Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение

Высшего образования

«Сибирский государственный университет

телекоммуникация и информатики»

Кафедра вычислительных систем

Курсовая работа

по дисциплине «Программирование»

на тему

«Разработка библиотеки сортировок»

Вариант 1.1

«Сортировка методом пузырька, быстрая сортировка»

Выполнил: студент I курса

ИВТ, гр. ИС-842

Пинов Д.И.

Проверил:ст.препод.

Перышкова Е.Н.

Новосибирск 2019

Содержание

Задание

Реализовать динамическую библиотеку сортировок. Алгоритмы сортировок выбираются в соответствии с вариантом задания. Проанализировать эффективность алгоритмов сортировки. Разработать демонстрационную программу, использующую созданную библиотеку.

Анализ задачи.

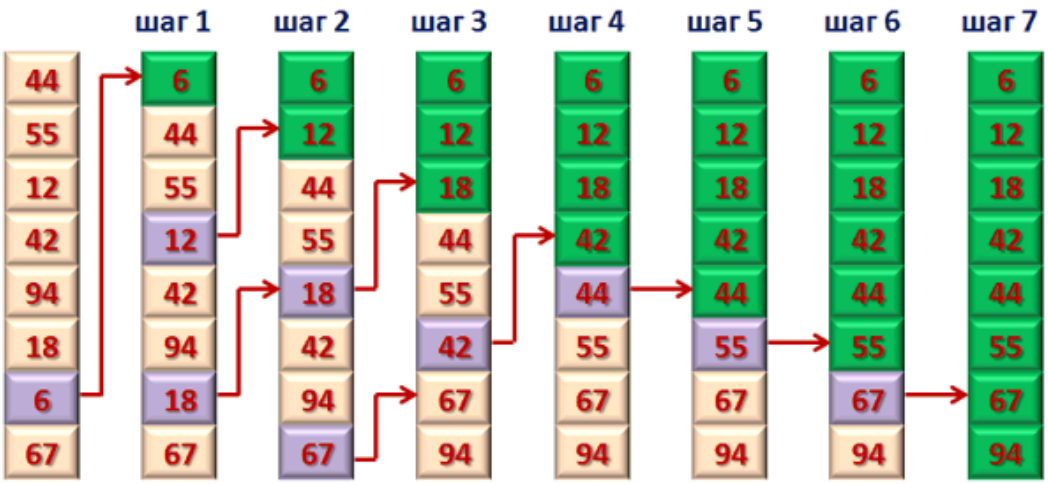
Реализация динамической библиотеки сортировок. Быстрая сортировка и сортировка пузырьком.

Динамическая библиотека - это созданная специальным образом библиотека, которая присоединяется к результирующей программе в два этапа. Первый этап, это естественно этап компиляции. На этом этапе линковщик встраивает в программу описания требуемых функций и переменных, которые присутствуют в библиотеке. Сами объектные файлы из библиотеки не присоединяются к программе. Присоединение этих объектных файлов осуществляет системный динамический загрузчик во время запуска программы. Загрузчик проверяет все библиотеки прилинкованные с программе на наличие требуемых объектных файлов, затем загружает их в память и присоединяет их в копии запущенной программы, находящейся в памяти.

Bubble Sort:

Сортировка пузырьком – простой в реализации и малоэффективный алгоритм сортировки. Алгоритм основан на принципе сравнения и обмена пары соседних элементов до тех пор, пока не будут отсортированы все элементы. Проходя по массиву, сдвигаем каждый раз наименьший элемент оставшейся последовательности к началу массива.

