Лабораторная работа 4

Исследование и оценка алгоритмов сортировки

Цель работы: Разработка программ, реализующих различные алгоритмы сортировки, и оценка их временной и пространственной с сложности.

Вариант: 9.

Условие: Составить две программы, которые реализуют алгоритмы ускоренной сортировки пузырьком и выбором. Исходные данные задавать с помощью датчика случайных чисел.

Метод ускоренной сортировки пузырьком

Асимтотическая сложность алгоритма

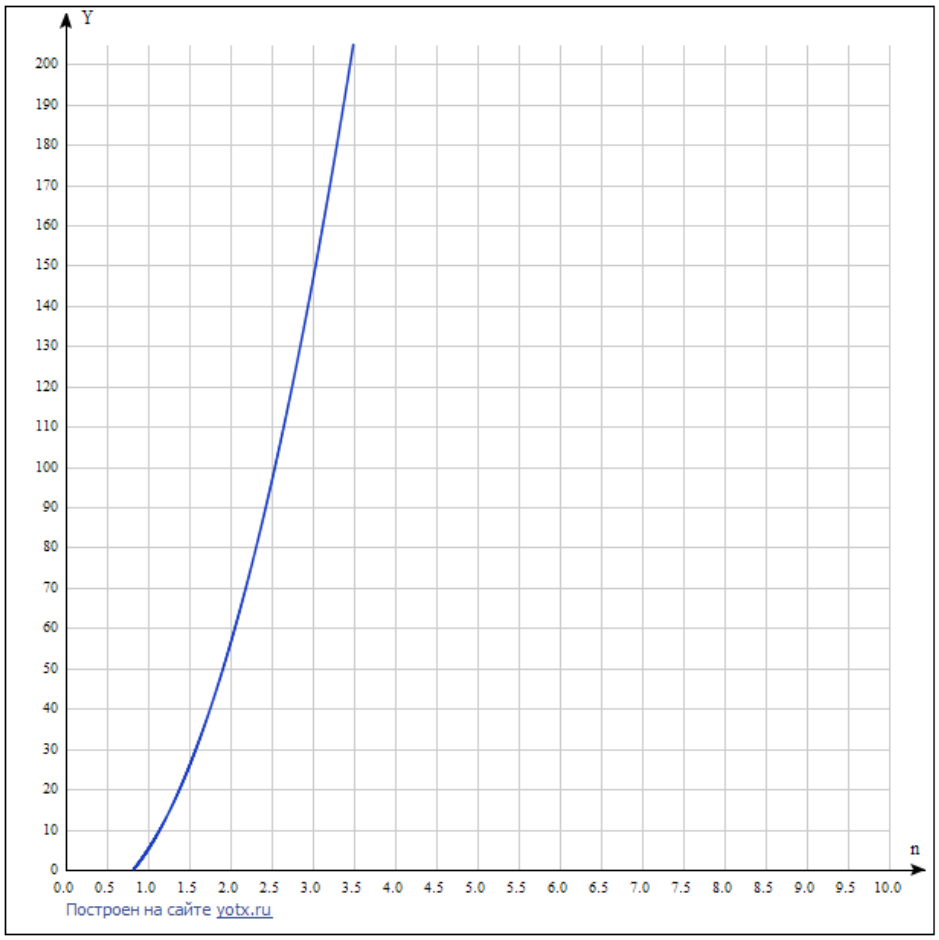
1 + 1 + 1 + n*2 + (n – 1)*3 + (n – 1) * 1 + [I = 1, I = n - 1]Σ(1 + (I + 1)*3 + I*3 + I*(1 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 1))+ (n – 1)*2 = 3 + 2*n + 3*n – 3 + n – 1 + [I = 1, I = n - 1]Σ(1 + 3*I + 3 + 3*I + I*33) + 2*n – 2 = 8*n – 3 + [I = 1, I = n – 1]Σ(39*I + 4) = [S(N) = (a1 + aN) / 2 * N, N = n – 1 , a1 = 39*1 + 4 = 43, aN = 39*(n – 1) + 4 = 39*n – 35] = 8*n – 3

