Сортировка распределяет элементы в порядке, удобном для работы. Если отсортировать массив чисел в порядке убывания, то первый элемент всегда будет наибольшим, а последний наименьшим. Поэтому желательно хранить информацию упорядочено, чтобы было проще проводить над ней операции.

В данной статье вы научитесь разным техникам сортировок на языке С++. Мы затронем 7 видов:

- Пузырьковая сортировка (Bubble sort);

- Сортировка выбором (Selection sort);

- Сортировка вставками (Insertion sort);

- Быстрая сортировка (Quick sort);

- Сортировка слиянием (Merge sort);

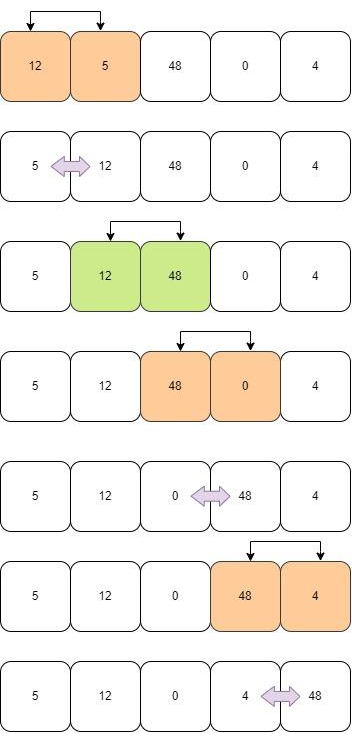
- Сортировка Шелла (Shell sort);

- Сортировка кучей (Heap sort).

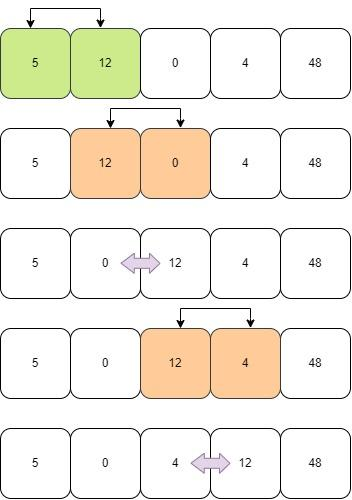
**1. Пузырьковая сортировка**

В пузырьковой сортировке каждый элемент сравнивается со следующим. Если два таких элемента не стоят в нужном порядке, то они меняются между собой местами. В конце каждой итерации (далее называем их проходами) наибольший/наименьший элемент ставится в конец списка.

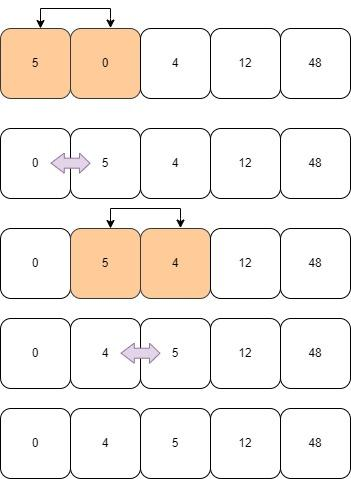
Проход 1



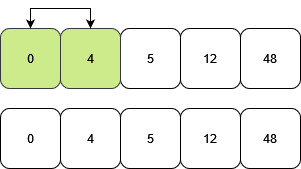
Проход 2



Проход 3



Проход 4



Сложность в лучшем случае: O(n).

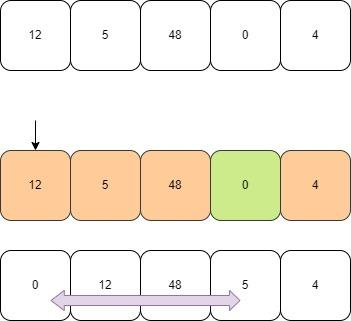
Сложность в среднем случае: O(n2).

Сложность в худшем случае: O(n2).

