Государственное образовательное учреждение
высшего профессионального образования
«Московский Государственный Технический Университет
имени Н.Э. Баумана»

Отчет
По лабораторной работе №3
По курсу «Анализ Алгоритмов»
На тему «Исследование сложности алгоритмов сортировки»

Кизилов Дмитрий, ИУ7-54

Москва, 2017

Оглавление

Постановка задачи

Реализовать алгоритмы сортировки массивов, найти сложность этих алгоритмов и провести временные эксперименты

Блок-схемы

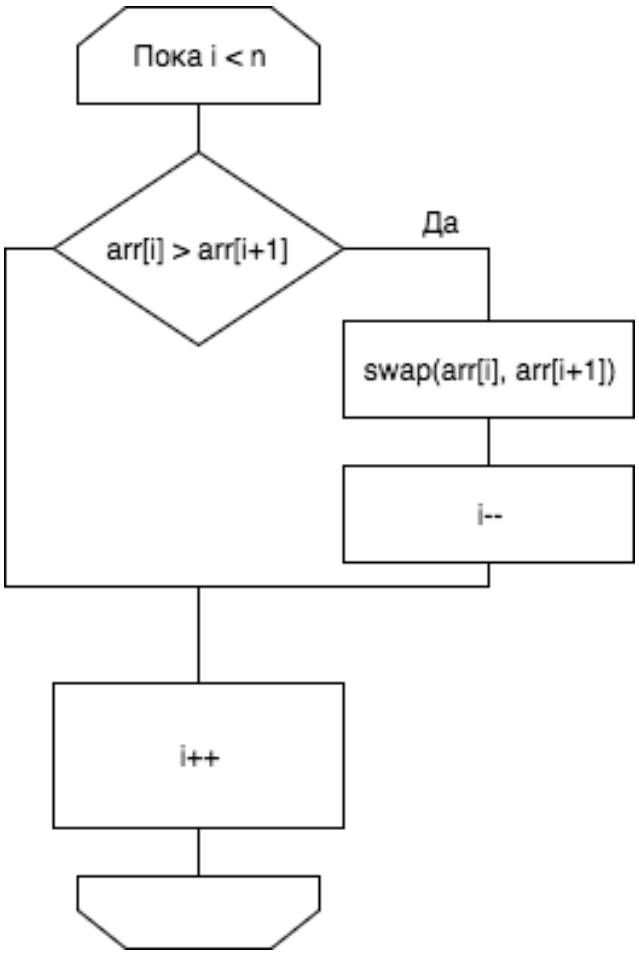

Рис. 1: Блок-схема Гномьего алгоритма

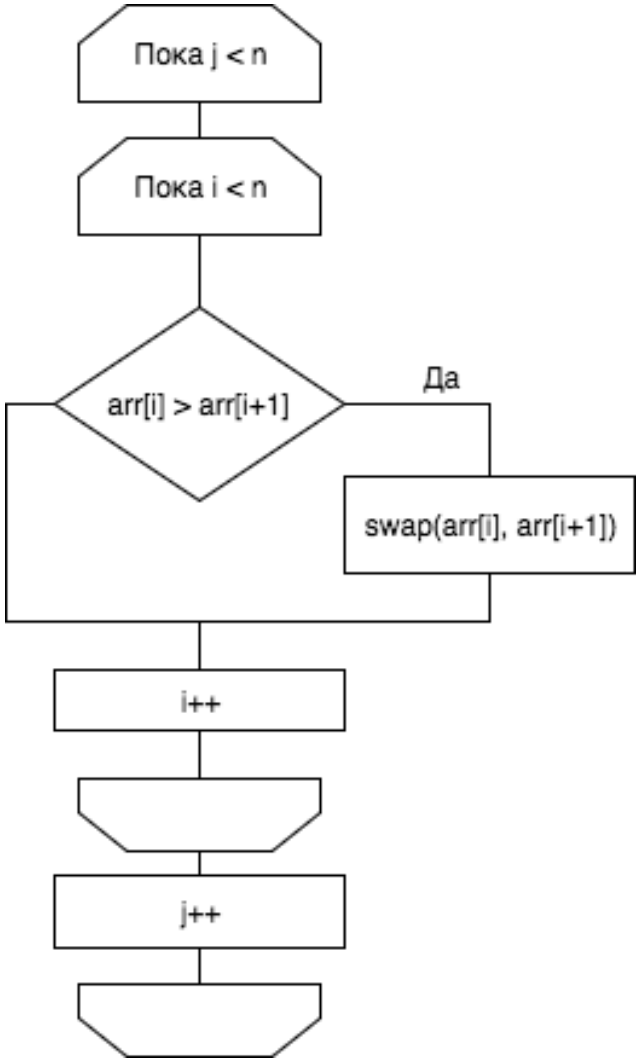

Рис. 2: Блок-схема алгоритма Пузырька

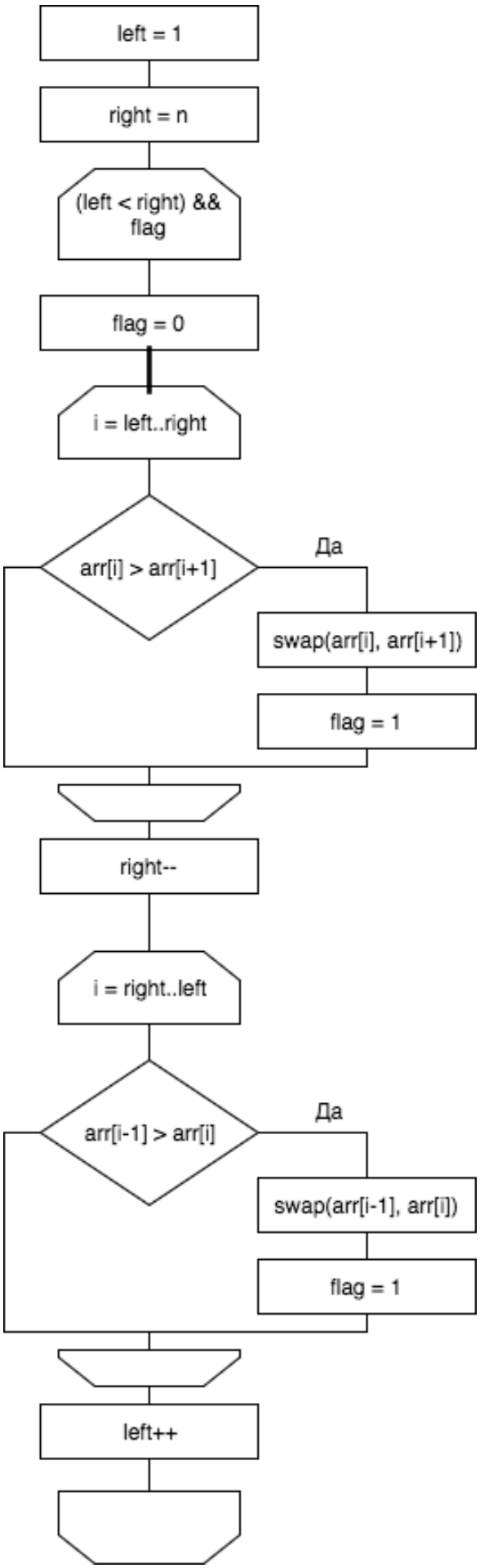

Рис. 3: Блок-схема алгоритма Шейкера

Листинг

MAIN.CPP:

```
1  #include <iostream>
2  #include <fstream>
3  #include "sorts.cpp"
4
5  #define ST_NUM 100            // std == 100
6  #define REP_NUM 10            // std == 10
7  #define TIME_REP_NUM 10       // std == 10
8  #define D_NUM 100         // std == 100
9
10
11 using namespace std;
12
13 template <typename T>
14 int SIGN(T& in)
15 {
16     return (in >= 0) ? (1) : (-1);
17 }
18
19 template <typename T>
20 void timeCnt(T st_num, T d_num, int rep_num, BaseSorter<T>* srtr,
       ofstream& file)
21 {
22     vector<T> arr;
23
24     for (int i = 0; i < rep_num; i++)
25     {
26         long double time;
27         for (int j = 0; j < TIME_REP_NUM; j++)
28         {
29             arr.clear();
30             srtr->fill(arr, (st_num + d_num*i) * SIGN(d_num));
31
32             time += (srtr->sort(arr)/CLOCKS_PER_SEC)*1000;
33         }
34         time /= TIME_REP_NUM;
35
36         cout.precision(4);
37         cout.setf(ios::fixed);
38         cout << st_num + d_num*i << " : " << time << endl;
39
40         file.precision(4);
41         file.setf(ios::fixed);
42         file << time << "\n";
43     }
44
45     file << "\n";
46     cout << "\n";
```

```
47
48      delete srtr;
49  }
50
51  int main(void)
52  {
53      srand(time(0));
54      ofstream bub("bub.txt");