+ (43 + 39*n – 35) / 2 * (n – 1) = O(n^2)

Таблица зависимости времени выполнения алгоритма от объема исходных данных

Время выполнения(в секундах)	Объем исходных данных
15.7822	Массив с 10000 элементами
9.06402	Массив с 7500 элементами
4.02566	Массив с 5000 элементами
1.04546	Массив с 2500 элементами
0.172659	Массив с 1000 элементами

График зависимости времени выполнения алгоритма от объема исходных данных



Код алгоритма

```
13  //*****************************************************************************
14  template<class ElementType>
15  void BubbleArraySortingMethod<ElementType>::sortArray(const Array<ElementType> &sortedArray) const
16  {
17      bool isAlreadySorted;
18      for (int i = sortedArray.getAmountOfElements() - 1; i > 0; i--)
19      {
20          isAlreadySorted = true;
21          for (int j = 0; j < i; j++)
22          {
23              if (sortedArray[j] > sortedArray[j + 1])
24              {
25                  ElementType tempHolder = sortedArray[j];
26                  sortedArray[j] = sortedArray[j + 1];
27                  sortedArray[j + 1] = tempHolder;
28                  isAlreadySorted = false;
29              }
30          }
31          if (isAlreadySorted)
32          {
33              return;
34          }
35      }
36  }
37  //*****************************************************************************
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 10000 элементами

```
    //Quick bubble sort for array with 10000 elements:
    int amountOfElementsOfArray = 10000;
    const double *valuesOfElementsOfArray
        = elementsGenerator->generateElements(amountOfElementsOfArray);
    Array<double> *array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
    delete[] valuesOfElementsOfArray;

    ArraySortingMethod<double> *arraySortingMethod = new BubbleArraySortingMethod<double>();
    array->setArraySortingMethod(arraySortingMethod);

    std::function<void()> *executedFunction = new std::function<void()>([array]()
    {
        array->sortElements();
    });

    const MeasurerOfTimeOfExecution * const measurerOfTimeOfExecution = new MeasurerOfTimeOfExecution();
    double amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);

    cout << "Time of execution of quick bubble sorting of array with 10000 elements: " << amountOfSecondOfExecution << "\n";

    delete executedFunction;
    delete array;
```

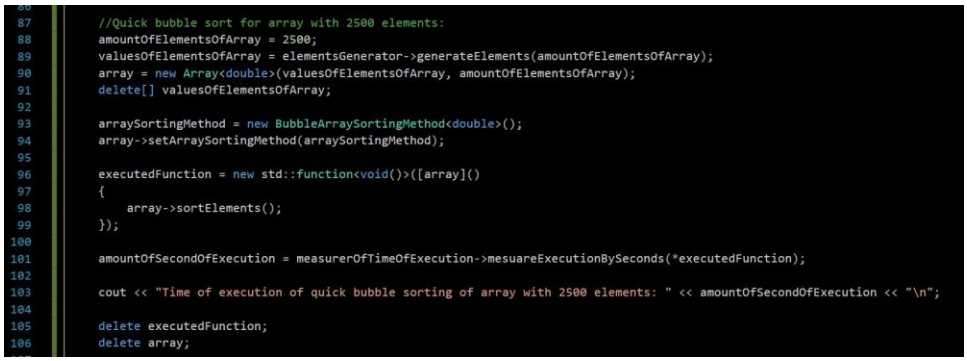
Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 7500 элементами

```
    //Quick bubble sort for array with 7500 elements:
    amountOfElementsOfArray = 7500;
    valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
    array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
    delete[] valuesOfElementsOfArray;

    arraySortingMethod = new BubbleArraySortingMethod<double>();
    array->setArraySortingMethod(arraySortingMethod);

    executedFunction = new std::function<void()>([array]()
    {
        array->sortElements();
    });

    amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);

    cout << "Time of execution of quick bubble sorting of array with 7500 elements: " << amountOfSecondOfExecution << "\n";

    delete executedFunction;
    delete array;
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 5000 элементами

