**Лабораторная работа №15**

**Тема работы: «**Разработка, отладка и испытание алгоритмов сортировки массивов»

**Цель:** научиться разрабатывать программы сортировки массивов

**Оснащение рабочего места:** ПК, VS, методические указания для проведения лабораторных работ, индивидуальные задания на ЛР

**КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

## Что такое пузырьковая сортировка

Пузырьковая сортировка - наверно самая простая сортировка, которую я встречал. Обычно она встречается в книгах по программированию и не выходит за ее пределы. Так как она работает намного медленнее, чем другие алгоритмы сортировки.

Но благодаря ей появились алгоритмы, которые более эффективнее чем она сама. Из таких сортировок можно отметить [шейкерную](https://codelessons.dev/ru/shejker-sortirovka-v-c-princip-raboty/) или как еще ее называют сортировка перемешиванием.

Как работает алгоритм пузырьковой сортировки

Принцип работы пузырьковой сортировки можно описать в три пункта:

* Прохождение по всему массиву;
* Сравнивание между собой пар соседних ячеек;
* Если при сравнении оказывается, что значение ячейки i больше, чем значение ячейки i + 1, то мы меняем значения этих ячеек местами;

Как создать пузырьковую сортировку

Вот что нам придется делать для создания пузырьковой сортировки:

* Создать два цикла for, чтобы проходить по всем элементам массива N раз (N это размер массива).
* Сравнивать ячейки массива, с помощью [оператора ветвления](https://codelessons.dev/ru/operator-if-else-in-cpp/) if.
* Менять местами значения ячеек.

В примере ниже мы предложили пользователю заполнить массив, который мы дальше отсортируем используя пузырьковую сортировку.



Давайте поподробнее разберем строки 16 - 24 (здесь находится пузырьковая сортировка):

В строке 16: мы создали первый цикл for.

В строке 17: аналогично был создан второй, но уже вложенный цикл.

В строке 18: происходит сравнивание двух элементов.

Если результат этого условия положительный, то мы меняем значение элементов.

Если же результат отрицателен, то мы пропускаем этап смены значений.

В строке 19: создали переменную b, чтобы менять значения ячеек digitals[i] и digitals[i + 1] местами.

Давайте посмотрим, что выведет программа выше при запуске:

**Введите 10 чисел для заполнения массива:**

**5 3 6 2 7 0 2 8 9 10**

**Массив в отсортированном виде: 0 2 2 3 5 6 7 8 9 10**

**Process returned 0 (0x0) execution time : 0.005 s**

**Press any key to continue.**

### Задание:

1. Заполните массив из 15 элементов случайными числами.
2. Отсортируйте массив используя алгоритм пузырьковой сортировки.
3. Выведите весь массив на экран.

Как следует из названия, алгоритм быстрой сортировки, как известно, сортирует массив элементов намного быстрее (два или три раза), чем различные алгоритмы сортировки.

Быстрая сортировка использует технику **«разделяй и властвуй»** , при которой мы разделяем массив вокруг элемента, известного как опорный элемент, который выбирается случайным образом из массива. Мы решаем разделы, чтобы получить упорядоченный массив. По этой причине он также известен как **алгоритм обмена разделами** .

## Работа алгоритма быстрой сортировки в C++

Алгоритм быстрой сортировки работает по принципу «разделяй и властвуй»:

* **Разделять :-**

Сначала выберите опорный элемент из массива. После этого разделите массив на два подмассива, при этом каждый элемент в левом подмассиве будет меньше или равен опорному элементу, а каждый элемент в правом подмассиве будет больше. Это будет продолжаться до тех пор, пока в подмассиве не останется только один элемент.

* **Завоевывать :**

Рекурсивно отсортируйте два подмассива, сформированных с помощью быстрой сортировки.

* **Объединить:**

Объедините уже отсортированный массив.

