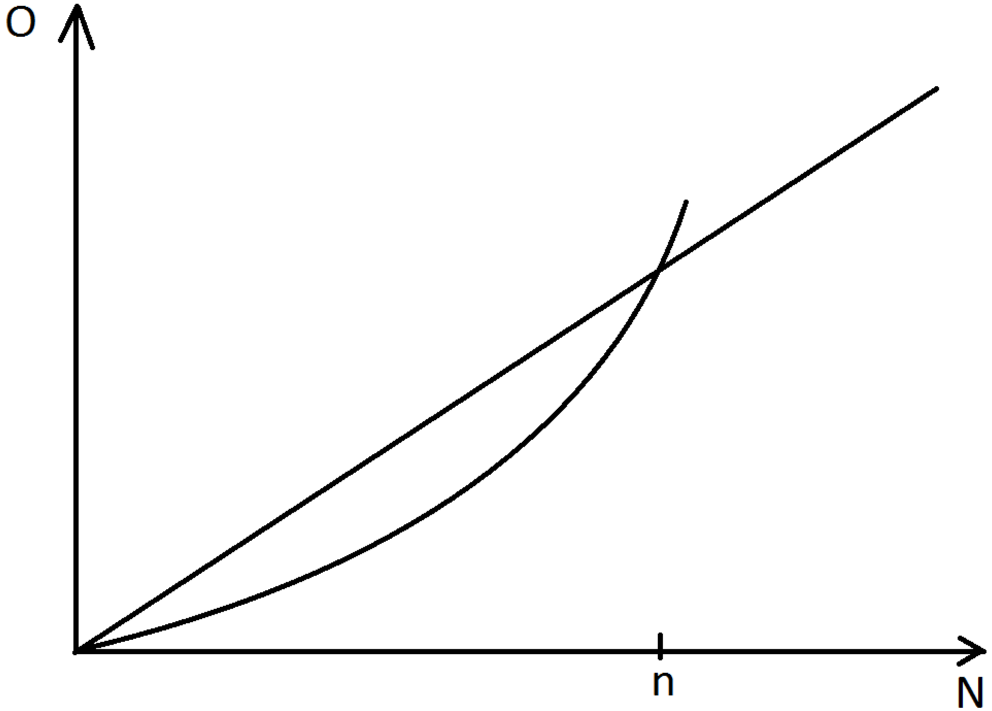
- Временную сложность оценивают для худшего случая – когда алгоритму требуется больше всего операций
- Например, в линейном поиске нужный элемент может оказаться первым, и тогда понадобится всего 1 итерация
- Но в худшем случае придется просмотреть весь массив, поэтому сложность будет O(N)

Пример – бинарный поиск

- Сложность бинарного поиска составляет $O(\log_2 N)$
- Логарифм растет гораздо медленнее линейной функции N
- Например, $\log_2 1024 = 10$
- Получается, бинарный поиск намного эффективнее линейного поиска
- Поэтому важно стараться применять алгоритмы, имеющую меньшую временную сложность

Поведение на малых данных

- Общее правило – выбирать алгоритм с меньшей сложностью, он лучше работает при больших N
- Но на малых данных алгоритм с большей сложностью может быть эффективнее



- На данных размера меньше n этот алгоритм с O(N^2) быстрее, чем с O(N)

Часто встречающиеся сложности

- O(1) – **константная**. Например, доступ по индексу массива, не зависит от длины массива
- $O(\log_2 N)$ – **логарифмическая**. Например, бинарный поиск
- O(N) – **линейная**. Цикл по массиву. Например, линейный поиск или поиск максимума
- $O(N \ * \ \log_2 N)$. Например, пирамидальная сортировка
- O(N^2) – **квадратичная**. 2 вложенных цикла по массиву. Например, сортировки
- В целом есть **степенные сложности** – **кубическая** и т.д.

Откуда берется сложность?

- Обычно это цикл по массиву или строке – это уже линейная сложность
- Если 2 вложенных цикла, то квадратичная
- Есть 3 – то кубическая и т.д.

