

Лабораторная работа 4

Исследование и оценка алгоритмов сортировки

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

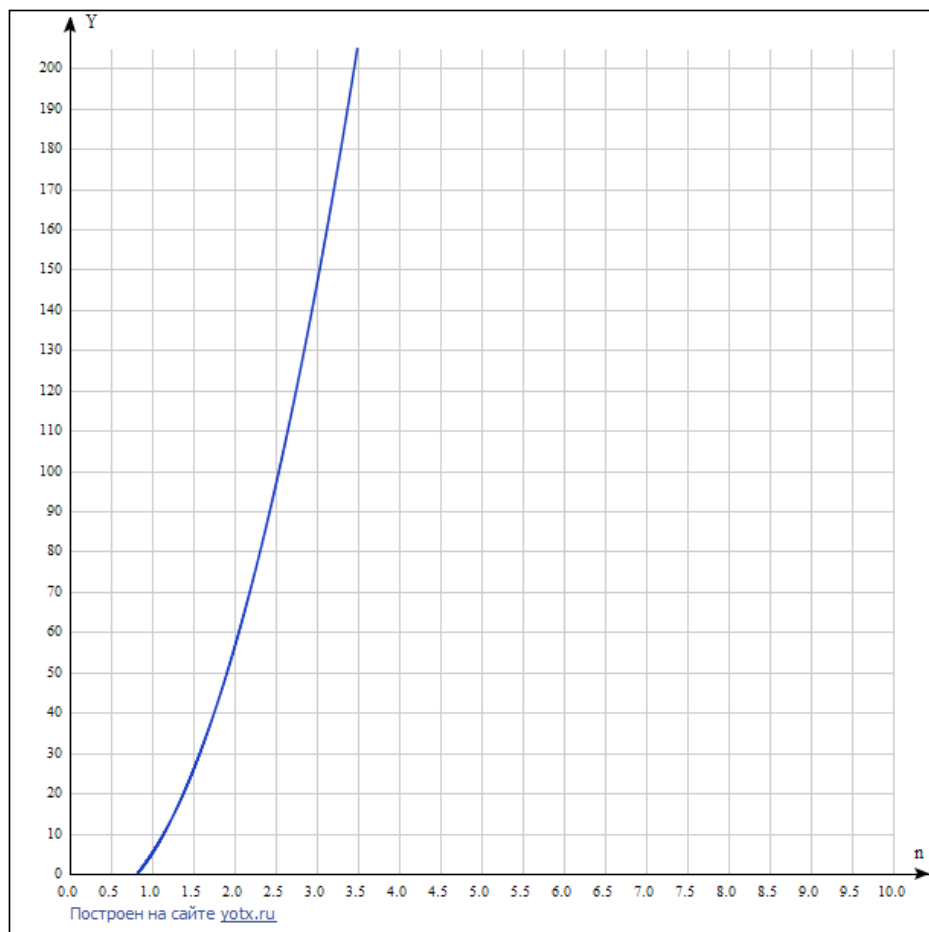
Асимптотическая сложность алгоритма

$$\begin{aligned} &1 + 1 + 1 + n \cdot 2 + (n - 1) \cdot 3 + (n - 1) \cdot 1 + [l = 1, l = n - 1] \sum (1 + (l + 1) \cdot 3 + l \cdot 3 + l \cdot (1 + \\ &2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 1 + 2 + 1 + \\ &1 + 1 + 1)) + (n - 1) \cdot 2 = 3 + 2 \cdot n + 3 \cdot n - 3 + n - 1 + [l = 1, l = n - 1] \sum (1 + 3 \cdot l + 3 + \\ &3 \cdot l + l \cdot 3) + 2 \cdot n - 2 = 8 \cdot n - 3 + [l = 1, l = n - 1] \sum (39 \cdot l + 4) = [S(N) = (a_1 + a_N) / 2 \cdot \\ &N, N = n - 1, a_1 = 39 \cdot 1 + 4 = 43, a_N = 39 \cdot (n - 1) + 4 = 39 \cdot n - 35] = 8 \cdot n - 3 \\ &+ (43 + 39 \cdot n - 35) / 2 \cdot (n - 1) = O(n^2) \end{aligned}$$