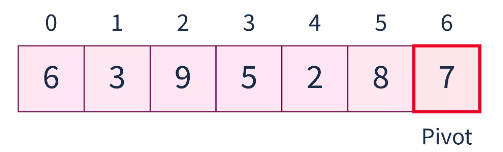
Чтобы обсудить работу алгоритма быстрой сортировки, давайте возьмем пример массива:

Array: {6, 3, 9, 5, 2, 8, 7}

1. **Выберите опорный элемент: -**

Во-первых, мы выбираем опорный элемент из элемента. Существует много разных способов выбора опорного элемента: \* Всегда выбирайте первый элемент в качестве опорного. \* Всегда выбирайте последний элемент в качестве опорного. \* Выберите случайный элемент в качестве опорного. \* Выберите срединный элемент в качестве оси.

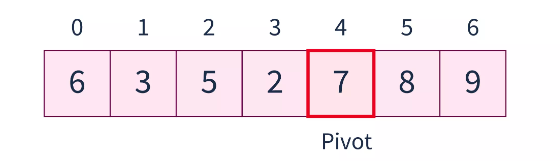
В этой статье мы будем рассматривать последний элемент массива как опорный элемент.



1. **Переставьте массив: -**

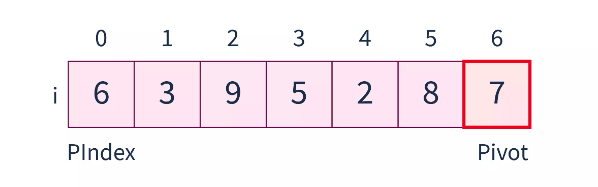
Мы изменим массив таким образом, чтобы элементы, меньшие, чем опорный элемент, располагались перед опорным элементом, а элементы, превышающие опорный элемент, располагались после опорного элемента. Таким образом, поворотный элемент займет свое фактическое положение.

В приведенном выше примере элементы[9,8] придет после[ 7 ], и [ 6 ,3 ,5 ,2 ]придет раньше [ 7 ].

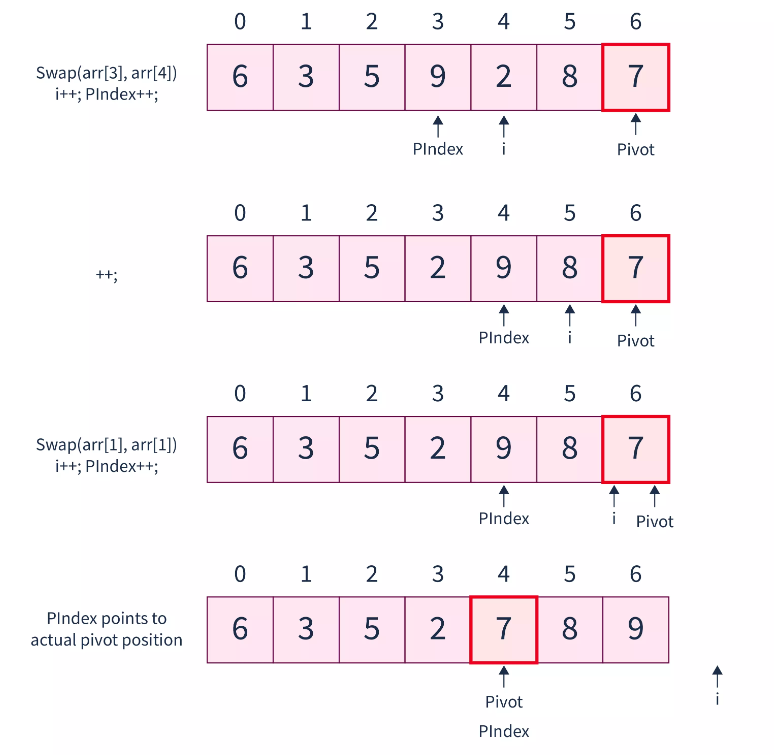
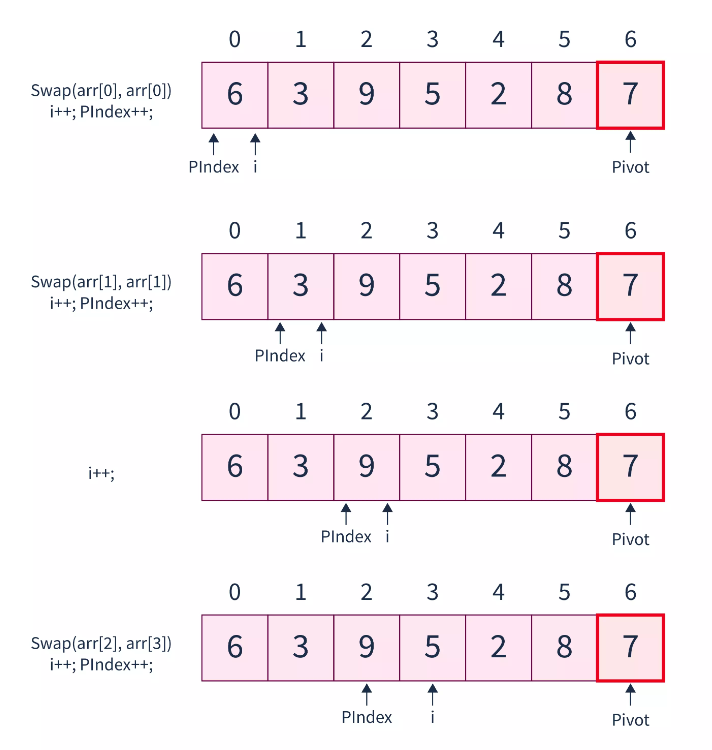


