

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Вариант 2
ТЕМА: АЛГОРИТМЫ СОРТИРОВКИ И ПОИСКА

Студент гр. 0309

Головатюк К.А.

Преподаватель

Тутуева А.В

Санкт-Петербург

2021

Постановка задачи

Реализовать алгоритмы сортировки и поиска. Сравнить время выполнения алгоритмов quicksort и bubbleSort.

Цель работы

Научиться создавать и оптимизировать алгоритмы сортировки и поиска.

Описание реализуемого класса и методов.

Среда разработки – Visual Studio

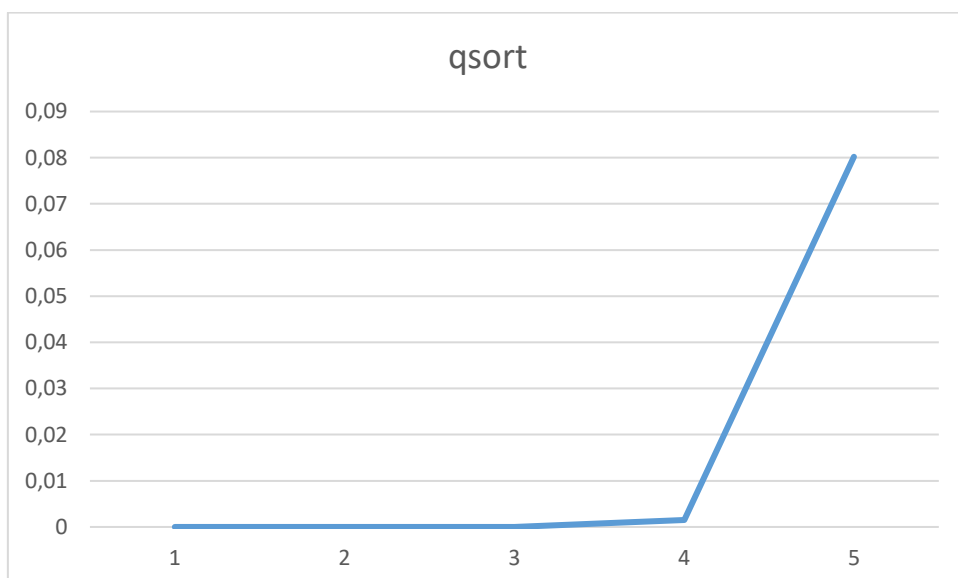
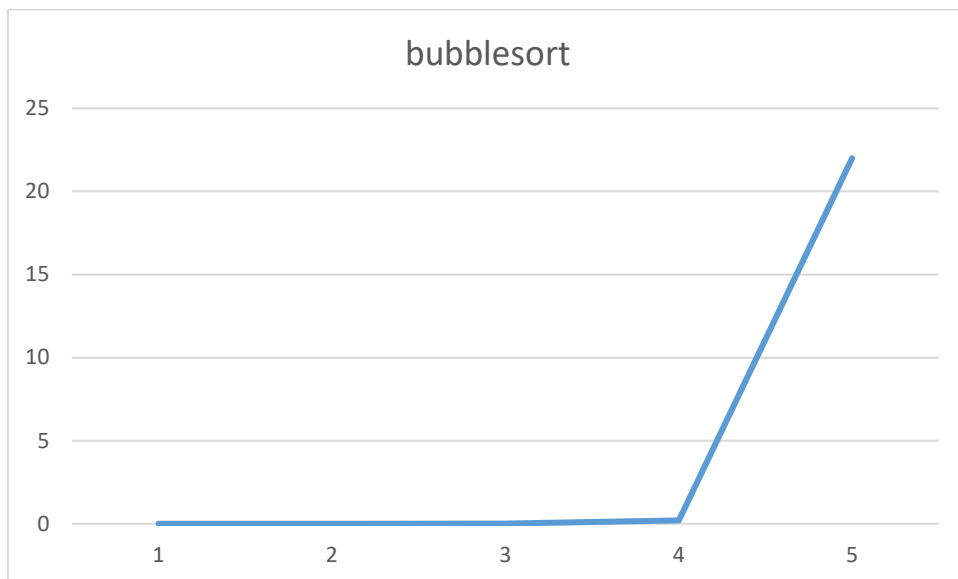
Язык программирования – C++

В ходе выполнения лабораторной работы было реализовано 4 алгоритма сортировки и 1 алгоритм поиска.

Таблица 1 – описание функций.

