Algoritms

Общие положения

* Алгоритмы могут быть inplace (меняют сам объект) и outplace (создают новый)
* I++ и i—выполняются с одинаковой скоростью, но сравнение с произвольным числом data.length медленнее чем сравнение с 1

Бинарный поиск

* Сколько будет операций поиска: например массив у нас 500\_000\_000 записей, log2(a\*b) = log2(a) + log2(b) => log2(500) + log2(1000) + log2(1000) = 9 + 10 + 10 = 29 раз

Инвертирование массивов

* Идем за один цикл сначала до середины (data.length/2) и меняем местами элементы по очереди

Инвертировать строку

* Делаем char[] d = “..”.toCharArray(); инвертируем как массив и возвращаем new String(d);

Сортировка пузырьком

* Двойной цикл, направо выгоняем наибольший элемент, начинаем сначала каждый раз
* Алгоритм:  
  for (int i = len - 1; i > 1; i--)

for (int j = 0; j < i; j++)

if (ar.get(j) > ar.get(j + 1))

swap(ar, j, j+ 1);

* Время выполнения за O(N2)

Сортировка выборкой

* Идем с начала, ищем min затем меняем с первым элементом, далее ищем min со второго
* Алгоритм:  
  for (int out = 0; out < len - 1; out++){

min = out;

for (int in = out + 1; in < len; in++){

if (ar.get(in)< ar.get(min))

min = in;

}

swap(ar, min, out);

}

* Время выполнения за O(N2)
* Но перестановок меньше, соответственно исполнение в целом быстрее пузырька

Сортировка вставкой

* Наиболее быстрый из итеративных алгоритмов
* Берем элемент, ищем позицию слева куда вставить его, вставляем, а массив сдвигаем вправо
* Алгоритм:  
  for (int out = 0; out < len; out++){

int tmp = ar.get(out);

int in = out;

while (in > 0 && ar.get(in -1) >= tmp){

ar.set(in, ar.get(in - 1));

in--;

}

ar.set(in, tmp);

}

* Время выполнения за O(N2)
* На обратноотсортированом работает не быстрее пузырька
* Можно ускорить бинарным поиском и копировать массив как System.arraycopy()
* Устойчивость сортировки в том что если объекты отсортированы по какому-либо признаку, а потом по другому, то первоначальный порядок тоже будет сохранен (т.е. будет сохранен вторичный порядок)

Рекурсия

* Главная задача разбить большую задачу на подзадачи

Сортировка слиянием

* Выполняется за время O(N\*logN)
* Требует выделение в памяти еще одного массива
* По сути это слияние двух отсортированных массивов
* Реализуется рекурсивно, делится пополам пока не дойдет до одного элемента, далее происходит слияние двух отсортированных половин
* Алгоритм:  
  if (lbound == rbound)

return;

else {

int m = (rbound + lbound )/2;

mergeSorting(ar, lbound, m);

mergeSorting(ar, m+1, rbound);

merge(ar, lbound,rbound);

}

Слияние сортированных массивов

Умножение матриц

Структуры данных

* Это способ организации данных в памяти компьютера или на диске, алгоритмы обеспечивают выполнение различных операций с ними
* Типы структур данных:   
  - массивы (плюс: быстрая вставка, быстрый доступ, минус: медленный поиск, медленное удаление, фикс размер)   
  - упорядоченный массив (плюс: поиск выполняется быстрее, минус: медленная вставка и удаление, фикс размер)  
  - стек (плюс: доступ LIFO, минус: медленный доступ к другим элементам)  
  - очередь (плюс: доступ FIFO, минус: медленный доступ к другим элементам)  
  - связанный список (плюс: быстрая вставка, быстрое удаление, минус: медленный поиск)  
  - двоичное дерево (плюс: быстрый поиск, вставка, удаление (если дерево сбалансировано), минус: сложный алгоритм удаления)  
  - красно-черное дерево (плюс: быстрый поиск, вставка, удаление, всегда сбалансировано минус: сложность)  
  - дерево 2-3-4 (плюс: быстрый поиск, вставка, удаление, всегда сбалансировано, подходит для хранения на диске, минус: сложность)  
  - хеш-таблица (плюс: быстрый доступ, минус: медленное удаление, неэффективное использование памяти)  
  - куча (плюс: быстрая вставка, удаление, доступ к наибольшему элементу, минус: медленный доступ к другим элементам)  
  - граф (плюс: моделирование реальных ситуаций, минус: некоторые алгоритмы медленные и сложные)
* Дерево это тот же граф, но без цикличных записей, корень может быть любой веткой
* Как правило в java запись представляет из себя объект, а поля объекта – поля записи

Массивы

* Вставка в конец происходит быстро, т.к. всегда известно количество элементов
* Поиск медленный и в среднем проходит за время N/2
* При удалении все последующие значения смещаются к началу (сначала нужно элемент найти это N/2, затем удалить это N/2,в сумме в среднем получается N шагов)
* Это все если нет в массиве дубликатов
* Главное преимущество упорядоченных массивов – многократное ускорение поиска
* Оценивают эффективность структуры данных с помощью О-синтаксиса (линейный поиск пропорционален O(N) шагов, а двоичный O(log N), вставка в неупорядоченный массив занимает О(1) – т.е. время постоянно)

Стеки и очереди

* Более абстрактные структуры данных
* В стеке доступен только последний элемент
* Работает по принципу LIFO
* Занесение и извлечение элементов происходит за время O(1)
* Можно использовать решение на основе массива или связанных списков
* Очередь работает по принципу FIFO
* Как только очередь достигнет конца – маркер переносится в начало, если там были удаленные, это называется циклической очередью
* Дек (dequeuer) двунаправленная очередь – вставка удаление могут быть как с конца, так и с начала

Приоритетные очереди

* Обычная очередь, но данные хранятся в упорядоченном виде
* Может быть реализована на основе массива или связанного списка
* Вставка происходит за время O(N), извлечение O(1)

Связные списки

* Каждый элемент списка встраивается в объект называемый элементом списка
* В каждом из объектов хранится ссылка на следующий
* Доступ к произвольному элементу возможен только если пройти по цепочке от начала до элемента
* Вставка и удаление вначале за O(1), поиск, удаление и вставка в середину на O(N), по сути как в массиве, но работает быстрее т.к. не требует перемещения элементов

Вопросы на собеседовании

* Алгоритмы поиска (5 штук)
* Хеш структуры
* Деревья (B-tree (индексы в БД) и красно-черные RB (используется в tree map))

Литература

* Макконел Совершенный код
* Мартин Чистый код
* Бек Шаблоны реализации корпоративных приложений
* Фаулер Рефакторинг