Перегруппируем массив следующим образом:

* Мы инициализируем указатель PIndex с индексом 0. В конечном итоге этот указатель будет указывать на фактическое положение поворотного элемента.

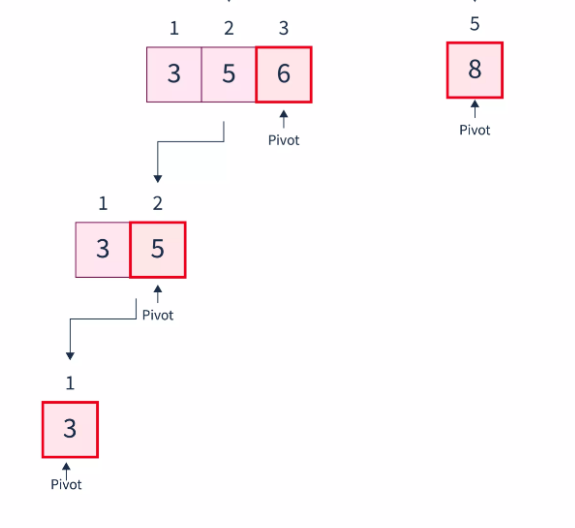
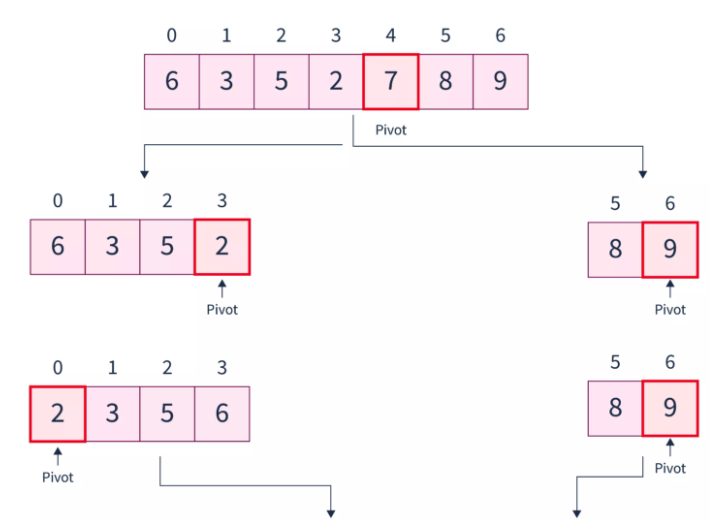