- Поэтому важно не делать одни и те же операции на каждой итерации цикла, если их результат один и тот же

Сортировка

- **Сортировка** – это упорядочивание элементов массива в определенном порядке (например, в порядке неубывания)
- Будем рассматривать на примере массивов целых чисел, но алгоритмы верны для массивов любых типов

Свойства и классификация сортировок

- **Устойчивая сортировка** – если не меняет взаимного положения равных элементов
- **Естественная сортировка** – если алгоритм хорошо работает на частично или полностью упорядоченных данных
- **Внутренняя сортировка** – работает с массивом, целиком помещающимся в памяти и быстрым доступом по любому индексу. Обычно упорядочивается сам массив на месте, без дополнительной памяти
- **Внешняя сортировка** – работает с большими структурами данных с последовательным доступом, не помещающимися в оперативную память. Например, это сортировка файлов или списков

Простые сортировки

- Алгоритмов сортировки существует просто огромное количество
- **Простыми сортировками** называют несложные алгоритмы сортировки, которые имеют временную сложность $O(N^2)$
- В дальнейшем считаем что хотим упорядочить массив по неубыванию
- Простые сортировки не требуют дополнительной памяти, а переупорядочивают исходный массив

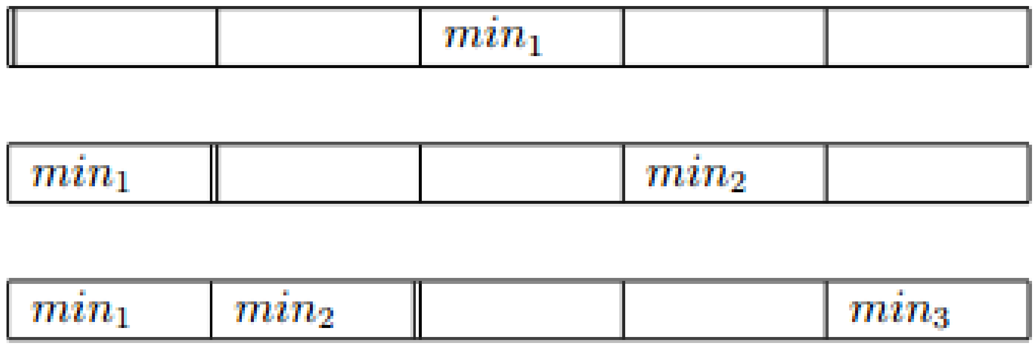
Обмен двух переменных

- Чтобы обменять значения двух переменных, нужно ввести третью вспомогательную переменную
- Например, надо обменять переменные x и y
- Вводим дополнительную переменную temp
- ```
  int temp = x;
  x = y;
  y = temp;
  ```
- Аналогично можно обменивать элементы массива, вместо x и y будут некоторые a[i] и a[j]
  ```

Сортировка выбором

- Шаг 1: линейно ищем минимальный элемент в массиве и обмениваем его с первым элементом
- Шаг 2: линейно ищем минимальный элемент в массиве, начиная со второго элемента, обмениваем его со вторым элементом
- …
- Шаг N-1: ставим минимальный элемент из последних двух на N-1 место

Сортировка выбором



- То есть многократно делаем следующие операции:
  - Ищем индекс минимального элемента в неотсортированной части массива
  - Обмениваем этот элемент с первым элементом неотсортированной части массива

Сортировка выбором

- Временная сложность:
  (N - 1) + (N - 2) + … + 1 $\sim$ $O(N^2)$

Задача

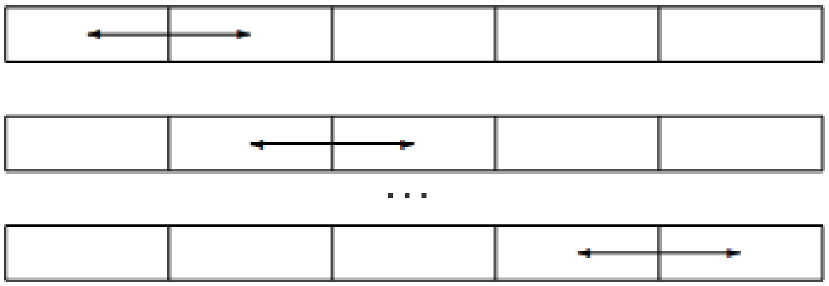
- Реализовать функцию поиска минимума в массиве
- Переделать на функцию, которая ищет индекс, по которому лежит минимум в массиве
- Переделать, чтобы функция поиска индекса минимума работала не по всему массиву, а только в части массива, начинающейся с индекса start

Задача на курс «Сортировка выбором»

- Реализовать сортировку выбором

Сортировка пузырьком

- Выполняем проход по массиву слева направо, сравнивая и при необходимости меняя местами соседние элементы



- После этого максимальный элемент окажется последним
- Повторяем процесс N-1 раз (или меньше, если за некоторую итерацию не произошло ни одного обмена)

Сортировка пузырьком