Название функции	Описание	Оценка временной сложности (Для методов)
int binarySearch(int* arr, int left, int right, int num)	Бинарный поиск	$O(\log(2) n)$
void bubbleSort(int* arr, int left, int right)	Сортировка пузырьком	$O(n*n)$
void quickSort(int* arr, int left, int right)	Быстрая сортировка	Лучший случай: $O(n * \log(2) n)$ Средний случай: $O(n * \log(2) n)$ Худший случай: $O(n * n)$
void bogoSort(int* arr, int left, int right)	Глупая сортировка	$O(n * n!)$
void countingSort(char* arr, int left, int right)	Сортировка подсчетом	$O(n+k)$

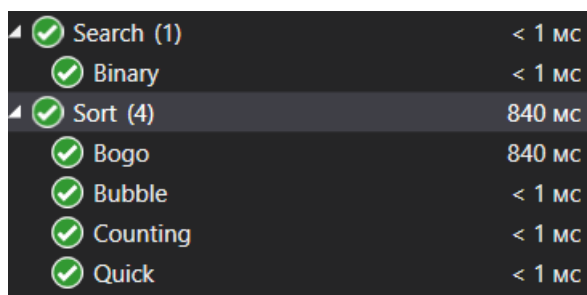
Сравнение временной сложности алгоритмов 2 и 4.



Из графиков видно , что quicksort намного быстрее чем, bubblesort.

Описание реализованных unit-тестов

Для проверки реализованных функций были написаны unit-тесты. Их названия представлены ниже.



✓ Search (1)	< 1 мс
✓ Binary	< 1 мс
✓ Sort (4)	840 мс
✓ Bogo	840 мс
✓ Bubble	< 1 мс
✓ Counting	< 1 мс
✓ Quick	< 1 мс

Пример работы :

```
int main()
{
    int arr1[10];
    int arr2[100];
    int arr3[1000];
    int arr4[10000];
    int arr5[100000];

    double qtime1[10];
    double qtime2[10];
    double qtime3[10];
    double qtime4[10];
    double qtime5[10];
    double btime1[10];
    double btime2[10];
    double btime3[10];
    double btime4[10];
    double btime5[10];

    clock_t start;
    clock_t end;
    for (int i = 0; i < 10; i++) {
        generateArr(arr1, 10);
        generateArr(arr2, 100);
        generateArr(arr3, 1000);
        generateArr(arr4, 10000);
        generateArr(arr5, 100000);

        start = clock();
        quickSort(arr1, 0, 9);
        end = clock();
        qtime1[i] = (double)(end - start) / CLOCKS_PER_SEC;

        start = clock();
        quickSort(arr2, 0, 99);
        end = clock();
        qtime2[i] = (double)(end - start) / CLOCKS_PER_SEC;
```

```
start = clock();
quickSort(arr3, 0, 999);
end = clock();
qtime3[i] = (double)(end - start) / CLOCKS_PER_SEC;

start = clock();
quickSort(arr4, 0, 9999);
end = clock();
qtime4[i] = (double)(end - start) / CLOCKS_PER_SEC;

start = clock();
quickSort(arr5, 0, 99999);
end = clock();
qtime5[i] = (double)(end - start) / CLOCKS_PER_SEC;

generateArr(arr1, 10);
generateArr(arr2, 100);
generateArr(arr3, 1000);
generateArr(arr4, 10000);
generateArr(arr5, 100000);

start = clock();
bubbleSort(arr1, 0, 9);
end = clock();
btime1[i] = (double)(end - start) / CLOCKS_PER_SEC;

start = clock();
bubbleSort(arr2, 0, 99);
end = clock();
btime2[i] = (double)(end - start) / CLOCKS_PER_SEC;

start = clock();
bubbleSort(arr3, 0, 999);
end = clock();
```

```

        end = clock();
        btime3[i] = (double)(end - start) / CLOCKS_PER_SEC;

        start = clock();
        bubbleSort(arr4, 0, 9999);
        end = clock();
        btime4[i] = (double)(end - start) / CLOCKS_PER_SEC;

        start = clock();
        bubbleSort(arr5, 0, 99999);
        end = clock();
        btime5[i] = (double)(end - start) / CLOCKS_PER_SEC;
    }

    cout << "quickSort 10: " << sumDiv(qtime1) << " seconds" << endl;
    cout << "quickSort 100: " << sumDiv(qtime2) << " seconds" << endl;
    cout << "quickSort 1000: " << sumDiv(qtime3) << " seconds" << endl;
    cout << "quickSort 10000: " << sumDiv(qtime4) << " seconds" << endl;
    cout << "quickSort 100000: " << sumDiv(qtime5) << " seconds" << endl;

    cout << "bubleSort 10: " << sumDiv(btime1) << " seconds" << endl;
    cout << "bubleSort 100: " << sumDiv(btime2) << " seconds" << endl;
    cout << "bubleSort 1000: " << sumDiv(btime3) << " seconds" << endl;
    cout << "bubleSort 10000: " << sumDiv(btime4) << " seconds" << endl;
    cout << "bubleSort 100000: " << sumDiv(btime5) << " seconds" << endl;
}