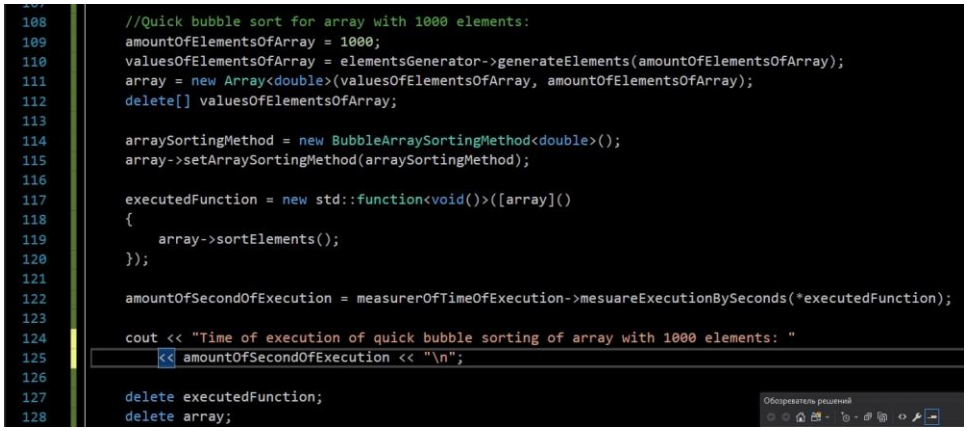
```
    //Quick bubble sort for array with 5000 elements:
    amountOfElementsOfArray = 5000;
    valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
    array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
    delete[] valuesOfElementsOfArray;

    arraySortingMethod = new BubbleArraySortingMethod<double>();
    array->setArraySortingMethod(arraySortingMethod);

    executedFunction = new std::function<void()>([array]()
    {
        array->sortElements();
    });

    amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);

    cout << "Time of execution of quick bubble sorting of array with 5000 elements: " << amountOfSecondOfExecution << "\n";

    delete executedFunction;
    delete array;
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 2500 элементами



```
//Quick bubble sort for array with 2500 elements:
amountOfElementsOfArray = 2500;
valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
delete[] valuesOfElementsOfArray;

arraySortingMethod = new BubbleArraySortingMethod<double>();
array->setArraySortingMethod(arraySortingMethod);

executedFunction = new std::function<void()>([array]()
{
    array->sortElements();
});

amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);

cout << "Time of execution of quick bubble sorting of array with 2500 elements: " << amountOfSecondOfExecution << "\n";

delete executedFunction;
delete array;
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 1000 элементами



```
//Quick bubble sort for array with 1000 elements:
amountOfElementsOfArray = 1000;
valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
delete[] valuesOfElementsOfArray;

arraySortingMethod = new BubbleArraySortingMethod<double>();
array->setArraySortingMethod(arraySortingMethod);

executedFunction = new std::function<void()>([array]()
{
    array->sortElements();
});

amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);

cout << "Time of execution of quick bubble sorting of array with 1000 elements: "
     << amountOfSecondOfExecution << "\n";

delete executedFunction;
delete array;
```

<u>Метод сортировки выбором</u>

Асимтотическая сложность алгоритма

$1 + 1 + 1 + n*(1 + 1) + (n - 1)*(1 + 1 + 1) + [I = 1, I = n - 1]\Sigma(1 + 1 + 1 + (1 + 1 + 1)$

$*(I + 1) + I*(1 + 1 + 1) + I*(1 + 2 + 1 + 1 + 2 + 1 + 1 + 1 + 1 + 1)) + (1 + 1 + 1$

$+ 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1)*(n - 1)$

$= 3 + 2*n + 3*(n - 1) + 22*(n - 1) + [I = 1, I = n - 1]\Sigma(3 + 3*(I + 1) + 3*I + 12*I)$