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 экспериментальной оценки временной сложности алгоритма для массива с 10000 элементами

```

21
22 //Quick bubble sort for array with 10000 elements:
23 int amountOfElementsOfArray = 10000;
24 const double *valuesOfElementsOfArray
25     = elementsGenerator->generateElements(amountOfElementsOfArray);
26 Array<double> *array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
27 delete[] valuesOfElementsOfArray;
28
29 ArraySortingMethod<double> *arraySortingMethod = new BubbleArraySortingMethod<double>();
30 array->setArraySortingMethod(arraySortingMethod);
31
32 std::function<void()> *executedFunction = new std::function<void()>([array]()
33 {
34     array->sortElements();
35 });
36
37 const MeasurerOfTimeOfExecution * const measurerOfTimeOfExecution = new MeasurerOfTimeOfExecution();
38 double amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
39
40 cout << "Time of execution of quick bubble sorting of array with 10000 elements: " << amountOfSecondOfExecution << "\n";
41
42 delete executedFunction;
43 delete array;

```

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

```

45 //Quick bubble sort for array with 7500 elements:
46 amountOfElementsOfArray = 7500;
47 valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
48 array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
49 delete[] valuesOfElementsOfArray;
50
51 arraySortingMethod = new BubbleArraySortingMethod<double>();
52 array->setArraySortingMethod(arraySortingMethod);
53
54 executedFunction = new std::function<void()>([array]()
55 {
56     array->sortElements();
57 });
58
59 amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
60
61 cout << "Time of execution of quick bubble sorting of array with 7500 elements: " << amountOfSecondOfExecution << "\n";
62
63 delete executedFunction;
64 delete array;

```

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

```

66 //Quick bubble sort for array with 5000 elements:
67 amountOfElementsOfArray = 5000;
68 valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
69 array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
70 delete[] valuesOfElementsOfArray;
71
72 arraySortingMethod = new BubbleArraySortingMethod<double>();
73 array->setArraySortingMethod(arraySortingMethod);
74
75 executedFunction = new std::function<void()>([array]()
76 {
77     array->sortElements();
78 });
79
80 amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
81
82 cout << "Time of execution of quick bubble sorting of array with 5000 elements: " << amountOfSecondOfExecution << "\n";
83
84 delete executedFunction;
85 delete array;
86

```

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

```

86 //Quick bubble sort for array with 2500 elements:
87 amountOfElementsOfArray = 2500;
88 valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
89 array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
90 delete[] valuesOfElementsOfArray;
91
92 arraySortingMethod = new BubbleArraySortingMethod<double>();
93 array->setArraySortingMethod(arraySortingMethod);
94
95 executedFunction = new std::function<void>([array]()
96 {
97     array->sortElements();
98 });
99
100 amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
101
102 cout << "Time of execution of quick bubble sorting of array with 2500 elements: " << amountOfSecondOfExecution << "\n";
103
104 delete executedFunction;
105 delete array;
106

```

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

```

108 //Quick bubble sort for array with 1000 elements:
109 amountOfElementsOfArray = 1000;
110 valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
111 array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
112 delete[] valuesOfElementsOfArray;
113
114 arraySortingMethod = new BubbleArraySortingMethod<double>();
115 array->setArraySortingMethod(arraySortingMethod);
116
117 executedFunction = new std::function<void>([array]()
118 {
119     array->sortElements();
120 });
121
122 amountOfSecondOfExecution = measurerOfTimeOfExecution->mesuareExecutionBySeconds(*executedFunction);
123
124 cout << "Time of execution of quick bubble sorting of array with 1000 elements: "
125 << amountOfSecondOfExecution << "\n";
126
127 delete executedFunction;
128 delete array;