- В сортировке пузырьком отсортированная часть формируется справа
- Для сортировки пузырьком есть оптимизация – если за полный проход по массиву не было ни одного обмена, то массив уже отсортирован, и алгоритм нужно завершить

Задача на курс «Сортировка пузырьком»

- Реализовать сортировку пузырьком

Сортировка вставками

- Так же выполняем N – 1 итераций
- В левой части массива будем выстраивать отсортированную последовательность, на каждой итерации туда будет добавляться один элемент
- Перед первой итерацией считаем, что отсортированная последовательность состоит из первого элемента
- Далее, выполняем для каждого элемента от 2 до N - 1

5	2	1	3	6

Идея итерации алгоритма

- На итерации уже есть какая-то отсортированная часть массива
- И есть первый элемент неотсортированной части
- Надо найти индекс, куда надо вставить этот элемент
- А всё, что правее в отсортированной части – сдвинуть на 1 индекс вправо

3	5	6	4	1

3	4	5	6	1

- Тут 4 надо вставить по индексу 1, а числа 5 и 6 надо сдвинуть вправо на 1 индекс

Итерация сортировки вставками

- Пусть **i** – индекс первого элемента неотсортированной части
- Запоминаем в переменную **temp** элемент **array[i]**
- Идем справа налево по отсортированной части при помощи счетчика **j**, сначала он равен **i – 1**
  - Если **j < 0** или **temp >= array[j]**, то заканчиваем идти
    - Вставляем **temp** по индексу **j + 1**
    - На этом итерация завершена
  - Иначе сдвигаем **array[j]** вправо:
    - array[j + 1] = array[j]

Задача на курс «Сортировка вставками»

- Реализовать сортировку вставками

Быстрая сортировка

- **Быстрая сортировка** – это уже более сложный алгоритм
- Его временная сложность в худшем случае $\boldsymbol{O(N^2)}$, но в среднем $\boldsymbol{O(\mathrm{N} * \log_2 N)}$, что является лучше, чем простые сортировки

Быстрая сортировка

- Быстрая сортировка реализуется с помощью рекурсивной функции:
- static void QuickSort(int[] a, int left, int right)
- **left** и **right** обозначим индексы границ массива **a**

Быстрая сортировка

- Выберем некоторое произвольное число **x** в диапазоне от минимума до максимума по массиву, например, первый (или средний) элемент
- Хотим сделать следующее: чтобы все элементы до некоторого индекса были меньше, либо равны x, а остальные – больше, либо равны x

4	3	5	6	10	7	8	11

$x = 5$

- После этого рекурсивно вызываем этот же алгоритм для левой части массива и для правой. Но это если эта часть массива содержит как минимум два элемента

Быстрая сортировка

- Как нужным образом поделить массив на две части?

1. Запускаем два счетчика: **i** слева направо от **left** до **right**;
j – справа налево от **right** до **left**
2. Пока **i <= j**:
   - Сначала двигаем **i**, пока не встретим элемент, который **>= x**. После этого начинаем двигать **j**, пока не встретим элемент, который **<= x**
   - Если **i <= j**, то делаем обмен элементов по этим индексам, затем сдвигаем оба счетчика еще на один элемент