55      ofstream shk("skr.txt");
56      ofstream gnm("gnome.txt");
57
58      timeCnt<double>(ST_NUM, D_NUM, REP_NUM, new BubbleSorter<double
            >(new RandFiller<double>()), bub);
59      timeCnt<double>(ST_NUM, D_NUM, REP_NUM, new BubbleSorter<double
            >(new UpFiller<double>()), bub);
60      timeCnt<double>(ST_NUM*10, -D_NUM, REP_NUM, new BubbleSorter<
            double>(new UpFiller<double>()), bub);
61
62      timeCnt<double>(ST_NUM, D_NUM, REP_NUM, new ShakerSorter<double
            >(new RandFiller<double>()), shk);
63      timeCnt<double>(ST_NUM, D_NUM, REP_NUM, new ShakerSorter<double
            >(new UpFiller<double>()), shk);
64      timeCnt<double>(ST_NUM*10, -D_NUM, REP_NUM, new ShakerSorter<
            double>(new UpFiller<double>()), shk);
65
66      timeCnt<double>(ST_NUM, D_NUM, REP_NUM, new GnomeSorter<double>(
            new RandFiller<double>()), gnm);
67      timeCnt<double>(ST_NUM, D_NUM, REP_NUM, new GnomeSorter<double>(
            new UpFiller<double>()), gnm);
68      timeCnt<double>(ST_NUM*10, -D_NUM, REP_NUM, new GnomeSorter<
            double>(new UpFiller<double>()), gnm);
69
70      bub.close();
71      shk.close();
72      gnm.close();
73      return 0;
74  }
```

SORTS.CPP:

```
1  #include "sorts.h"
2  #include <iostream>
3  #include <stdio.h>
4
5  template <typename T>
6  void RandFiller<T>::fill(vector<T> &arr, T cnt)
7  {
8      int Cnt = int(cnt);
9
10     if (arr.empty() || (arr.size() != Cnt))
11     {
```

```
12          arr.resize(Cnt, 0);
13      }
14
15      for (int i = 0; i < Cnt; i++)
16      {
17          arr[i] = rand();
18      }
19  }
20
21  template <typename T>
22  void UpFiller<T>::fill(vector<T> &arr, T cnt)
23  {
24      T st = 0;
25      int d = 1;
26      if (cnt < 0)
27      {
28          cnt *= -1;
29          st = cnt;
30          d = -1;
31      }
32      int Cnt = int(cnt);
33
34      if (arr.empty() || (arr.size() != Cnt))
35      {
36          arr.resize(Cnt, 0);
37      }