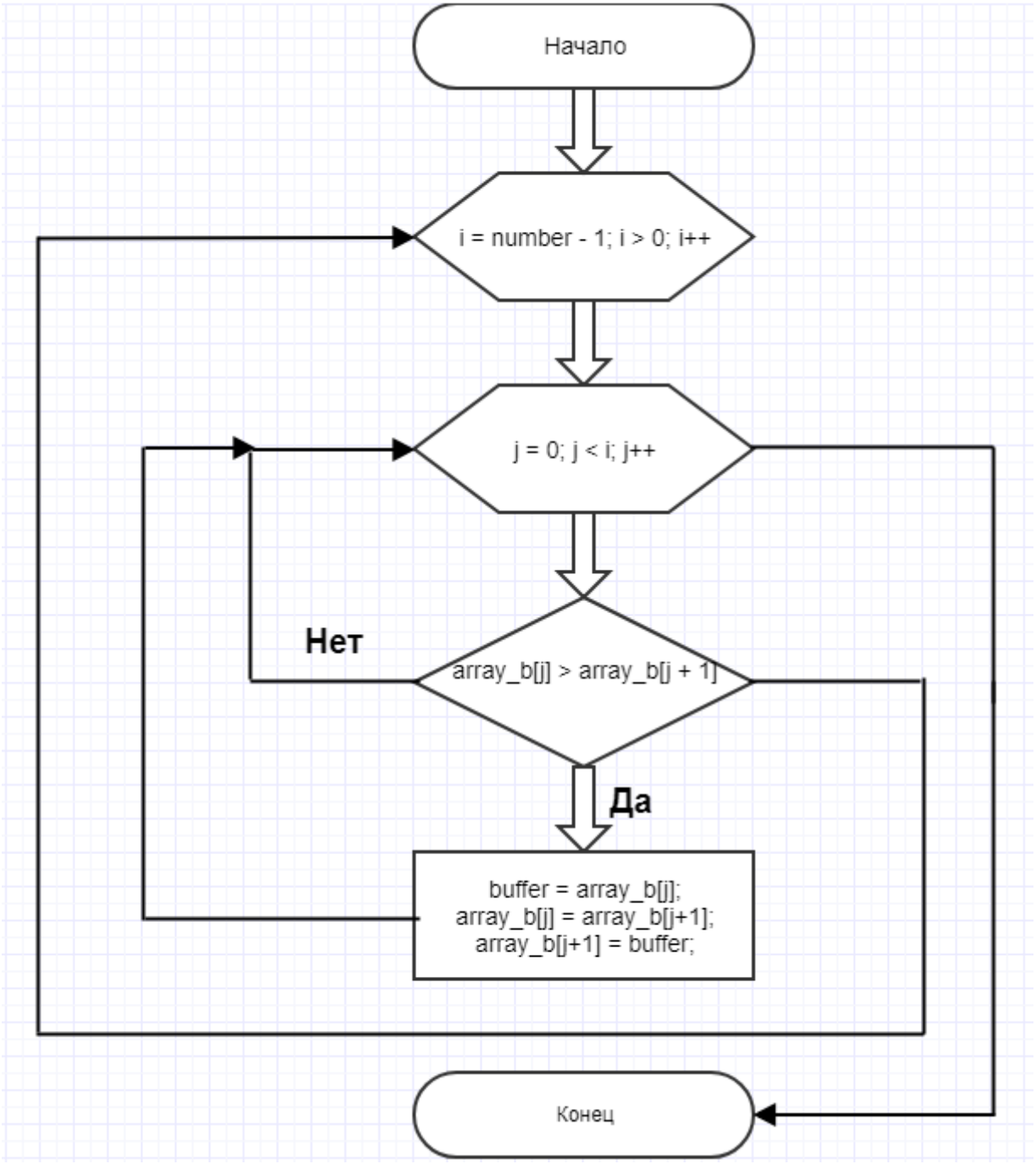
Если рассматривать массивы как вертикальные, а не горизонтальные построения, то элементы можно интерпретировать как пузырьки в банке с водой, причем вес каждого соответствует его ключу. В этом случае при каждом проходе один пузырек как бы поднимается до уровня, соответствующего его весу.



Вычислительная сложность сортировки пузырьком:

Худшее время	$O(n^2)$
Лучшее время	$O(n)$
Среднее время	$O(n^2)$
Затраты памяти	$O(1)$

Блок схема Bubble Sort



Quick Sort:

Быстрая сортировка — рекурсивный алгоритм сортировки. Среднее время работы O(n*logn) что является асимптотически оптимальным временем работы для алгоритма, основанного на сравнении.

Особенностью быстрой сортировки является операция разбиения массива на две части относительно опорного элемента. Например, если последовательность требуется упорядочить по возрастанию, то в левую часть будут помещены все элементы, значения которых меньше значения опорного элемента, а в правую элементы, чьи значения больше или равны опорному. Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов. Если длина какой-то из получившихся в результате разбиения частей превышает один элемент, то для нее нужно рекурсивно выполнить упорядочивание, т. е. повторно запустить алгоритм на каждом из отрезков.

Алгоритм быстрой сортировки включает в себя два основных этапа:

- разбиение массива относительно опорного элемента;
- рекурсивная сортировка каждой части массива.