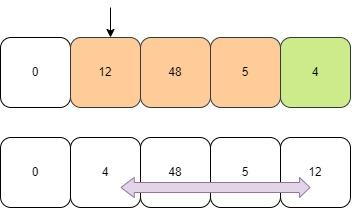
**2. Сортировка выбором**

Ищем наименьшее значение в массиве и ставим его на позицию, откуда начали проход. Потом двигаемся на следующую позицию.

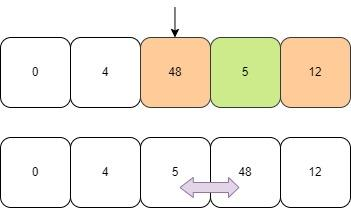
Проход 1



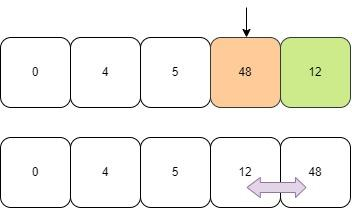
Проход 2



Проход 3



Проход 4

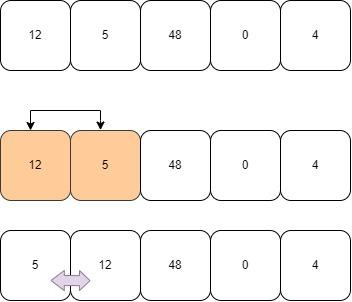


Сложность в любом случае: O(n2).

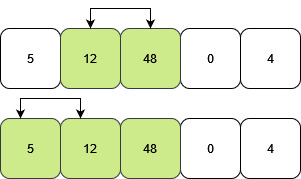
**3. Сортировка вставками**

В сортировке вставками начинаем со второго элемента. Проверяем между собой второй элемент с первым и, если надо, меняем местами. Сравниваем следующую пару элементов и проверяем все пары до нее.