38
39      for (int i = 0; i < Cnt; i++)
40      {
41          arr[i] = st+i*d;
42      }
43  }
44
45  template <typename T>
46  BaseSorter<T>::BaseSorter(Filler<T> *in)
47  {
48      fl = in;
49  }
50
51
52  template <typename T>
53  long double GnomeSorter<T>::sort(vector<T> &arr)
54  {
55      int i = 0;
56      long double start = clock();
57
58      while (i < arr.size())
59      {
60          if (i == 0 || arr[i - 1] <= arr[i])
61          {
```

```
62              ++i;
63          }
64          else
65          {
66              std::swap(arr[i - 1], arr[i]);
67              --i;
68          }
69      }
70
71      return clock() - start;
72  }
73
74  template <typename T>
75  long double BubbleSorter<T>::sort(vector<T> &arr)
76  {
77      long double start = clock();
78
79      for (int i = 0; i < arr.size(); i++)
80      {
81          for (int j = i; j < arr.size(); j++)
82          {
83              if (arr[i] > arr[j])
84              {
85                  std::swap(arr[i], arr[j]);
86              }
87          }
88      }
89
90      return clock() - start;
91  }
92
93  template <typename T>
94  long double ShakerSorter<T>::sort(vector<T> &arr)
95  {
96      long double start = clock();
97
98      int left = 0;
99      int right = arr.size() - 1;
100     int flag = 1;
101     while ((left < right) && flag)
102     {
103         flag = 0;
104         for (int i = left; i < right; i++)
105         {
106             if (arr[i] > arr[i+1])
107             {
108               std::swap(arr[i], arr[i+1]);
109               flag = 1;
110             }
111         }
```

```
112
113             right--;
114
115             for (int i = right; i > left; i--)
116             {
117                 if (arr[i-1] > arr[i])
118                 {
119                     std::swap(arr[i], arr[i-1]);
120                     flag = 1;
121                 }
122             }
123
124             left++;
125         }
126
127         return clock() - start;
128     }
```