и на шаг 2. Иначе – завершаем процесс и на шаг 3
3. В этот момент все элементы, которые **<= x**, находятся левее **i**, а которые **>= x** – правее **j**

   Если **i** < **right**, то вызываем рекурсивно для части от **i** до **right**. Если **j** > **left**, то и для части от **left** до **j**

Быстрая сортировка

- Для остановки рекурсии надо рассмотреть два выделенных случая:
  - Передали массив длины 1 – можно считать, что он уже отсортирован, ничего делать не нужно
  - Передали массив длины 2 – если нужно, меняем эти два элемента местами

Опорный элемент

- Число x называют **опорным элементом**
- Выбирать его можно любым образом из диапазона **[min, max]**, где **min** и **max** – минимум и максимум из значений в массиве
- В идеале, опорный элемент должен делить массив на две равные части, тогда скорость работы алгоритма максимальна
- Но чтобы выбрать элемент таким образом, нужно тоже затратить время, что в итоге не окупается, поэтому в качестве опорного элемента часто берут первый или среднее арифметическое первого и последнего элементов

Задача на курс «Быстрая сортировка»

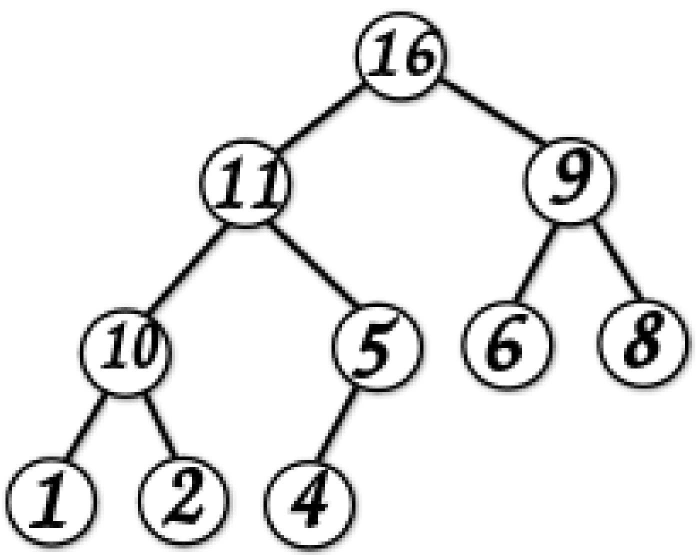
- Реализовать быструю сортировку

Пирамидальная сортировка

- **Пирамидальная сортировка** – тоже сложный алгоритм сортировки
- Его временная сложность даже в худшем случае $\boldsymbol{O(\mathrm{N} * \log_2 N)}$
- Но зато если массив уже почти отсортирован, то алгоритм все равно будет работать долго
- У алгоритма сложная и интересная идея

Пирамида (куча)

- **Пирамида (куча)** – это двоичное дерево, у которого каждый родитель больше либо равен своих детей
- Дерево двоичное, т.к. у каждого узла не более 2 детей
- Число 16 здесь – это корень дерева. Дети этого узла – это 11 и 9 и т.д.

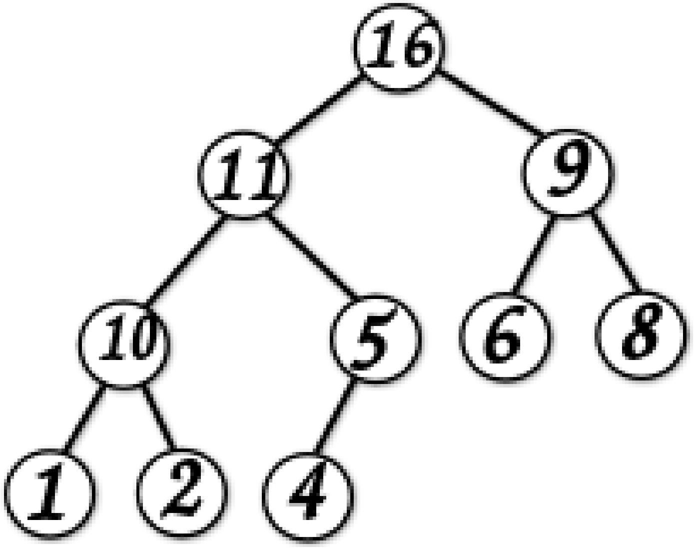


Пирамида (куча)

- Хоть куча и иерархична, ее можно хранить в плоском виде – в массиве
- Правило такое – если узел лежит по индексу i, то его дети лежат по индексам: 2i + 1 и 2i + 2

16	**11**	**9**	**10**	**5**	**6**	**8**	**1**	**2**	**4**

- Поэтому правило кучи такое:
- a[i] >= a[2i + 1]
  a[i] >= a[2i + 2]



Пирамидальная сортировка – 1 этап

- 1 этап алгоритма – привести массив к виду кучи
- Это делается следующим образом:
  - Пусть длина массива равна N
  - Тогда если взять элементы с индексами >= N / 2, то у них нет детей
  - Поэтому эта часть массива уже не противоречит свойству кучи
  - А дальше мы начинаем идти от индекса N / 2 - 1 справа налево, выполняя для каждого элемента так называемое **просеивание**

Просеивание

- **Просеивание** – это процесс, при котором мы сравниваем элемент с его максимальным ребенком и, если надо, делаем обмен
- Если максимальный ребенок не больше родителя (или детей нет), то просеивание завершается
- Иначе – нужно обменять элемент со своим максимальным ребенком и продолжать просеивание дальше, с этой новой позиции
- Т.к. после этого обмена на новом месте тоже может нарушаться свойство кучи

Пирамидальная сортировка

- Например, у нас такой массив:
  - 10 2 3 6 8 7 1 12
- Длина массива N = 8, индекс N / 2 – 1 равен 3
- Получается, часть, начиная с индекса 4 не нарушает свойство кучи
  - 10 2 3 **6** | 8 7 1 **12**
- Дальше пытаемся встроить в кучу число 6. Для этого сравниваем 6 с его детьми – элементами по индексам 7 и 8.
- Элемента с индексом 8 вообще нет, а элемент с индексом 7 больше, чем 6. Поэтому обмениваем их местами
  - 10 2 3 | 12 8 7 1 **6**
- У 6 после обмена нет детей в куче, поэтому с ним закончили

Пирамидальная сортировка

- 10 2 **3** | 12 8 **7** **1** 6
- Дальше пытаемся встроить 3, сравниваем его с детьми – это элементы с индексами 5 и 6
- Число 7 больше, поэтому обмениваем 3 с ним. Дальше у 3 детей уже нет, поэтому 3 встроена в кучу
  - 10 2 | 7 12 8 **3** 1 6

Пирамидальная сортировка

  - 10 **2** | 7 **12** **8** 3 1 6
- Дальше пытаемся встроить 2, сравниваем его с детьми – это элементы с индексами 3 и 4
- Обмениваем 2 с максимальным ребенком – числом 12
  - 10 | 12 7 **2** 8 3 1 **6**
- Но после обмена у 2 также есть дети в куче. И может получиться, что они больше. Поэтому процесс надо продолжить, пока не дойдем до узла без детей
- У 2 ребенком будет число 6, оно больше, поэтому обмениваем. Вот теперь 2 встроена в кучу
- 10 | 12 7 6 8 3 1 **2**

Пирамидальная сортировка

- **10** | **12** **7** 6 8 3 1 2
- Сравниваем 10 с детьми – 12 и 7, обмениваем с максимальным ребенком, большим 10
  - 12 **10** 7 **6** **8** 3 1 2
- Дальше смотрим детей 10, вдруг надо обменять с ними
- Но все дети меньше, чем 10, поэтому на этом первый этап алгоритма завершен

Пирамидальная сортировка – 2 этап

- Когда мы привели массив к виду кучи, то максимальный элемент будет по индексу 0
  - 12 10 7 6 8 3 1 2
- Дальше начинаем 2 этап алгоритма:
- Обмениваем нулевой элемент с последним элементом:
  - 2 10 7 6 8 3 1 | 12
- Число 12 – это будет отсортированная часть массива
- После обмена нулевой элемент может нарушать свойство кучи, поэтому эту часть массива надо опять привести к виду кучи – нужно сделать **просеивание** нулевого элемента
- Это будет намного быстрее, чем 1 этап алгоритма

Пирамидальная сортировка

- **2 10 7** 6 8 3 1 | 12
- Выполняем просеивание нулевого элемента
- Сравниваем 2 с его детьми, и обмениваем с максимальным ребенком, большим 2. Это число 10
- 10 **2** 7 **6 8** 3 1 | 12
- Дальше сравниваем 2 с его новыми детьми – 6 и 8, обмениваем с 8
- 10 8 7 6 **2** 3 1 | 12
- Дальше у 2 больше нет детей в неотсортированной части массива, поэтому закончили

Пирамидальная сортировка

- 10 8 7 6 **2** 3 1 | 12
- Неотсортированная часть массива пришла к виду кучи
- Обмениваем нулевой элемент с последним элементом неотсортированной части
  - 1 8 7 6 2 3 | 10 12
- Дальше просеиваем 1 и т.д.
- Так мы отсортируем весь массив

Задача на курс «Пирамидальная сорт-ка»

- Реализовать пирамидальную сортировку

Вопросы

- Какие свойства у рассмотренных алгоритмов сортировок – устойчивость и естественность?
- И это внутренняя сортировка или внешняя?