М8О-101Б-21 Постнов Александр Вячеславович;

Вариант 17;

Зачет № 3 Сортировки;

1. (17 + 5) % 6 + 1 = 5

5) Индексная сортировка(выборка с подсчетом)

Индексная сортировка:

Данный метод впервые упоминается в работе Э.Х. Фрэнда, хотя он и не заявил о ней как о

своей собственной разработке.

При упорядочении таблицы этим методом необходима память для хранения исходной таблицы,

а также дополнительно должна быть выделена память под счетчик для каждого элемента

таблицы.

В этом алгоритме элементы не перемещаются. Он подобен сортировке таблицы адресов,

поскольку таблица счетчиков определяет конечное положение элементов.

Просмотр таблицы начинается с первой записи. Ее ключ сравнивается с ключами последующих

записей. При этом счетчик большего из сравниваемых ключей увеличивается на 1. При втором

просмотре таблицы первый ключ уже не рассматривается, второй ключ сравнивается со всеми

последующими. Результаты сравнений фиксируются в счетчиках. Для таблицы, содержащей N

элементов, этот процесс повторяется N-1 раз.

После выполнения всех просмотров счетчик каждого элемента указывает, какое количество

ключей таблицы меньше ключа этого элемента. Эти счетчики используются затем в качестве

индексов элементов результирующей таблицы. Поместив записи в результирующую таблицу в

соответствии со значениями их счетчиков, получим упорядоченную таблицу. Другими словами,

если известно, что некоторый ключ превышает ровно 27 других, то после сортировки

соответствующий элемент должен занять 28-е место.

*Асимптотика O(n\*\*2). Так как для каждого элемента мы проходимся еще раз по массиву.*

Код:

void sort\_table(int \*a, int n) { //индексная сортировка массива

int i, j, k;

int \*b = new int[n];

for (i = 0; i < n; i++) {

for (j = k = 0; j < n; j++)

if (a[j] < a[i] || a[j] == a[i] && i < j)

k++;

b[k] = a[i];

}

for (i = 0; i < n; i++)

a[i] = b[i];

}

*Заключение: эту сортировку можно использовать для небольших данных(до 1000 элементов, она там будет показывавать себя лучше эффективной ).*

—------------------------------------------------------------------------------------------------------------------------

1. (17+3) % 4 + 1 = 1
2. Пирамидальная сортировка (или сортировка кучей, Heap Sort)

Куча (heap) — это не что иное, как двоичное дерево с некоторыми дополнительными правилами, которым оно должно следовать: во-первых, оно всегда должно иметь структуру кучи, где все уровни двоичного дерева заполняются слева направо, и, во-вторых, оно должно быть упорядочено в виде max-кучи или min-кучи. В качестве примера я буду использовать min-кучу.

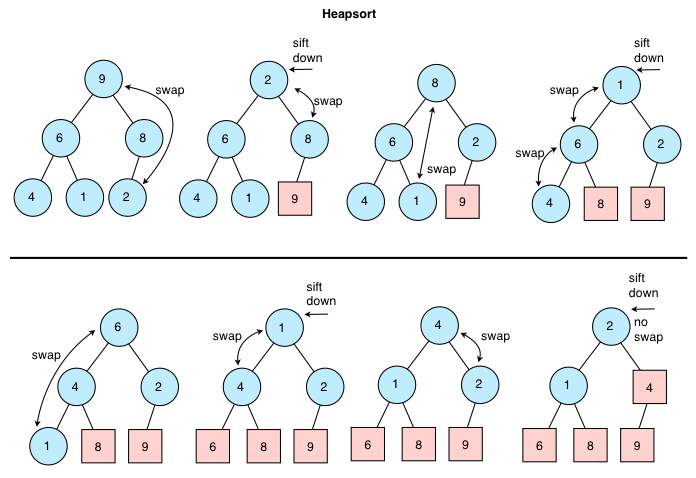
Алгоритм пирамидальной сортировки — это метод сортировки, который полагается на такие структуры данных как двоичные кучи. Поскольку мы знаем, что кучи всегда должны соответствовать определенным требованиям, мы можем использовать это для поиска элемента с наименьшим значением, последовательно сортируя элементы, выбирая корневой узел кучи и добавляя его в конец массива.

В самом начале мы имеем несортированный массив. Первый шаг — взять этот массив и превратить его в кучу; в нашем случае мы хотим превратить его в min-кучу. Итак, нам нужно преобразовать данные несортированного массива и построить из них min-кучу. Обычно это инкапсулируется одной функцией, которую можно назвать, например, buildHeap.

Как только данные нашего массива будут представлены в формате min-кучи, мы можем быть уверенными, что наименьшее значение находится в корневом узле нашей кучи. Помните, что даже если куча не будет полностью отсортирована, если мы построили нашу min-кучу правильно и без каких-либо ошибок, каждый отдельно взятый родительский узел в нашей куче будет меньше по значению, чем его дочерние элементы. Мы собираемся переместить наименьшее значение, расположенное в корневом узле, в конец кучи, заменив его последним элементом.

Теперь самый маленький элемент в куче находится в последнем узле, что нам и нужно. Мы знаем, что он находится в отсортирован относительно остальных элементов, поэтому его можно полностью удалить из кучи (функция extract-min). Но нам нужно сделать еще кое-что: убедиться, что новый элемент корневого узла находится на своем месте! Маловероятно, что элемент, который мы сделали корневым узлом, находится на своем месте, поэтому мы переместим элемент корневого узла вниз в его корректное положение, используя функцию, которая обычно называется heapify (приведение к виду кучи).

Вот и все! Алгоритм продолжает повторять эти шаги до тех пор, пока куча не сократится до одного единственного узла. На момент, когда это произойдет, мы будем уверенны, что все элементы в несортированном массиве находятся в своих отсортированных позициях и что последний оставшийся узел в конечном итоге станет первым элементом в отсортированном массиве. Общее время работы этого алгоритма составляет O(n log n).



Заключение: эту сортировка относится к категории эффективных сортировок. Ее следует использовать для больших наборов данных.