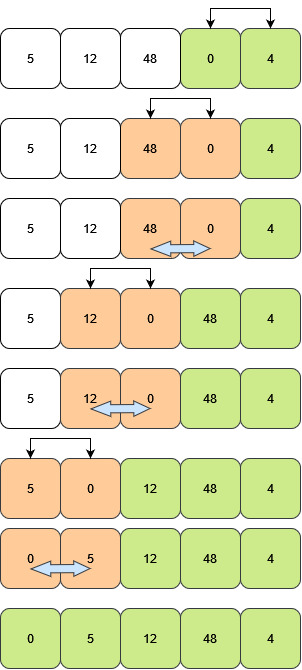
Проход 1



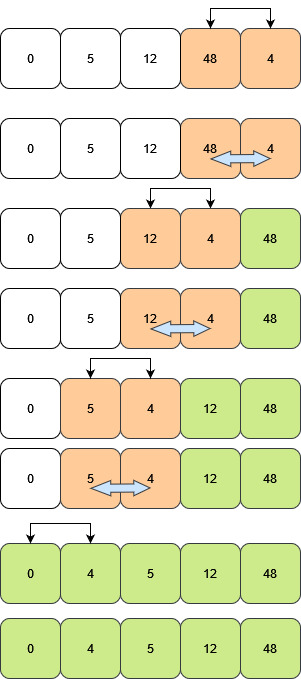
Проход 2



Проход 3



Проход 4



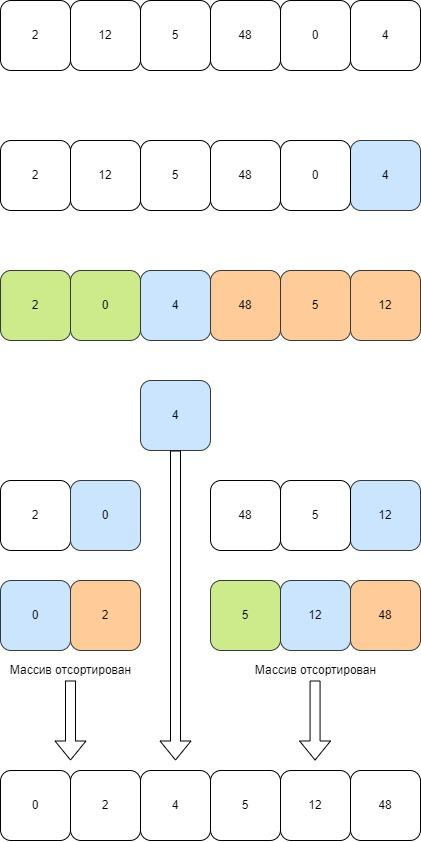
Сложность в лучшем случае: O(n).

Сложность в худшем случае: O(n2).

**4. Быстрая сортировка**

В основе быстрой сортировки лежит стратегия «разделяй и властвуй». Задача разделяется на более мелкие подзадачи. Подзадачи решаются отдельно, а потом решения объединяют. Точно так же, массив разделяется на подмассивы, которые сортируются и затем сливаются в один.

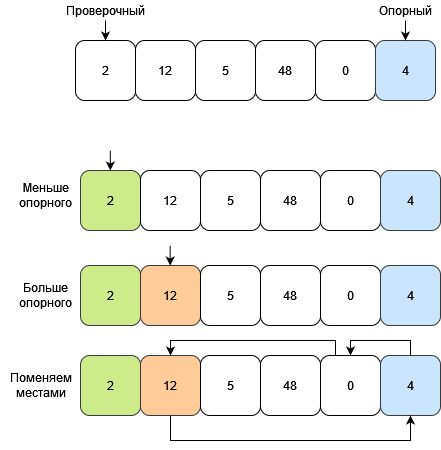
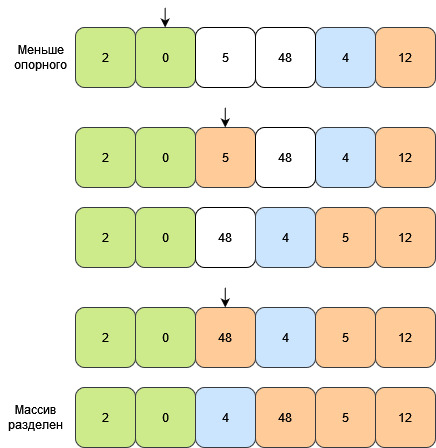
В первую очередь выбираем опорный элемент. Отметим его синим. Все значения больше опорного элемента ставятся после него, остальные — перед.



На иллюстрации массив разделяется по опорному элементу. В полученных массивах также выбираем опорный элемент и разделяем по нему.

Опорным может быть любой элемент. Мы выбираем последний в списке.

Чтобы расположить элементы большие — справа от опорного элемента, а меньшие — слева, будем двигаться от начала списка. Если число будет больше опорного, то оно ставится на его место, а сам опорный на место перед ним.

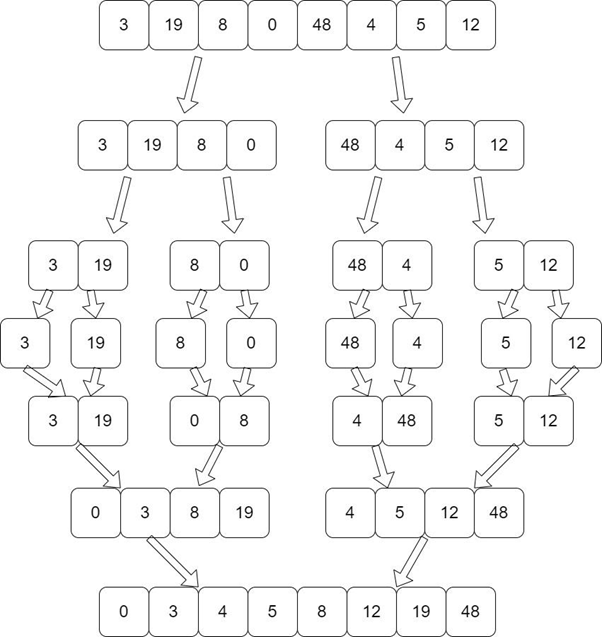


Сложность в лучшем случае: O(n\*logn).