```

```

Консоль отладки Microsoft Vis
0 0 0 0 0 0 0 0 0 0
quickSort 10: 0 seconds
0 0 0 0 0 0 0 0 0 0
quickSort 100: 0 seconds
0 0 0 0 0 0 0 0 0 0
quickSort 1000: 0 seconds
0.002 0.002 0.001 0.001 0.002 0.001 0.002 0.001 0.002 0.001
quickSort 10000: 0.0015 seconds
0.077 0.079 0.082 0.079 0.082 0.081 0.082 0.081 0.08 0.079
quickSort 100000: 0.0802 seconds
0 0 0 0 0 0 0 0 0 0
bubleSort 10: 0 seconds
0 0 0 0 0 0 0 0 0 0
bubleSort 100: 0 seconds
0.001 0.002 0.002 0.001 0.001 0.001 0.001 0.002 0.001 0.001
bubleSort 1000: 0.0013 seconds
0.239 0.187 0.185 0.191 0.198 0.189 0.189 0.19 0.194 0.188
bubleSort 10000: 0.195 seconds
22.554 22.479 22.413 22.51 22.951 22.718 22.65 22.657 23.126 22.738
bubleSort 100000: 22.6796 seconds

C:\Users\Zipzap\source\repos\lab2_kirill_2511\Debug\lab2_kirill_2511.exe (процесс 1828) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрывать консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...

```

Листинг

Sort.h

```
#pragma once
#include <iostream>

int binarySearch(int* arr, int left, int right, int num);

void bubbleSort(int* arr, int left, int right);

void quickSort(int* arr, int left, int right);

void bogoSort(int* arr, int left, int right);

void countingSort(char* arr, int left, int right);
```

Sort.cpp

```
#include "Sort.h"

using namespace std;

int binarySearch(int* arr, int left, int right, int num) {
    if (left < 0)
        left = 0;
    int midd = 0;
    while (true)
    {
        midd = (left + right) / 2;

        if (num < arr[midd])
            right = midd - 1;
        else if (num > arr[midd])
            left = midd + 1;
        else
            return midd;

        if (left > right)
            return -1;
    }
    return -1;
}

void bubbleSort(int* arr, int left, int right) {
    if (left < 0)
        left = 0;
    for (int i = 0; i < right; i++) {
        for (int j = left; j < right - i; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```



```

void quickSort(int* arr, int left, int right){
    if (left < 0)
        left = 0;
    int pivot;
    int l_hold = left;
    int r_hold = right;
    pivot = arr[left];
    while (left < right)
    {
        while ((arr[right] >= pivot) && (left < right))
            right--;
        if (left != right)
        {
            arr[left] = arr[right];
            left++;
        }
        while ((arr[left] <= pivot) && (left < right))
            left++;
        if (left != right)
        {
            arr[right] = arr[left];
            right--;
        }
    }
    arr[left] = pivot;
    pivot = left;
    left = l_hold;
    right = r_hold;
    if (left < pivot)
        quickSort(arr, left, pivot - 1);
    if (right > pivot)
        quickSort(arr, pivot + 1, right);
}

```

```

void bogoSort(int* arr, int left, int right) {
    if (left < 0)
        left = 0;
    bool sorted = 0;

    while (!sorted)
    {
        sorted = 1;
        for (int i = left; i < right + 1; i++) {
            int pos = left + (rand() % (right - left + 1));

            int temp = arr[pos];
            arr[pos] = arr[i];
            arr[i] = temp;
        }

        for (int i = left; i < right; i++) {
            if (arr[i] > arr[i + 1]) {
                sorted = 0;
                break;
            }
        }
    }
}

```