В своей реализации сортировки, в качестве опорного элемента я выбрал крайний правый элемент массива. В общем случае это не самый эффективный вариант выбора опорного элемента.

Псевдокод Quick Sort:

```
function QuickSort(array[1;n], 0, size – 1)
   int position
   int left_copy = left
   int right_copy = right

   position = array[right]

   while left < right
      while array[left] < position
         left++
      while array[right] > position
         right--
      if left < right then
         swap(array[right], array[left])
         left++
         right--
      end if

   array[right] = position
   position = right
   right = right_copy
   left = left_copy

   if left < position then
      QuickSort (array, left, position - 1)
   end if

   if right > position then
      QuickSort (array, position + 1, right)
   end if
```

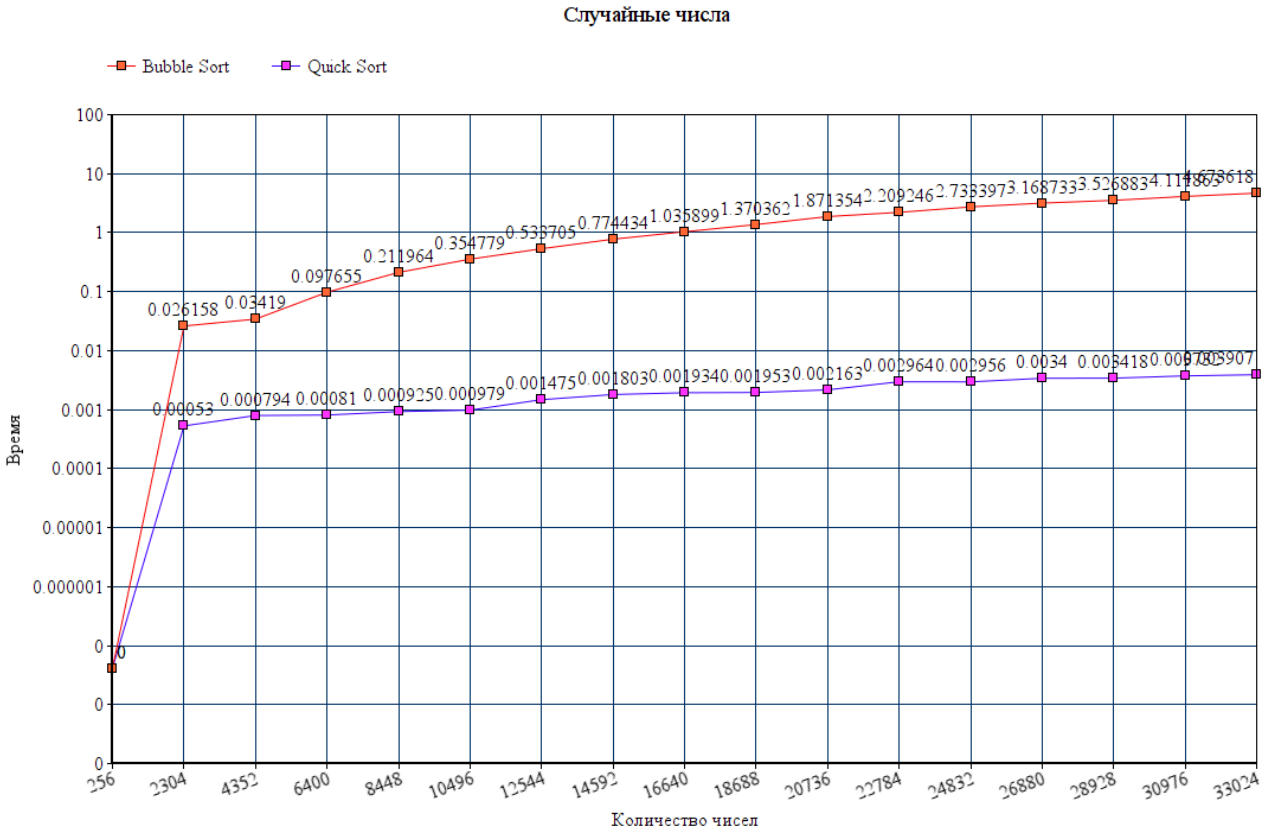
Быстрая сортировка с центральным опорным элементом.

