МИНИСТЕРСТВО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
«МАТИ» – РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. К. Э. ЦИОЛКОВСКОГО

Отчет по курсовой работе

"ИССЛЕДОВАНИЕ АЛГОРИТМОВ СОРТИРОВКИ В СРЕДЕ ОС LINUX"

Преподаватель:	Лидовский В.В.
Студент:	Нагорный А.А.
Группа:	14ИВТ-2ДБ-005
Вариант:	14

February 24, 2013 LaTeX

Contents

1 Цель исследования

Изучить различные алгоритмы сортировки, провести исследования времени их выполнения, обозначить их как положительные, так и отрицательные стороны, научиться умело выбирать определенный алгоритм для конкретной задачи. Получить навык программирования на языке Pascal, используя среды разработки и рабочее окружение ОС Linux, набраться опыта в составлении технических отчетов с помощью LaTeXс использованием GNUplot для построения графиков.

2 Краткая характеристика исследуемых алгоритмов сортировки

2.1 Сортировка бинарным деревом

Бинарным деревом называют упорядоченную структуру данных, в которой каждому элементу поставлены в соответсвие до трех других: левый и правый ребенок (преемник) и родитель (предшественник). Левый ребенок должен быть больше, а правый – меньше или равен родителю. Единственный элемент, не имеющий родителя, называется корнем дерева.

Если по исходной последовательности данных построить бинарное дерево, а затем вывести его элементы по правилам обхода дерева, то полученная последовательность окажется отсортируемой.

Правила обхода дерева:

- Обход начинается с корня, предыдущим элементом считается верхний.
- Если предыдущий элемент – верхний, то если левый преемник существует, то совершить переход к этому элементу, иначе вывод текущего элемента, и если правый преемник существует, то переход к нему, в противном случае - переход к предшественнику.
- Если предыдущий элемент – левый, то вывод текущего элемента, и если правый преемник существует, то переход к правому преемнику, в противном случае - переход к предшественнику
- Если предыдущий элемент – правый, то переход к предшественнику.
- Обход заканчивается после вывода последнего элемента по счетчику.

2.2 Метод Хоара ("Быстрая сортировка")

Суть данного метода заключается в нахождении такого элемента сортируемой последовательности, который бы делил последовательность на две части так, что слева от него находились бы элементы не меньшие его, а справа - не большие. Поиск можно организовать разными способыми, например:

Установим два индекса на 1-й (индекс i) и на последний (индекс j) элемент последовательности. Затем, пока элемент с индексом $i \leq$ элементу с индексом j, будем декрементировать i (т.е. уменьшать на 1). Если же i-й элемент больше j-го, то их меняем местами. Затем, пока j-й элемент $\leq i$-го, будем инкрементировать i (увеличивать на 1). Если же j-й $\geq i$-го, то менямем местами i и j. Этот процесс продолжаем до тех пор, пока $i \neq j$, и элемент с индексом $i = j$ - искомый.

Далее, ищем такой же элемент для 2х полученных в результате разбиений последовательностей, и продолжаем процесс рекурсивно с вновь полученными разбиениями. Разбиения, содержащие 1 или 2 элемента, являются конечными и далее не делятся.