```

        }
    }

}

void countingSort(char* arr, int left, int right) {
    if (left < 0)
        left = 0;
    int k = (int)arr[left];
    for (int i = left; i < right + 1; i++)
        if (k < (int)arr[i])
            k = (int)arr[i];

    int* c = new int[k];
    memset(c, 0, sizeof(int) * (k + 1));

    for (int i = left; i < right + 1; ++i) {
        ++c[(int)arr[i]];
        //cout << (int)arr[i] << " " << (char)(int)arr[i] << " " << c[(int)arr[i]] << endl;
    }

    int b = left;
    for (int i = 0; i < k + 1; ++i) {
        for (int j = 0; j < c[i]; ++j) {
            arr[b++] = (char)i;
        }
    }
}

```

main.cpp

```

#include <iostream>
#include <time.h>
#include "Sort.h"

using namespace std;

double sumDiv(double arr[10]) {
    double sum = 0;
    for (int i = 0; i < 10; i++) {
        cout << arr[i] << " ";
        sum += arr[i];
    }
    cout << endl;
    return sum / 10;
}

void generateArr(int* arr, int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 101;
    }
}

```

```

//bs qs
int main()
{
    int arr1[10];
    int arr2[100];
    int arr3[1000];
    int arr4[10000];
    int arr5[100000];

    double qtime1[10];
    double qtime2[10];
    double qtime3[10];
    double qtime4[10];
    double qtime5[10];
    double btime1[10];
    double btime2[10];
    double btime3[10];
    double btime4[10];
    double btime5[10];

    clock_t start;
    clock_t end;
    for (int i = 0; i < 10; i++) {
        generateArr(arr1, 10);
        generateArr(arr2, 100);
        generateArr(arr3, 1000);
        generateArr(arr4, 10000);
        generateArr(arr5, 100000);

        start = clock();
        quickSort(arr1, 0, 9);
        end = clock();
        qtime1[i] = (double)(end - start) / CLOCKS_PER_SEC;

        start = clock();
        quickSort(arr2, 0, 99);
        end = clock();
        qtime2[i] = (double)(end - start) / CLOCKS_PER_SEC;

        start = clock();
        quickSort(arr3, 0, 999);
        end = clock();
        qtime3[i] = (double)(end - start) / CLOCKS_PER_SEC;

        start = clock();
        quickSort(arr4, 0, 9999);
        end = clock();
        qtime4[i] = (double)(end - start) / CLOCKS_PER_SEC;

        start = clock();
        quickSort(arr5, 0, 99999);
        end = clock();
    }
}

```

```

qtime5[i] = (double)(end - start) / CLOCKS_PER_SEC;

generateArr(arr1, 10);
generateArr(arr2, 100);
generateArr(arr3, 1000);
generateArr(arr4, 10000);
generateArr(arr5, 100000);

start = clock();
bubbleSort(arr1, 0, 9);
end = clock();
btime1[i] = (double)(end - start) / CLOCKS_PER_SEC;

start = clock();
bubbleSort(arr2, 0, 99);
end = clock();
btime2[i] = (double)(end - start) / CLOCKS_PER_SEC;

start = clock();
bubbleSort(arr3, 0, 999);
end = clock();
btime3[i] = (double)(end - start) / CLOCKS_PER_SEC;

start = clock();
bubbleSort(arr4, 0, 9999);
end = clock();
btime4[i] = (double)(end - start) / CLOCKS_PER_SEC;

start = clock();
bubbleSort(arr5, 0, 99999);
end = clock();
btime5[i] = (double)(end - start) / CLOCKS_PER_SEC;
}

cout << "quickSort 10: " << sumDiv(qtime1) << " seconds" << endl;
cout << "quickSort 100: " << sumDiv(qtime2) << " seconds" << endl;
cout << "quickSort 1000: " << sumDiv(qtime3) << " seconds" << endl;
cout << "quickSort 10000: " << sumDiv(qtime4) << " seconds" << endl;
cout << "quickSort 100000: " << sumDiv(qtime5) << " seconds" << endl;

cout << "bubleSort 10: " << sumDiv(btime1) << " seconds" << endl;
cout << "bubleSort 100: " << sumDiv(btime2) << " seconds" << endl;
cout << "bubleSort 1000: " << sumDiv(btime3) << " seconds" << endl;
cout << "bubleSort 10000: " << sumDiv(btime4) << " seconds" << endl;
cout << "bubleSort 100000: " << sumDiv(btime5) << " seconds" << endl;

}

```