Вычислительная сложность Quick Sort:

Худшее время	$O(n^2)$
Лучшее время	$O(n*\log n)$
Среднее время	$O(n*\log n)$
Затраты памяти	$O(n)$ вспомогательных $O(\log n)$ вспомогательных

Сравнение сортировок Bubble Sort и Quick Sort:

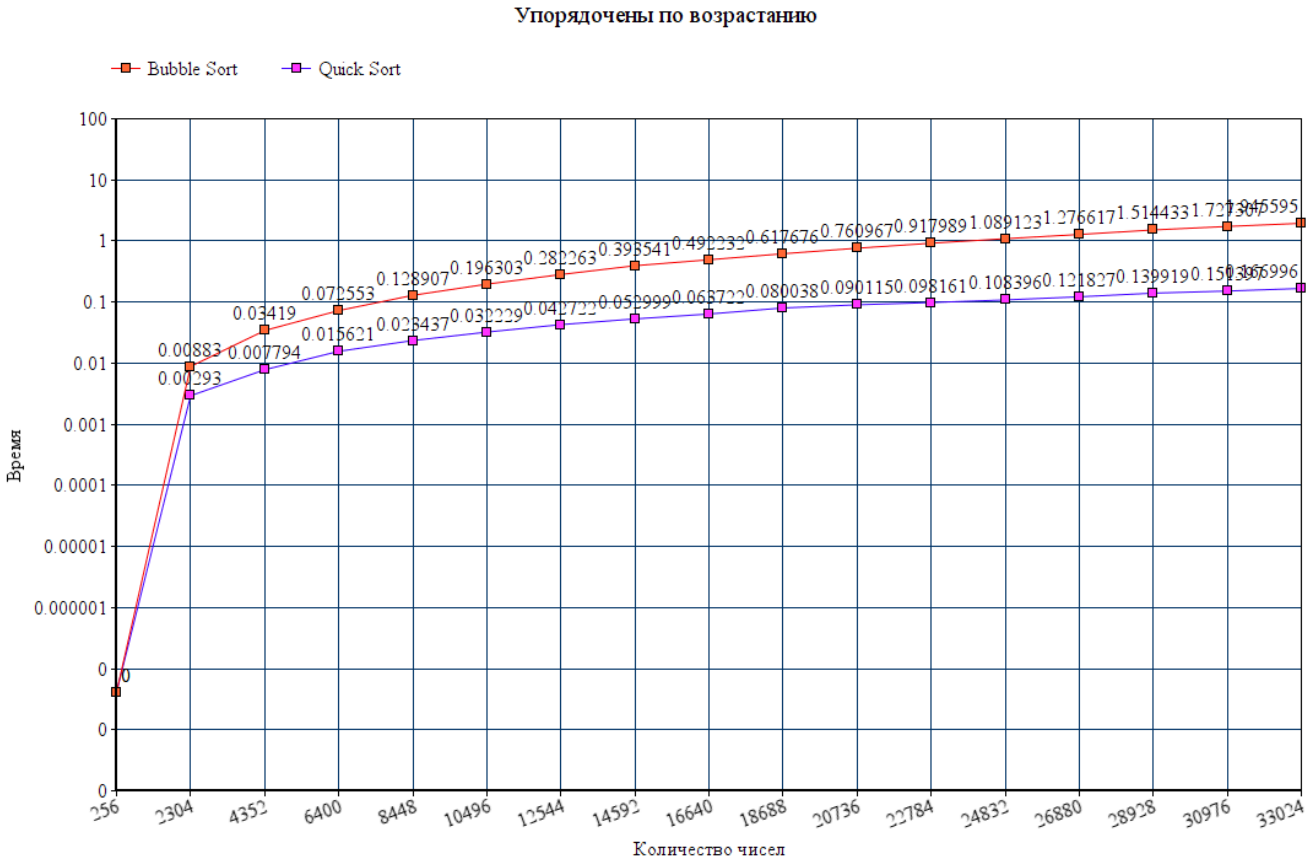
Сортировка массивов, заполненных случайными элементами.



По данному графику можно сделать вывод, что на массиве из случайных элементов “Быстрая сортировка” намного быстрее чем сортировка “Пузырьком”. На 30.000 элементов Bubble-Sort доходит до 4 секунд работы, когда Quick-Sort обрабатывает данный массив менее чем за 0.004 секунды.

Обусловлено это вычислительной сложностью алгоритмов.

