ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 16-17

Дисциплина: Основы алгоритмизации и программирования

Тема: "Алгоритм сортировки «Быстрая сортировка»"

Выполнил: студент группы 211-721
Дерендяев Дмитрий Сергеевич
(Фамилия И.О.)

Дата, подпись 4.12.2021 _____
(Дата) (Подпись)

Проверил: Новичков Иван Константинович _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания:

_____

_____

_____

Москва

2021

Лабораторная работа №16-17
"Алгоритм сортировки «Быстрая сортировка»"

Цель: Получить практические навыки разработке алгоритмов и их программной реализации.

Понятие алгоритма:

Быстрая сортировка (англ. *quick sort*, сортировка Хоара) — один из самых известных и широко используемых алгоритмов сортировки. Среднее время работы O(n log n), что является асимптотически оптимальным временем работы для алгоритма, основанного на сравнении. Хотя время работы алгоритма для массива из n элементов в худшем случае может составить Θ(n^2), на практике этот алгоритм является одним из самых быстрых. Для этого алгоритма применяется рекурсивный метод.

Рекурсией называется ситуация, когда программа вызывает сама себя непосредственно или косвенно (через другие функции).

Идея алгоритма:

Выбираем из массива элемент, называемый опорным, и запоминаем его значение. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

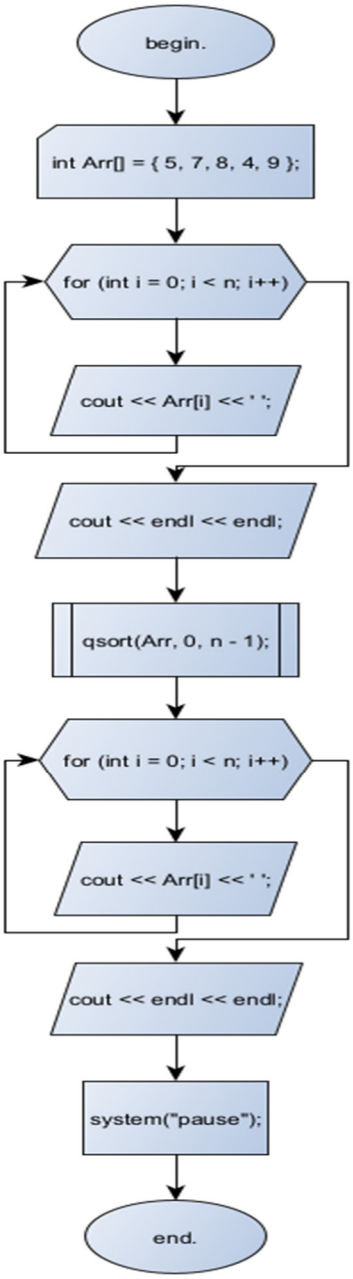
Далее начинаем двигаться от начала массива по возрастающей, а потом от конца массива по убывающей. Цель: переместить в правую часть элементы больше опорного, а в левую – элементы меньше опорного. Если во время движения по возрастающей находится элемент со значением больше опорного, то мы выходим из цикла, прибавляем единицу к индексу элемента, на котором остановились, и переходим к циклу с движением по убывающей. В этом цикле мы остаемся до тех пор, пока не находится элемент со значением меньше опорного.

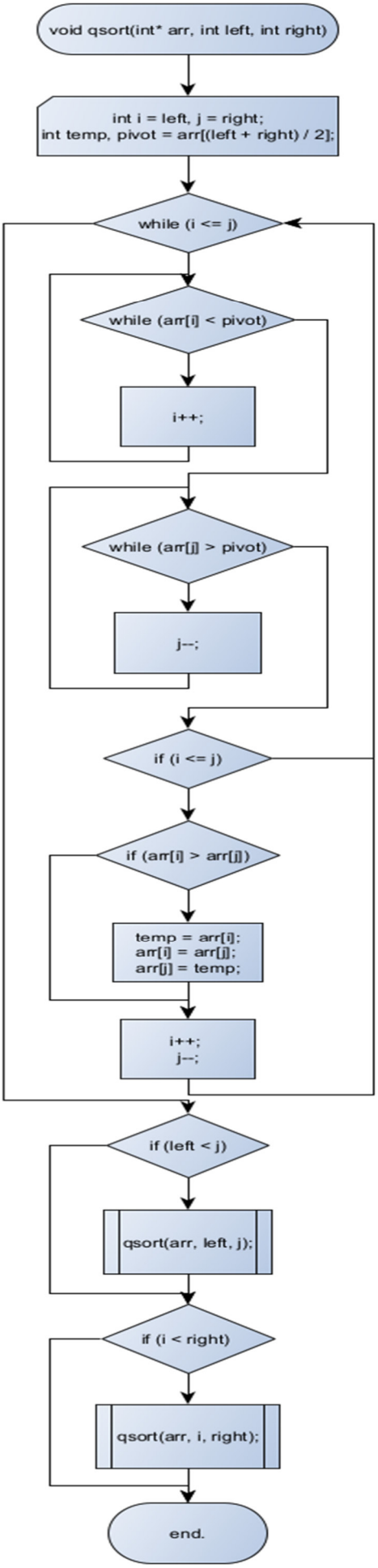
Как только такой элемент найден, мы отнимаем единицу от его индекса, и меняем значение элемента со значением элемента, на котором мы остановились в предыдущем цикле. Делаем так до тех пор, пока индекс левого элемента (найденного в первом цикле) меньше либо равен индексу правого элемента (найденного во втором цикле). В итоге получаем два подмассива (от начала до индекса правого элемента и от индекса левого элемента до конца). С этими подмассивами мы рекурсивно проделываем все то же самое, что и с большим массивом до тех пор, пока все элементы окончательно не отсортируются.