Сложность в худшем случае: O(n2).

**5. Сортировка слиянием**

Сортировка слиянием также следует стратегии «разделяй и властвуй». Разделяем исходный массив на два равных подмассива. Повторяем сортировку слиянием для этих двух подмассивов и объединяем обратно.



Цикл деления повторяется, пока не останется по одному элементу в массиве. Затем объединяем, пока не образуем полный список.

Алгоритм сортировки состоит из четырех этапов:

Найти середину массива.

Сортировать массив от начала до середины.

Сортировать массив от середины до конца.

Объединить массив.

Алгоритм объединения массивов:

Циклично проходим по двум массивам.

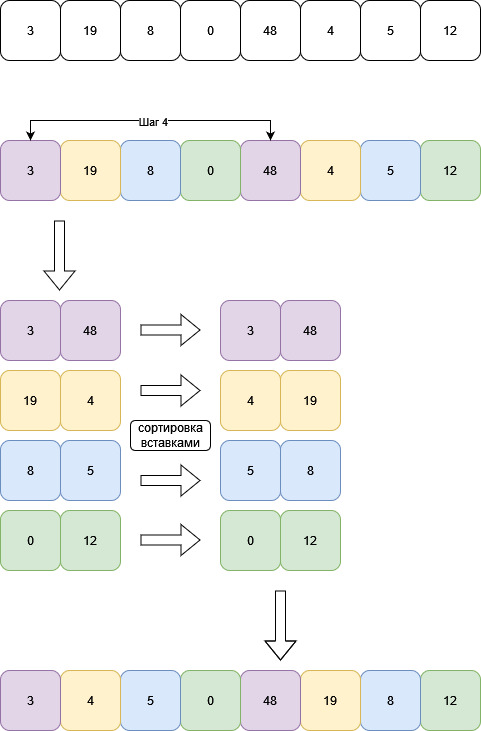
В объединяемый ставим тот элемент, что меньше.

Двигаемся дальше, пока не дойдем до конца обоих массивов.

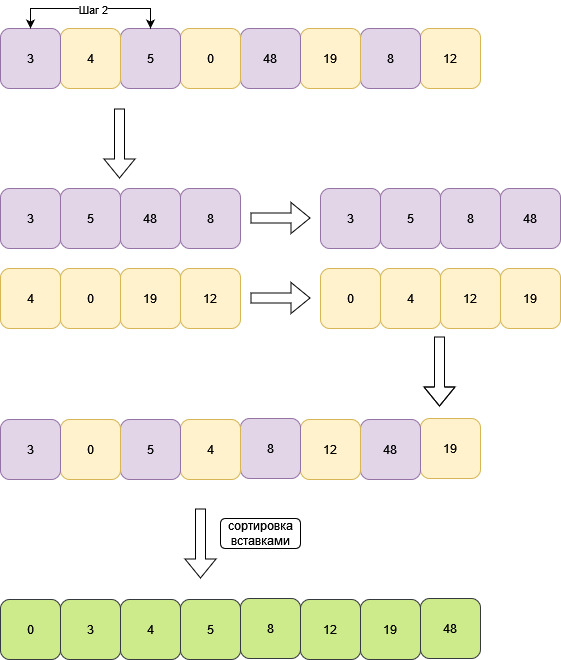
**6. Сортировка Шелла**

Алгоритм включает в себя сортировку вставками. Исходный массив размером N разбивается на подмассивы с шагомN/2. Подмассивы сортируются вставками. Затем вновь разбиваются, но уже с шагом равным N/4. Цикл повторяется. Производим целочисленное деление шага на два каждую итерацию. Когда шаг становится равен 1, массив просто сортируется вставками.