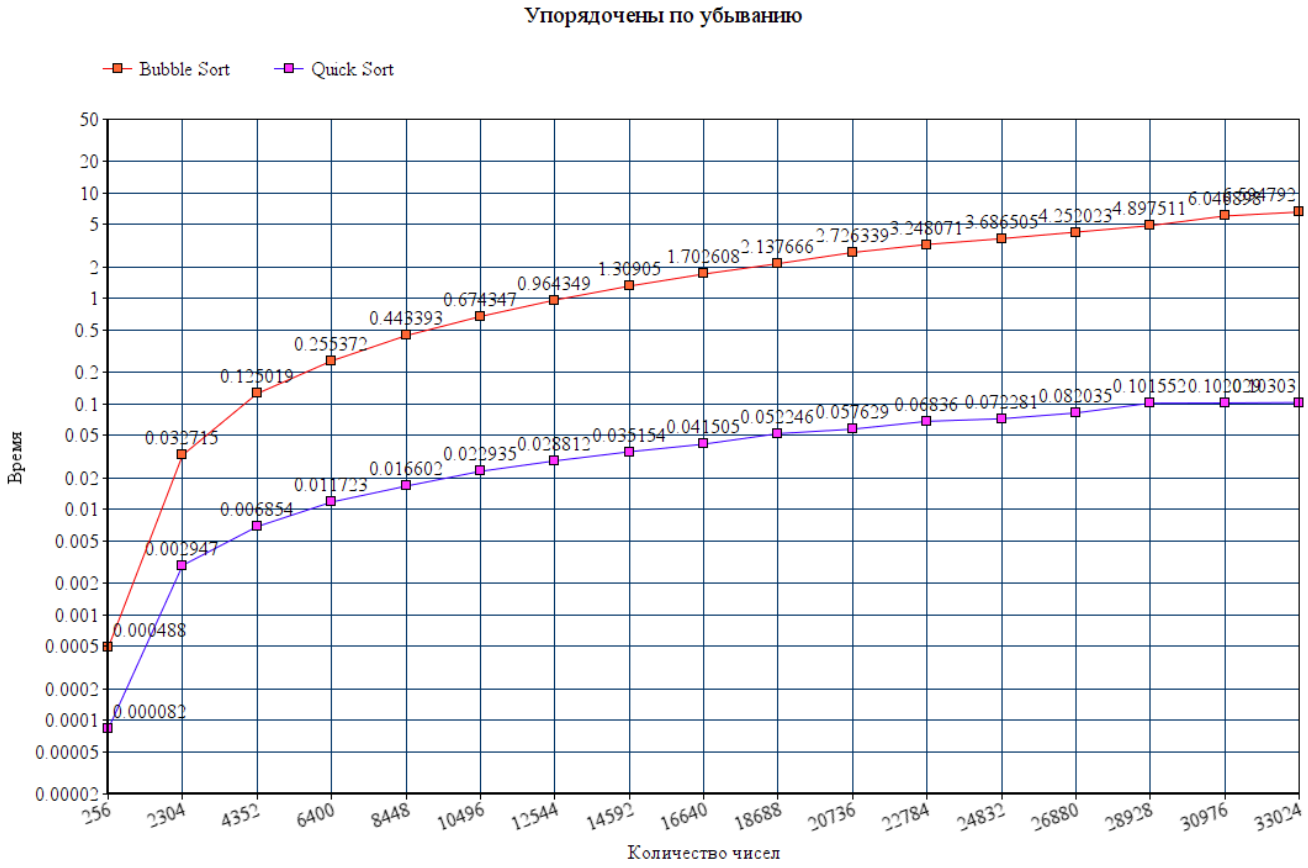
Сортировка массивов, заполненных элементами, расположенных по возрастанию:



При повторной сортировки уже отсортированного массива по возрастанию получается, что время Bubble-Sort закономерно уменьшилось, потому что алгоритму не приходится переставлять элементы, а только пройтись по всему массиву и проверить его.

А вот время Quick-Sort значительно увеличилось по сравнению с временем сортировки случайных чисел, произошло это потому что в качестве опорного элемента в моей реализации сортировки используется крайний правый элемент, то есть самое большое число, из-за чего время обработки массива значительно увеличивается.

Сортировка массивов, заполненных элементами, расположенных по убыванию:



При повторной сортировки уже отсортированного массива по убыванию, получаем, что Bubble-Sort закономерно увеличило время своей работы, так как массив отсортирован в обратном порядке. А вот Quick-Sort увеличил свою скорость по сравнению с прошлым экспериментом, опять же потому что опорный элемент крайний правый, то есть **0** в отсортированном массиве. Но все-таки работает намного медленнее, чем на массиве случайных чисел.