Задачи:

Необходимо выполнить и оформить описание следующих пунктов:

1. Сформулировать идею алгоритма
2. Выполнить словесное представление алгоритма
3. Выполнить полнить представление алгоритма с помощью блок схем с использованием элемента модификации и без него.
4. Выполнить программную реализацию алгоритмов на языке С с использованием параметрического цикла и цикла с предусловием.


begin.
int Arr[] = { 5, 7, 8, 4, 9 };
for (int i = 0; i < n; i++)
cout << Arr[i] << ' ';
cout << endl << endl;
qsort(Arr, 0, n - 1);
for (int i = 0; i < n; i++)
cout << Arr[i] << ' ';
cout << endl << endl;
system("pause");
end.


void qsort(int* arr, int left, int right)
int i = left, j = right;
int temp, pivot = arr[(left + right) / 2];
while (i <= j)
while (arr[i] < pivot)
i++;
while (arr[j] > pivot)
j--;
if (i <= j)
if (arr[i] > arr[j])
temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
i++;
j--;
if (left < j)
qsort(arr, left, j);
if (i < right)
qsort(arr, i, right);
end.

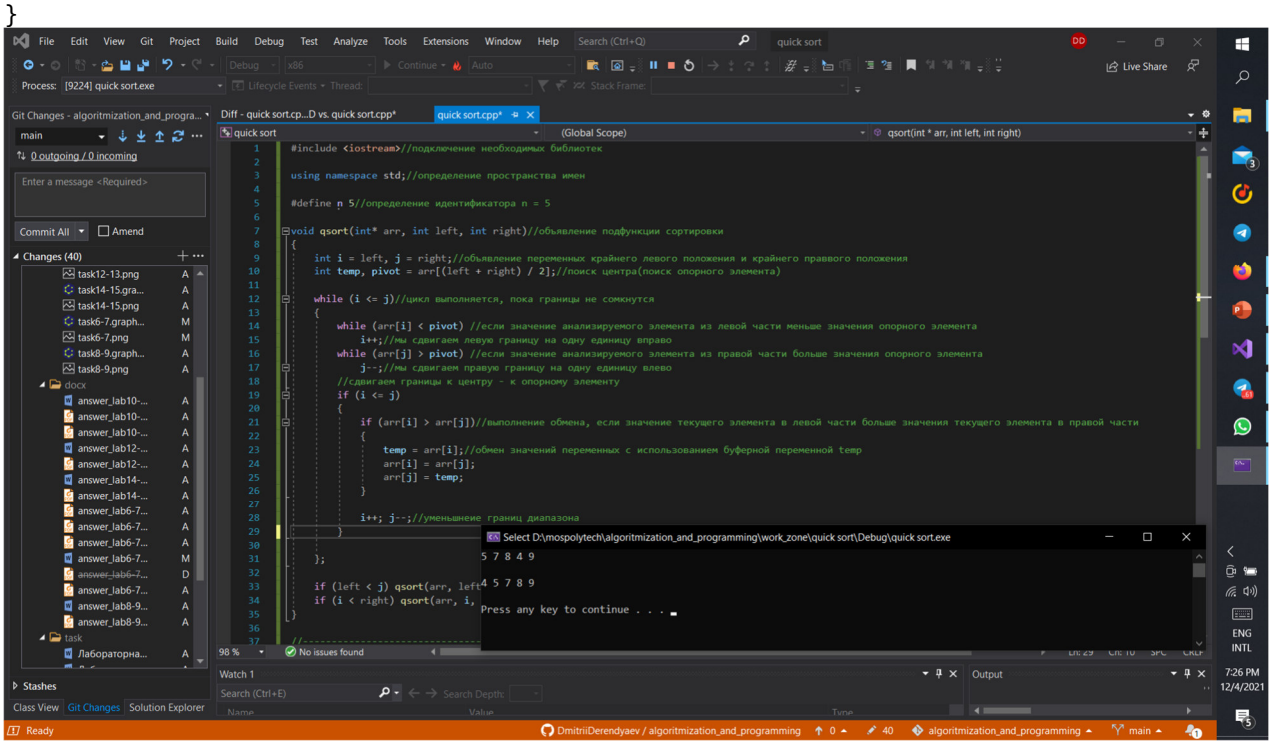
Словесное описание алгоритма(название переменных может отличаться от действительности, словесное описание, представленное ниже, отражает общую идею алгоритма):

