**Быстрая сортировка**

void Quicksort(int\* s\_arr, int one, int two)//быстрая сортировка

{

int i = one, j = two, x = s\_arr[(one + two) / 2];//первый и второй элемент

do

{

while (s\_arr[i] < x) i++;//пока массив от первого элемента меньше от (1+2элементов)\2 прибавляем

while (s\_arr[j] > x) j--;

if (i <= j)//пока первый меньше второго

{

if (s\_arr[i] > s\_arr[j]) swap(s\_arr[i], s\_arr[j]);//меняем местами наши элементы

i++;//первый прибавляем

j--;//второй убавляем

}

} while (i <= j);//идем пока первый меньше или равно второму

if (i < two)//если первый меньше второго

Quicksort(s\_arr, i, two);//сортируем

if (one < j)//и наоборот с другим элементом

Quicksort(s\_arr, one, j);

}

Для быстрой сортировки различают 3 случая: худший, лучший, и средний. В лучшем случае, если подмассивы будут делиться ровно по середине, то время будет O(n\*log2(n)), т.к дерево рекурсии будет сбалансированным, и поэтому будет выполняться log(n), Также будет необходимо пройтись по всем элементам, поэтому общее время выполнения будет составлять O(n\*log(n)). В худшем случае время будет составлять О(n^2), т.к массив может разделиться на 2 части следующим образом: 1|n-1. Тогда дерево рекурсии будет иметь n-1 вызовов, тем самым время работы будет n\*(n-1), и равно O(n^2). Сложно описать средний случай, его можно лишь оценить. 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**

void Heapsort(int arr[], int n, int i)

{

int largest = i;

// Инициализируем наибольший элемент как корень

int l = 2 \* i + 1; // левый = 2\*i + 1

int r = 2 \* i + 2; // правый = 2\*i + 2

// Если левый дочерний элемент больше корня

if (l < n && arr[l] > arr[largest])

largest = l;

// Если правый дочерний элемент больше, чем самый большой элемент на данный момент

if (r < n && arr[r] > arr[largest])

largest = r;

// Если самый большой элемент не корень

if (largest != i)

{

swap(arr[i], arr[largest]);

// Рекурсивно преобразуем в двоичную кучу затронутое поддерево

Heapsort(arr, n, largest);

}

}

void heapSort(int arr[], int n)

{

// Построение кучи (перегруппируем массив)

for (int i = n / 2 - 1; i >= 0; i--)

Heapsort(arr, n, i);

// Один за другим извлекаем элементы из кучи

for (int i = n - 1; i >= 0; i--)

{

// Перемещаем текущий корень в конец

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

// вызываем процедуру на уменьшенной куче

Heapsort(arr, i, 0);

}

}}

Пирамидальная сортировка всегда имеет время выполнения O(n\*log(n)). рассмотрим время выполнения функции heapify. В общем случае при выполнении данной функции данные будут разбиваться в дерево с log(n) уровнями, т.е всего будет log(n) вызовов. Остальные операции в функции heapify занимают O(1), значит общее время выполнения будет составлять O(log(n)). Перейдем к самой функции heapSort. Как можно видеть сначала используется цикл, который формирует дерево и выполняется за (n/2-1)\*log(n) времени. Второй цикл проходится по всем элементам и будет работать за O(n\*log(n)). Поэтому общее время выполнения будет составлять O(n\*log(n)).