Листинг программы

Файл sorts.c (main file):

```
#include <stdlib.h>
#include <stdio.h>
#include "sorts.h"
#include "time.h"

int main()
{
	int number;

	for (number = 256; number <= 33025; number = number + 2048) {

		int* array_q = NULL;
		int* array_b = NULL;

		array_q = (int*)malloc(number * sizeof(int));
		array_b = (int*)malloc(number * sizeof(int));

		generator(array_q, number);
		generator(array_b, number);

		double t = time_sort();
		bubble(array_b, number);
		t = time_sort() - t;
		double t_b = t;

		t = time_sort();
		quick(array_q, 0, number - 1);
		t = time_sort() - t;
		double t_q = t;

		/*int i;
		for (i = 0; i < number - 1; i++) {
			if (array_b[i] > array_b[i + 1]) {
				printf("The array_bubble is not sorted");
				return -1;
				}
			if (array_q[i] > array_q[i + 1]) {
				printf("The array_quick is not sorted");
				return -1;
			}
		}*/

		printf("\n");
		printf("%d\t", number);
		printf("Time Bubble Sort = %.10f\n\t", t_b);
		printf("Time Quick Sort = %.10f\n", t_q);

		free(array_b);
		free(array_q);
	}

	return 0;
}
```

Файл quick.c (Quick-Sort)

```
#include "sorts.h"

void quick(int* array_q, int left, int right)
{
    int position;
   int left_copy = left;
   int right_copy = right;

   position = array_q[right];

   while (left < right) {
      while (array_q[left] < position)
         left++;
      while (array_q[right] > position)
         right--;
      if (left < right) {
          swap(&array_q[right], &array_q[left]);
         left++;
         right--;
      }
   }
   array_q[right] = position;
   position = right;
   right = right_copy;
   left = left_copy;

   if (left < position)
      quick(array_q, left, position - 1);
   if (right > position)
      quick(array_q, position + 1, right);
}
```

Файл bubble.c (Bubble-Sort)

```
#include "sorts.h"

void bubble(int *array_b, int number)
{
        int i, j;
        for (i = number - 1; i > 0; i--) {
                for (j = 0; j < i; j++) {
                        if (array_b[j] > array_b[j + 1])
                                swap(&array_b[j], &array_b[j + 1]);
                }
        }
}

void swap(int* a, int* b)
{
        int buffer;

        buffer = *a;
        *a = *b;
        *b = buffer;
}
```

Файл generator.c (Генератор чисел)

```
#include "sorts.h"

void generator(int* array, int number)
{
	int i, j;
	int min = 0, max = 1000;

	srand(time(NULL));

	for (i = 0; i < number; i++) {
		j = min + rand() % (max - min + 1);
		array[i] = j;
	}
}
```

Файл time_sort.c(Таймер)

```
#include "sorts.h"

double time_sort()
{
	struct timeval t;
	gettimeofday(&t, NULL);
	return (double)t.tv_sec+(double)t.tv_usec*1E-6;
}
```

Файл sorts.h

```
#ifndef H_SORTS
#define H_SORTS

#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

void quick(int* array_q, int left, int right);
void bubble(int* array_b, int number);
void swap(int* a, int* b);
double time_sort();
void generator(int* array, int number);

#endif
```

Файл test.c(Тесты)

```
#include "../thirdparty/ctest.h"
#include "sorts.h"
#include <stdio.h>
#include <stdlib.h>

#define number 1024

CTEST(sorts_test, bubble_sort)
{
	int* array = NULL;
	int expected = 1024;

	array = (int*)malloc(number * sizeof(int));
	generator(array, number);

	bubble(array, number);
	int i, real = 0;

	for (i = 0; i < number; i++) {
		if (array[i] <= array[i + 1])
			real++;
		else
			real = 0;
	}

	ASSERT_EQUAL(expected, real);
}

CTEST(sorts_test, quick_sort)
{
	int* array = NULL;
	int expected = 1024;

	array = (int*)malloc(number * sizeof(int));
	generator(array, number);

	quick(array, 0, number - 1);
	int i, real = 0;

	for (i = 0; i < number; i++) {
		if (array[i] <= array[i + 1])
			real++;
		else
			real = 0;
	}

	ASSERT_EQUAL(expected, real);
}

CTEST(other_test, swap)
{
	int a = 1, b = 5;

	swap(&a, &b);

	ASSERT_EQUAL(a, 5);
	ASSERT_EQUAL(b, 1);
}
```