1. **Номер опорного элемента равен (b+e)/2**
2. **Начало курсор. Если l <= r (где l = b, а r = e), то переходим к пункту 3, иначе к пункту 13**
3. **Если array[l] < piv, то переходим к пункту 4, иначе к пункту 6.**
4. **Прибавляем единицу к индексу левого элемента (l++).**
5. **Переходим к пункту 3.**
6. **Если array[r] > piv, то переходим к пункту 7, иначе к пункту 9.**
7. **Убавляем на единицу индекс правого элемента (r--)**
8. **Переходим к пункту 6.**
9. **Если l <= r, то переходим к пункту 10, иначе переходим к пункту 13.**
10. **Меняем значения элементов с индексами l и r местами (array[l] и array[r])**
11. **Прибавляем единицу к индексу левого элемента и убавляем на единицу индекс правого элемента.**
12. **Переходим к пункту 2.**
13. **Если b < r (индекс первого элемента массива меньше индекса правого элемента массива), то переходим к пункту 14, иначе к пункту 15.**
14. **Вызов курсор: (array[], b, r).**
15. **Если l < e (индекс левого элемента меньше индекса последнего элемента массива), то переходим к пункту 16, иначе к пункту 17.**
16. **Вызов курсор: (arr[], l, e).**
17. **Конец алгоритма**
18.

Листинг программы:

```
#include <iostream>//подключение необходимых библиотек

using namespace std;//определение пространства имен

#define n 5//определение идентификатора n = 5

void qsort(int* arr, int left, int right)//объявление подфункции сортировки
{
    int i = left, j = right;//объявление переменных крайнего левого положения и крайнего праввого положения
    int temp, pivot = arr[(left + right) / 2];//поиск центра(поиск опорного элемента)

    while (i <= j)//цикл выполняется, пока границы не сомкнутся
    {
        while (arr[i] < pivot) //если значение анализируемого элемента из левой части меньше значения опорного элемента
            i++;//мы сдвигаем левую границу на одну единицу вправо
        while (arr[j] > pivot) //если значение анализируемого элемента из правой части больше значения опорного элемента
            j--;//мы сдвигаем правую границу на одну единицу влево
        //сдвигаем границы к центру - к опорному элементу
        if (i <= j)
        {
            if (arr[i] > arr[j])//выполнение обмена, если значение текущего элемента в левой части больше значения текущего элемента в правой части
            {
                temp = arr[i];//обмен значений переменных с использованием буферной переменной temp
                arr[i] = arr[j];
                arr[j] = temp;
```

```
            }

            i++;
            j--;//уменьшение границ диапазона
        }

    }

    if (left < j)
        qsort(arr, left, j);//если обнаружено нарушение порядка в одной из частей,
вызывается функция сортировки определенной части
    if (i < right)
        qsort(arr, i, right);//если обнаружено нарушение порядка в одной из частей,
вызывается функция сортировки определенной части
}

//------------------------------------------------------------------------

int main()//начало главной функции
{
    int Arr[] = { 5, 7, 8, 4, 9 };//инициализация массива

    // Вывод элементов массива до сортировки
    for (int i = 0; i < n; i++)
        cout << Arr[i] << ' ';
    cout << endl << endl;

    qsort(Arr, 0, n - 1); // вызов функции сортировки

    // Вывод элементов массива после сортировки
    for (int i = 0; i < n; i++)
        cout << Arr[i] << ' ';
    cout << endl << endl;

    system("pause");//системная пауза для задержки консоли на экране

    return 0;
}
```



При необходимости, вы можете найти всю историю разработки программы на моем GitHub:

https://github.com/DmitriiDerendyaev/algoritmization_and_programming