```

Метод сортировки выбором

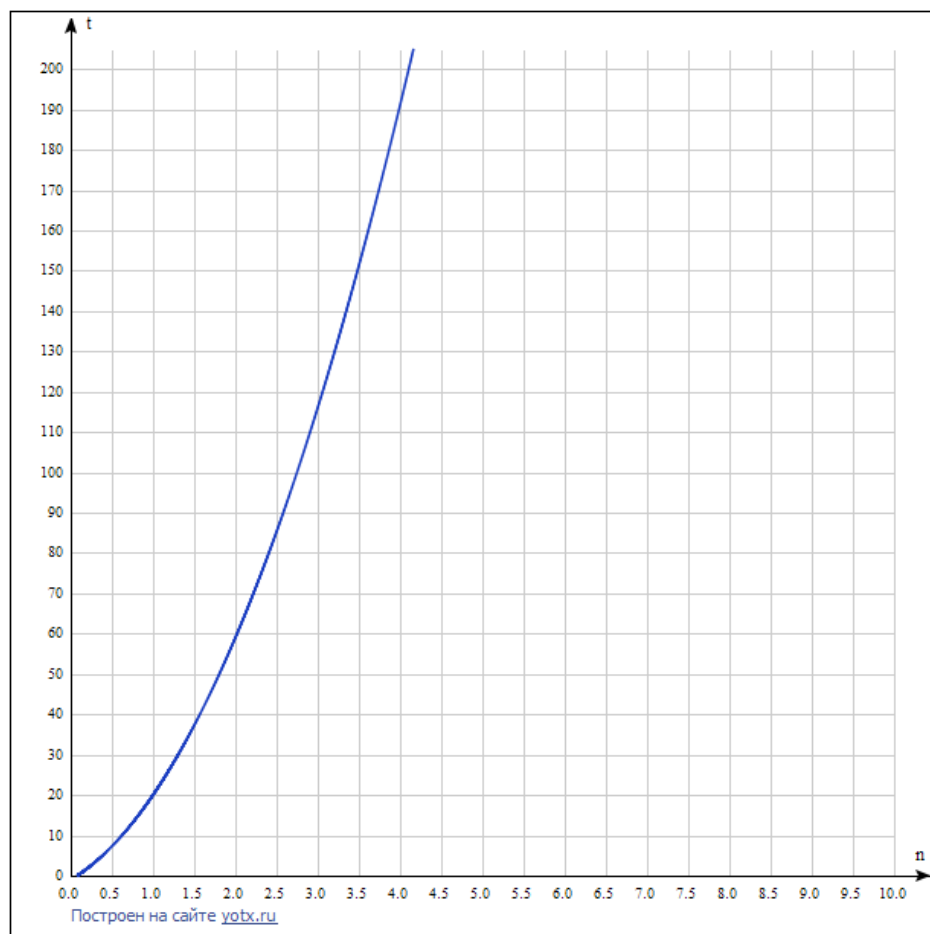
Асимптотическая сложность алгоритма

$$\begin{aligned}
 & 1 + 1 + 1 + n \cdot (1 + 1) + (n - 1) \cdot (1 + 1 + 1) + [l = 1, l = n - 1] \sum (1 + 1 + 1 + (1 + 1 + 1) \\
 & \cdot (l + 1) + l \cdot (1 + 1 + 1) + l \cdot (1 + 2 + 1 + 1 + 2 + 1 + 1 + 1 + 1 + 1)) + (1 + 1 + 1 \\
 & + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 1) \cdot (n - 1) \\
 & = 3 + 2 \cdot n + 3 \cdot (n - 1) + 22 \cdot (n - 1) + [l = 1, l = n - 1] \sum (3 + 3 \cdot (l + 1) + 3 \cdot l + 12 \cdot l) \\
 & = 3 + 2 \cdot n + 25 \cdot (n - 1) + [l = 1, l = n - 1] \sum (18 \cdot l + 6) \\
 & = [S(N) = (a_1 + a_N) / 2 \cdot N, a_1 = 18 \cdot 1 + 6 = 24, a_N = 18 \cdot (n - 1) + 6, N = n - 1] \\
 & = 3 + 2 \cdot n + 25 \cdot n - 25 + (24 + 18 \cdot (n - 1) + 6) / 2 \cdot (n - 1) \\
 & = 27 \cdot n - 22 + 15 + (9 \cdot n - 6) \cdot (n - 1) = O(n^2)
 \end{aligned}$$

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

```

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

```

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

```

132 //Selection sort for array with 10000 elements:
133 amountOfElementsOfArray = 10000;
134 valuesOfElementsOfArray = elementsGenerator->generateElements(amountOfElementsOfArray);
135 array = new Array<double>(valuesOfElementsOfArray, amountOfElementsOfArray);
136 delete[] valuesOfElementsOfArray;
137
138 arraySortingMethod = new SelectionArraySortingMethod<double>();
139 array->setArraySortingMethod(arraySortingMethod);
140
141 executedFunction = new std::function<void>()>([array]()
142 {
143     array->sortElements();
144 });
145
146 amountOfSecondOfExecution = measurerOfTimeOfExecution->meswareExecutionBySeconds(*executedFunction);
147
148 cout << "Time of execution of selection sorting of array with 10000 elements: "
149      << amountOfSecondOfExecution << "\n";
150
151 delete executedFunction;
152 delete array;

```

Исходный код программы экспериментальной 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->mesureExecutionBySeconds(*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;
175

```

Исходный код программы экспериментальной 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->mesureExecutionBySeconds(*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;
197

```

Исходный код программы экспериментальной 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->mesureExecutionBySeconds(*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;
219

```

Исходный код программы экспериментальной 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 \(github.com\)](https://github.com/vladzuev00/sorting-method-project-for-bsu-cpp)