**Быстрая сортировка (Quicksort).**

Быстрая сортировка (англ. quick sort, сортировка Хоара) — один из самых известных и широко используемых алгоритмов сортировки. Среднее время работы , что является асимптотически оптимальным временем работы для алгоритма, основанного на сравнении. Хотя время работы алгоритма для массива из элементов в худшем случае может составить , на практике этот алгоритм является одним из самых быстрых.

Быстрый метод сортировки функционирует по принципу "разделяй и властвуй".

* Массив a[l…r]a[l…r] типа TT разбивается на два (возможно пустых) подмассива a[l…q]a[l…q] и a[q+1…r]a[q+1…r], таких, что каждый элемент a[l…q]a[l…q] меньше или равен a[q]a[q], который в свою очередь, не превышает любой элемент подмассива a[q+1…r]a[q+1…r]. Индекс вычисляется в ходе процедуры разбиения.
* Подмассивы a[l…q]a[l…q] и a[q+1…r]a[q+1…r] сортируются с помощью рекурсивного вызова процедуры быстрой сортировки.
* Поскольку подмассивы сортируются на месте, для их объединения не требуются никакие действия: весь массив a[l…r]a[l…r] оказывается отсортированным.

void quicksort(vector <int> &a, int l, int r) { // Быстрая сортировка

int x = a[l], ll = l, rr = r;

for (int i = l; i < r; i++) {

while ((a[r] >= x) && (l < r))

r--;

if (l != r) {

a[l] = a[r];

l++;

}

while ((a[l] <= x) && (l < r))

l++;

if (l != r) {

a[r] = a[l];

r--;

}

}

a[l] = x;

x = l;

l = ll;

r = rr;

if (l < x)

quicksort(a, l, x - 1);

if (r > x)

quicksort(a, x + 1, r);

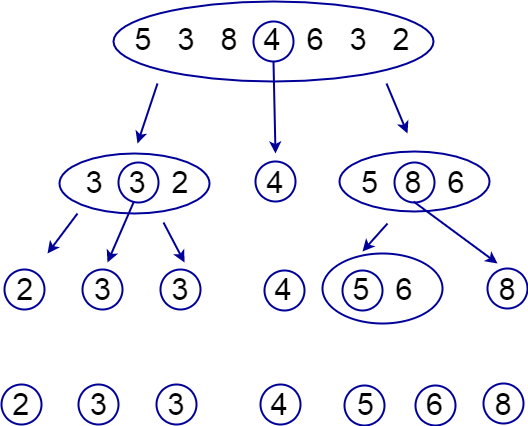
}

Алгоритм быстрой сортировки работает с заданным массивом чисел, который состоит из n элементов.

1.В наиболее сбалансированном варианте при каждой операции разделения массив делится на две одинаковые (Если размерность массива нечетная, то части могут отличаться на 1) части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1 составит {\displaystyle \log \_{2}n} log2(n). В результате количество сравнений, совершаемых быстрой сортировкой было бы равно значению рекурсивного выражения {\displaystyle C\_{n}=2\cdot C\_{n/2}+n} Cn=2\*Cn/2+ n, что даёт общую сложность алгоритма {\displaystyle O(n\cdot \log \_{2}n)} O(n\*log2(n)).

2. В самом несбалансированном варианте каждое разделение даёт два подмассива размерами 1 и n-1, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых. При простейшем выборе опорного элемента — первого или последнего в массиве, — такой эффект даст уже отсортированный (в прямом или обратном порядке) массив, для среднего или любого другого фиксированного элемента «массив худшего случая» также может быть специально подобран. В этом случае общее время работы составит О(n^2), то есть сортировка будет выполняться за квадратичное время.

3. Среднюю сложность при случайном распределении входных данных можно оценить лишь вероятностно. Допустим массив разделится как 75%|25%. Тогда в нашем случае, из-за такого разбиения будет сильно увеличиваться лишь константа. Общее время так и останется O(n\*log(n)).



**Пирамидальная сортировка (сортировка кучи, Heapsort).**

**Пирамидальная сортировка** (англ. *Heapsort*, «Сортировка кучей») — алгоритм сортировки, работающий в худшем, в среднем и в лучшем случае (то есть гарантированно) за *О(n* log *n)* операций при сортировке *n* элементов. Количество применяемой служебной памяти не зависит от размера массива (то есть, O(1)).

void Heap(vector<int>& a, int i, int n) { // Пирамидальная сортировка

int max = i;

for (int j = 1; j > 0; j++) {

int c = 2 \* i + 1;

if (c < n && a[c] > a[max])

max = c;

c++;

if (c < n && a[c]> a[max])

max = c;

if (max == i)

break;

else {

swap(a[max], a[i]);

i = max;

}

}

}

void HeapSort(vector<int>& a) {

for (int i = a.size() / 2; i >= 0; i--)

Heap(a, i, a.size());

for (int i = a.size() - 1; i >= 1; i--) {

swap(a[0], a[i]);

Heap(a, 0, i);

}

}

Алгоритм пирамидальной сортировки состоит из двух функций:

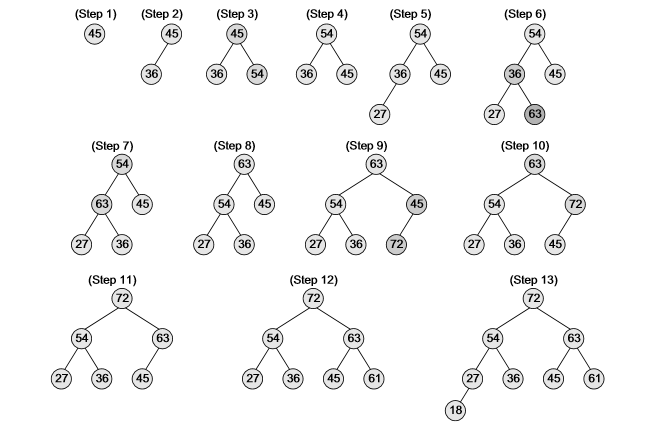
1. void HeapSort(vector<int>& a) {

– функция построения кучи

void Heap(vector<int>& a, int i, int n) { - Основная функция, выполняющая пирамидальную сортировку. Рассмотрим функцию «Куча», данная функция отвечает за построение дерева с log(n) уровнями и просеивание его по ходу выполнения сортировки, остальные операции занимают O(1) времени. В конечном итоге, общее время выполнения данной функции будет составлять O(log(n)).

В основной функции сортировки HeapSort используется функция Heap. В HeapSort используется два цикла, которые

1. Формирует дерево (выполняется за (n/2-1)\*log(n) времени)
2. Проходится по всем элементам и будет работать за O(n\*log(n)).



Общее время выполнения будет составлять O(n\*log(n)).