$= 3 + 2*n + 25*(n - 1) + [I = 1, I = n - 1]\Sigma(18*I + 6)$

$= [S(N) = (a1 + aN) / 2 * N, a1 = 18*1 + 6 = 24, aN = 18*(n - 1) + 6, N = n - 1]$

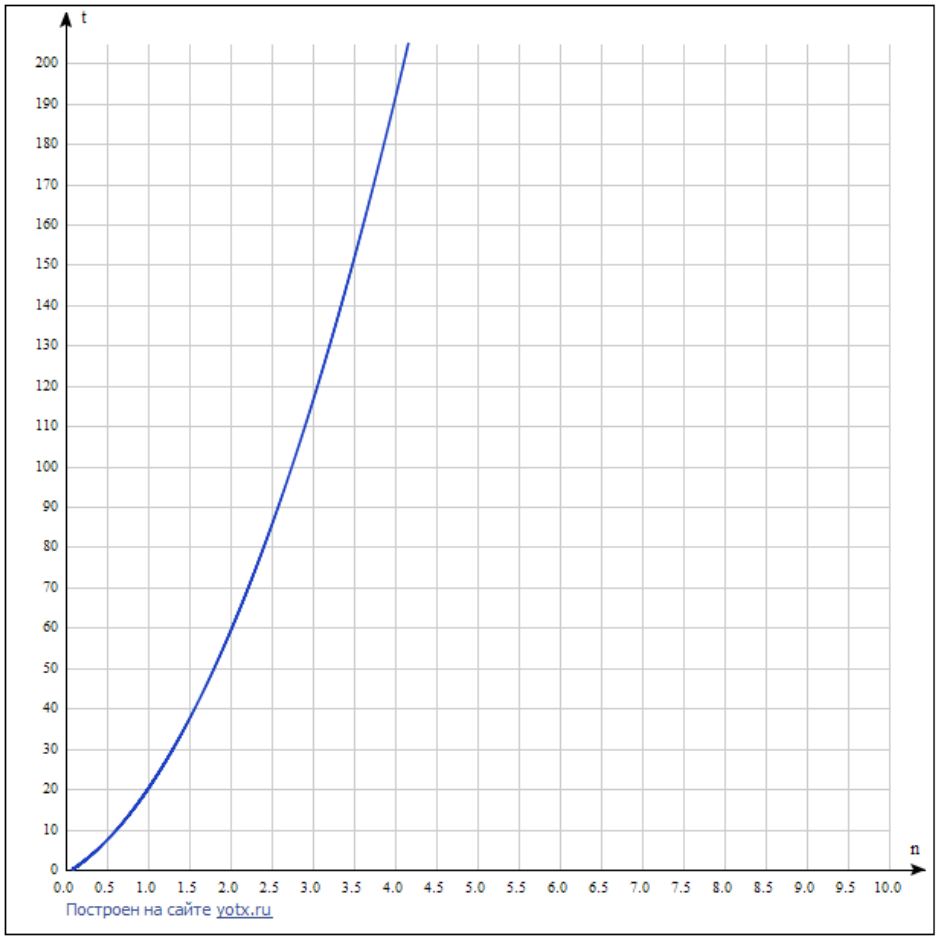
$= 3 + 2*n + 25*n - 25 + (24 + 18*(n - 1) + 6) / 2 * (n - 1)$

$= 27*n - 22 + 15 + (9*n - 6)*(n - 1) = O(n^2)$

Таблица зависимости времени выполнения алгоритма от объема исходных данных

Время выполнения(в секундах)	Объем исходных данных
7.69175	Массив с 10000 элементами
4.23196	Массив с 7500 элементами
1.95373	Массив с 5000 элементами
0.4915	Массив с 2500 элементами
0.0792789	Массив с 1000 элементами

