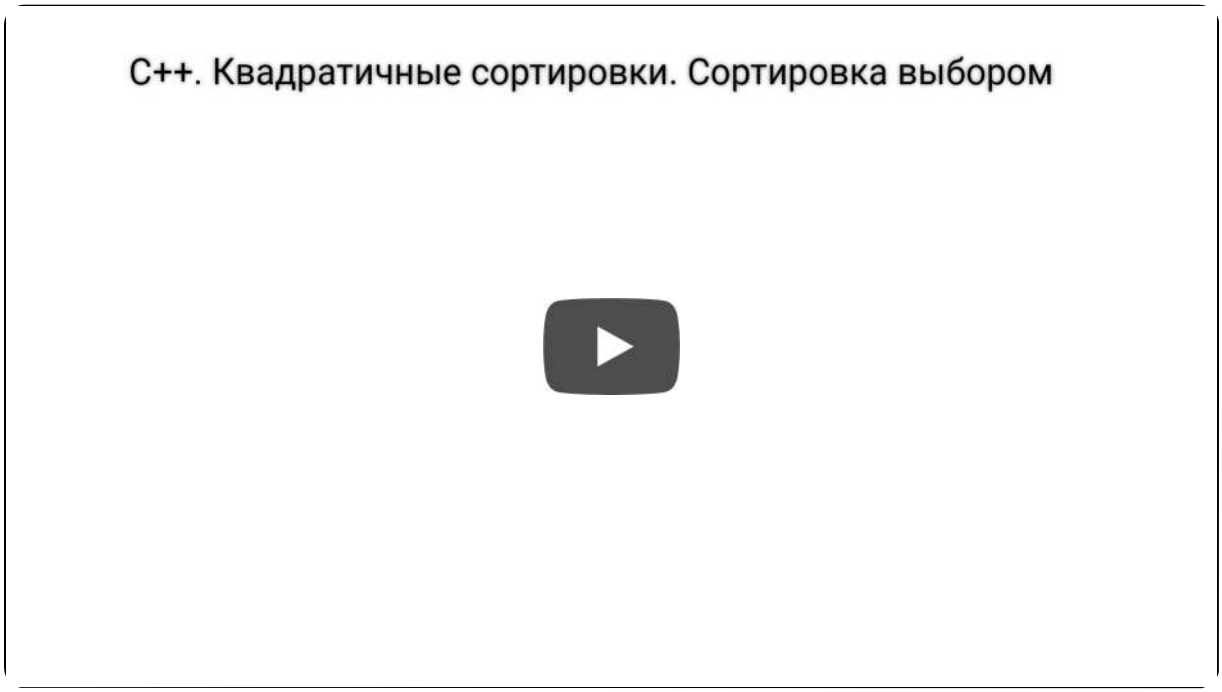
Сортировка выбором



Рассмотрим следующий алгоритм сортировки. Будем выбирать для каждой позиции, начиная с нулевой, элемент, который должен на ней стоять. Для нулевого элемента это минимальный элемент во всём списке. Для первого элемента — минимальный из всех оставшихся (то есть кроме нулевого, который уже стоит на своём месте). Для i-го — минимальный элемент списка с индексом больше или равным i. На последнем шаге мы установим на место предпоследний элемент (последний автоматически окажется на нужном месте). Таким образом мы отсортируем массив. Данный алгоритм называется **сортировка выбором**.

Рассмотрим на примере списка:

$$[4, 9, 3, 5, 2]$$

Проиллюстрируем работу алгоритма таблицей:

Переставляемые элементы	Элементы списка A
-	[4, 9, 3, 5, 2]
$A[0], A[4]$	[2, 9, 3, 5, 4]
$A[1], A[2]$	[2, 3, 9, 5, 4]
$A[2], A[4]$	[2, 3, 4, 5, 9]

$A[5], A[5]$ [2, 3, 4, 5, 9]

Оценим количество операций сравнения у данного алгоритма. Минимум в списке из n элементов можно найти проходом по всем элементам за $n-1$ действий, следующий за ним элемент за $n-2$ действий и так далее. Отсюда следует, что количество операций вычисляется следующей формулой:

$$(n-1)+(n-2)+(n-3)+\ldots+2+1=n(n-1)/2$$

Из этого следует, что вычислительная сложность алгоритма – $O(n^2)$.

Однако, алгоритм весьма эффективен по количеству операций обмена элементов. Их будет не более $n-1$. Отсюда следует, что сортировка выбором может быть полезна при упорядочивании больших объектов.

Реализация алгоритма

```
void selection_sort(vector<int> &a)
{
    for (int i = 0; i < (int)a.size() - 1; ++i) {
          int imin = i;
          for (int j = i + 1; j < a.size();++j){
              if (a[j] < a[imin])
                  imin = j;
          }
          swap(a[i], a[imin]);
    }
}
```