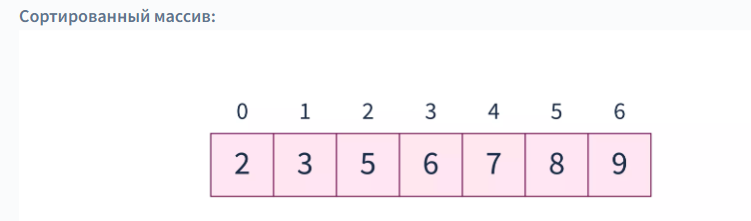
* Затем мы перемещаемся по массиву до точки Pivot с помощью указателя. Скажем i и проверим следующее условие:
* if(arr[i] <= Pivot) swap(arr[i], arr[Pindex]);
* Таким образом, мы помещаем все меньшие элементы слева от PIndex , а все более крупные элементы — справа от PIndex .



* После обхода массива и замены выполните PIndex= PIndex-1 . Таким образом, мы переставили элементы, разделив массив на два подмассива — один слева от PIndex и один справа от PIndex .

1. **Разделение подмассивов.** Мы рекурсивно передаем левый и правый подмассив сводного индекса тому же процессу, который описан выше. Поэтому в приведенном выше примере мы передаем подмассив — [от 0 до 3] с элементом поворота какарр[3]*а р р*[ 3 ]и подмассив — [от 5 до 6] со сводным элементом какарр[6]*а р р*[ 6 ]снова в качестве аргумента функции быстрой сортировки.





## Реализация быстрой сортировки в C++

* Во-первых, мы инициируем два указателя, lo и hi , на самый левый и самый правый элементы массива.
* Затем мы вызываем функцию секционирования, которая переупорядочивает массив таким образом, что все меньшие элементы остаются слева от опорного элемента, а все более крупные элементы — справа от опорного элемента.

Он возвращает окончательный сводный индекс PIndex в диапазоне [lo, hi] .

* Рекурсивно вызовите быструю сортировку для левого подмассива Pindex и правого подмассива Pindex .
* **Код С++:**
* #include <bits/stdc++.h>
* using namespace std;
* *//rearrange the elements to get the actual pivot index*
* int partition(int arr[], int low, int high, int pivot){
* int PIndex = low;
* *// Traverse the array from low to high*
* for(int i=low;i<=high;i++) {
* *// If the current element is smaller than the pivot element*
* if(arr[i]<=pivot) {
* *// Swap PIndex element with current element.*
* swap(arr[PIndex],arr[i]);
* *// Increment the pointer.*
* PIndex++;
* }
* }
* PIndex--;
* return PIndex;
* }
* *// We divide the array into two subarrays around*
* *// the pivot and recursively call for them separately.*
* void quickSort(int arr[], int low, int high){
* if(low < high) {
* int pivot = arr[high];
* *//Rearrange and get the actual pivot index*
* int PIndex = partition(arr, low, high, pivot);
* *// solve for the left and right subarrays*
* quickSort(arr, low, PIndex-1);
* quickSort(arr, PIndex+1, high);
* }
* }
* int main()
* {
* int arr[7]={6,3,9,5,2,8,7};
* int n=7;
* quickSort(arr, 0 , n-1);
* cout<<"The sorted array is: ";
* for( int i = 0 ; i < n; i++){
* cout<< arr[i]<<" ";
* }
* return 0;
* }
* **Выход :**
* The sorted array is: 2 3 5 6 7 8 9

*Задание:*

1. При помощи метода быстрой сортировки написать программу, которая будет брать массив и его элементы расставлять в порядке убывания.

**Содержание отчета**

1. Наименование и цель работы.
2. Задание на ЛР
3. Скопированный код из VS.
4. Скриншот выполненной программы
5. Ответы на контрольные вопросы.

**Контрольные вопросы**

### Какие бывают методы сортировки?

### В чем заключается их смысл?