Временные эксперименты

Измерения производились для вещественных массивов

Размер массива	Отсортированный	Отсортированный в обратном порядке	Случайный
100	0,1030	0,9690	0,2486
200	0,3601	0,0993	0,8915
300	0,8249	0,0132	1,9465
400	1,4108	0,0056	3,2779
500	2,3467	0,0065	5,0020
600	3,3259	0,0075	7,2402
700	4,6154	0,0094	9,6110
800	5,9703	0,0103	12,7577
900	7,6857	0,0111	15,8312
1000	9,6741	0,0123	17,2874

Замеры времени в мс (среднее из 10 замеров) для Гномьей сортировки

Размер массива	Отсортированный	Отсортированный в обратном порядке	Случайный
100	0,0340	0,0637	0,0893
200	0,1094	0,2910	0,2853
300	0,4105	0,6484	0,6150
400	0,5323	0,9075	1,1760
500	0,7580	1,3810	1,8694
600	1,0835	2,5466	2,6311
700	1,5378	3,4827	4,1932
800	2,1313	4,8444	5,0369
900	2,4246	5,4381	6,1783
1000	3,7684	6,0300	7,3179

Замеры времени в мс (среднее из 10 замеров) для Пузырьковой сортировки

Размер массива	Отсортированный	Отсортированный в обратном порядке	Случайный
100	0,0014	0,0551	0,0941
200	0,0024	0,1715	0,2969
300	0,0037	0,3713	0,6259
400	0,0055	0,6387	1,2558
500	0,0051	1,0345	1,6751
600	0,0060	1,4242	2,4253
700	0,0065	2,4775	3,1636
800	0,0075	3,8840	4,0739
900	0,0083	3,7626	6,2941
1000	0,0121	4,0406	5,9026

Замеры времени в мс (среднее из 10 замеров) для сортировки Шейкером

Рис. 4: Сравнение времени при нечетном кол-ве элементов

Модель вычислений

Операнды +, -, *, /, <, >, <=, >=, ==, !=, [ ] имеют значение f = 1;
Значение для цикла f = 2 + N(...), где N - количество итераций.

Рассчет сложности алгоритмов

Гномий алгоритм:

1. Лучший случай(отсортированный массив) : $\sim O(n)$
2. Худший случай(отсортированный обратно) : $\sim O(n^2)$

Пузырьковый алгоритм:

1. Лучший случай(отсортированный массив) :
   $2 + n * (2 + 2 + n * (2 + 3)) = 5 * n^2 + 4 * n + 2 \sim O(n^2)$
2. Худший случай(отсортированный обратно) :
   $2 + n * (2 + 2 + n * (2 + 10)) = 12 * n^2 + 4 * n + 2 \sim O(n^2)$

Алгоритм шейкера:

1. Лучший случай(отсортированный массив) : $\sim O(n)$
2. Худший случай(отсортированный обратно) : $\sim O(n^2)$

Выводы

В результате проведенных испытаний алгоритмов было установлено, что:

1. Алгоритм Шейкера значительно быстрее пузырька и гномьей сортировки, особенно на отсортированном наборе данных.

2. Гномий алгоритм самый базовый и медленный из рассмотренных, однако он имеет линейную сложность при наилучшем случае

3. У пузырька и в худшем, и в лучшем случае квадратичная сложность, что делает его "усредненным"алгоритмом по времени и сложности среди приведенных

Заключение

В ходе лабораторной работы были реализованы 3 алгоритма сортировок : гномий, пузырьковый и шейкер. Были получены навыки работы с контейнером std::vector в C++, Работой с паттернами проектирования и ООП, а так же с LaTeX. Изучен подход к вычислению сложности алгоритмов.