У массива размером с 8, первый шаг будет равен 4.



Уменьшаем шаг в два раза. Шаг равен 2.



Сложность в лучшем и среднем случае: O(n\*logn).

Сложность в худшем случае: O(n2).

**7. Сортировка кучей**

Исходный массив представляем в виде структуры данных кучи. Куча – это один из типов бинарного дерева.

У кучи есть следующие свойства:

Родительский узел всегда больше дочерних;

На i-ом слое 2i узлов, начиная с нуля. То есть на нулевом слое 1 узел, на первом – 2 узла, на втором – 4, и т. д. Правило для всех слоев, кроме последнего;

Слои заполняются слева направо.

После формирования кучи будем извлекать самый старший узел и ставить на конец массива.

Алгоритм сортировки кучей:

Формируем бинарное дерево из массива.

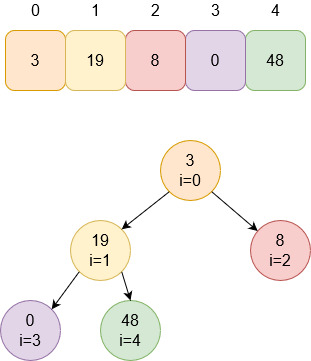
Расставляем узлы в дереве так, чтобы получилась куча (метод heapify()).

Верхний элемент помещаем в конец массива.

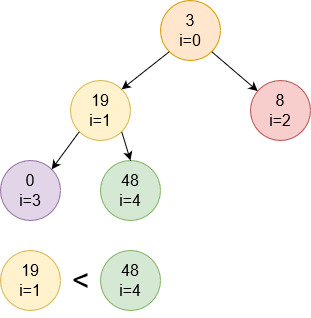
Возвращаемся на шаг 2, пока куча не опустеет.

Обращаться к дочерним узлам можно, зная, что дочерние элементы i-го элемента находятся на позициях 2\*i + 1 (левый узел) и 2\*i + 2 (правый узел).

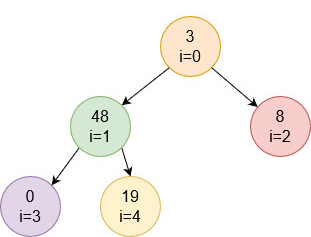
Изначальная куча:



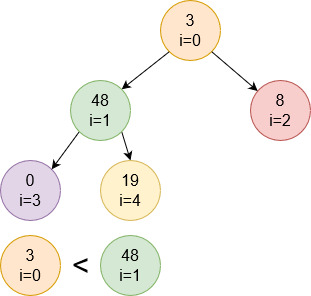
Индекс с нижним левым узлом определим по формуле n/2-1, где n – длина массива. Получается 5/2 – 1 = 2 – 1 = 1. С этого индекса и начинаем операцию heapify(). Сравним дочерние узлы 1-й позиции.



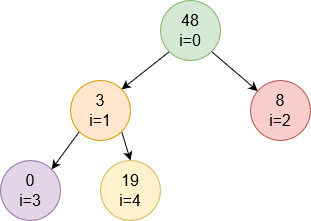
Дочерний узел оказался больше. Меняем местами с родителем.



Теперь проверяем родительский узел от позиции 1.



48 больше 3. Меняем местами.



После смены проверяем все дочерние узлы элемента, который опустили. То есть для числа 3 проводим heapify(). Так как 3 меньше 19, меняем местами.

Наибольший элемент оказался наверху кучи. Осталось поставить его в конце массива на позицию 4.

Теперь продолжаем сортировать кучу, но последний элемент игнорируем. Для этого просто будем считать, что длина массива уменьшилась на 1.

Повторяем алгоритм сортировки, пока куча не опустеет, и получаем отсортированный массив.

Сложность алгоритма в любом случае: O(n\*logn).