График зависимости времени выполнения алгоритма от объема исходных данных



Код алгоритма

```cpp
//*****************************************************************************
template<class ElementType>
void SelectionArraySortingMethod<ElementType>::sortArray(const Array<ElementType> &sortedArray) const
{
    int indexOfMaxElement;
    for (int i = sortedArray.getAmountOfElements() - 1; i > 0; i--)
    {
        indexOfMaxElement = i;
        for (int j = 0; j < i; j++)
        {
            if (sortedArray[j] >= sortedArray[indexOfMaxElement])
            {
                indexOfMaxElement = j;
            }
        }

        if (indexOfMaxElement != i)
        {
            ElementType tempHolder = sortedArray[indexOfMaxElement];
            sortedArray[indexOfMaxElement] = sortedArray[i];
            sortedArray[i] = tempHolder;
        }
    }
}
//*****************************************************************************
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 10000 элементами

```cpp
    //Selection sort for array with 10000 elements:
    amountOfElementsOfArray = 10000;
    valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
    array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
    delete[] valuesOfElementsOfArray;

    arraySortingMethod = new SelectionArraySortingMethod<double>();
    array->setArraySortingMethod(arraySortingMethod);

    executedFunction = new std::function<void()>([array]()
    {
        array->sortElements();
    });

    amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);

    cout << "Time of execution of selection sorting of array with 10000 elements: "
        << amountOfSecondOfExecution << "\n";

    delete executedFunction;
    delete array;
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 7500 элементами

```
154     //Selection sort for array with 7500 elements:
155     amountOfElementsOfArray = 7500;
156     valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
157     array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
158     delete[] valuesOfElementsOfArray;
159
160     arraySortingMethod = new SelectionArraySortingMethod<double>();
161     array->setArraySortingMethod(arraySortingMethod);
162
163     executedFunction = new std::function<void()>([array]()
164     {
165         array->sortElements();
166     });
167
168     amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
169
170     cout << "Time of execution of selection sorting of array with 7500 elements: "
171         << amountOfSecondOfExecution << "\n";
172
173     delete executedFunction;
174     delete array;
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 5000 элементами

```
176     //Selection sort for array with 5000 elements:
177     amountOfElementsOfArray = 5000;
178     valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
179     array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
180     delete[] valuesOfElementsOfArray;
181
182     arraySortingMethod = new SelectionArraySortingMethod<double>();
183     array->setArraySortingMethod(arraySortingMethod);
184
185     executedFunction = new std::function<void()>([array]()
186     {
187         array->sortElements();
188     });
189
190     amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
191
192     cout << "Time of execution of selection sorting of array with 5000 elements: "
193         << amountOfSecondOfExecution << "\n";
194
195     delete executedFunction;
196     delete array;
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 2500 элементами

```
198     //Selection sort for array with 2500 elements:
199     amountOfElementsOfArray = 2500;
200     valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
201     array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
202     delete[] valuesOfElementsOfArray;
203
204     arraySortingMethod = new SelectionArraySortingMethod<double>();
205     array->setArraySortingMethod(arraySortingMethod);
206
207     executedFunction = new std::function<void()>([array]()
208     {
209         array->sortElements();
210     });
211
212     amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
213
214     cout << "Time of execution of selection sorting of array with 2500 elements: "
215         << amountOfSecondOfExecution << "\n";
216
217     delete executedFunction;
218     delete array;
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для массива с 1000 элементами

```
220     //Selection sort for array with 1000 elements:
221     amountOfElementsOfArray = 1000;
222     valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
223     array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
224     delete[] valuesOfElementsOfArray;
225
226     arraySortingMethod = new SelectionArraySortingMethod<double>();
227     array->setArraySortingMethod(arraySortingMethod);
228
229     executedFunction = new std::function<void()>([array]()
230     {
231         array->sortElements();
232     });
233
234     amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
235
236     cout << "Time of execution of selection sorting of array with 1000 elements: "
237         << amountOfSecondOfExecution << "\n";
238
239     delete executedFunction;
240     delete array;
```

Полный проект: [vladzuev00/sorting-method-project-for-bsu-cpp (github.com)]