3 Листинг исследовательской программы

```
1  program KursWork_Nagorny;
2  (*объявление констант*)
3  const
4      maxnumber=70000;
5  (*Объявление пользовательских типов*)
6  type
7      arr= array of longint;
8
9      menu1=(BINTREE,QSORT);
10     menu2=(t250,t500,t1000,t2000,t4000,t8000,t16000);
11     menu3=(SORTED,ASORTED,CHILD,RAND);
12
13     tdirs =(left,right,up);
14
15     table=record
16         size:longint;
17         Time: array[menu3] of extended;
18     end;
19
20     TableType=array [menu2] of table;
21
22     rectree=record
23         data:longint;
24         dirs: array [tdirs] of word
25     end;
26
27 (*Подключение библиотеки, для работы с сис. таймером*)
28
29 {$L timer.o}
30 Procedure init_timer; cdecl; external;
31 Function get_timer: longint; cdecl; external;
32 {$LinkLib c}
33
34 (*Проверка на сортировку*)
35 function TestSort(a:arr):boolean;
36 var
37     i:longint;
38     sorttest:boolean;
39 begin
40     sorttest:=true;
41     for i:=0 to high(a)-1 do
42         if a[i] < a[i+1] then begin
43         sorttest:=false;
44         break;
45     end;
46     TestSort:=sorttest;
47
48 end;
49
50 (*Копирование массива*)
51 procedure CopyArr(a,b:arr);
52 var
53     i:longint;
54 begin
55     for i:=low(a) to high(a) do
56         b[i]:=a[i];
57     i:=random(high(a)-1)+1;
58     if a[i]<> b[i] then writeln('ERR_COPY');
59
60     end;
61
62 (*процедура смены x2 элтов- массива*)
63 procedure exch(var ar:arr;a,b:longint);
64 var
65     tmp:longint;
66 begin
67     tmp:=ar[a];
68     ar[a]:=ar[b];
69     ar[b]:=tmp;
70 end;
71
72 (*Быстрая сортировка, метод Хоара*)
73 procedure SortQuick(var a:arr;l,u:longint);
74 var
75     i,j:longint;
76 begin;
```

```
77      i:=l;
78       j:=u;
79      repeat
80          while i<>j do begin
81              if a[i]>=a[j] then
82                  dec(j)
83              else begin
84                  exch(a,i,j);
85                  break
86              end;
87          end;
88          while i<>j do begin
89              if a[i]>=a[j] then
90                  inc(i)
91              else begin
92                  exch(a,i,j);
93                  break
94              end;
95          end;
96      until i=j;
97      if i-1>l then
98                      SortQuick(a,l,i-1);
99      if j+1<u then
100                     SortQuick(a,j+1,u);
101 end;
102
103 (*Сортировка бинарным деревом*)
104 procedure SortTree(a:arr);
105 var
106     tree:array of rectree;
107     dir:tdirs;
108     i,j,k:longint;
109 begin
110     setlength(tree,high(a)+1);
111     for i:= 1 to high(a)+1 do
112         for dir :=left to right do
113             tree[i].dirs[dir]:=0;
114     tree[1].data:=a[1-1];
115     tree[1].dirs[up]:=1;
116     for i:=2 to high(a)+1 do begin
117         j:=1;
118         repeat
119             k:=j;
120             if tree[j].data<a[i-1] then
121                 dir:=left
122             else
123                 dir:=right;
124             j:= tree[j].dirs[dir]
125         until j=0;
126         tree[i].data:=a[i-1];
127         tree[i].dirs[up]:=k;
128         tree[k].dirs[dir]:=i;
129     end;
130     dir:= up;
131     i:=1;
132     j:=1;
133     repeat
134         case dir of
135             up:begin
136                 while tree[j].dirs[left] <>0 do
137                     j:=tree[j].dirs[left];
138                 a[i-1]:=tree[j].data;
139                 inc(i);
140                 if tree[j].dirs[right]<>0 then
141                     j:=tree[j].dirs[right]
142                 else begin
143                     if tree[tree[j].dirs[up]].dirs[left]=j then
144                         dir:=left
145                     else
146                         dir:=right;
147                     j:=tree[j].dirs[up]
148                 end
149             end;
150             left:begin
151                 a[i-1]:=tree[j].data;
152                 inc(i);
```

```
153                     if tree[j].dirs[right] = 0 then begin
154                         if tree[tree[j].dirs[up]].dirs[left]<>j then
155                             dir :=right;
156                         j:= tree[j].dirs[up];
157                     end else begin
158                         j:= tree[j].dirs[right];
159                         dir:=up
160                     end
161                 end;
162                 right:begin
163                     if tree[tree[j].dirs[up]].dirs[left]=j then
164                         dir:=left;
165                     j:=tree[j].dirs[up];
166                     end
167                 end;
168         until i> high(a)+1;
169     end;
170
171     (*Процедура печати таблицы*)
172     procedure print(a:TableType;b:TableType );
173     var
174         i:menu2;
175     begin
176         writeln();
177         writeln('':15, 'Сортировка бинарным деревом / Быстрая сортировка мс()');
178         write('Размер' :15,'        Упорядоченные':15, '        Обратный порядок        ');
179         writeln('Вырожденные            Случайные');
180         for i:=t250 to t16000 do begin
181             write(a[i].size :7,'    ', a[i].Time[SORTED]:10:4 ,' / ', b[i].Time[SORTED]:5:4,'    ');
182             write( a[i].Time[ASORTED]:10:4 ,' / ', b[i].Time[ASORTED]:5:4,'    ');
183             write( a[i].Time[CHILD]:10:4 ,' / ', b[i].Time[CHILD]:5:4,'    ');
184             writeln( a[i].Time[RAND]:10:4 ,' / ', b[i].Time[RAND]:5:4,'    ');
185
186
187
188         end;
189     end;
190
191     (*Главная часть программы*)
192     var
193         mas1,mas2:arr;
194         i,max :longint;
195         t2:menu2;
196         t3:menu3;
197         TableBinTreeSort:TableType;
198         TableQSort: TableType;
199     begin
200         randomize;
201         for t3:=SORTED to RAND do
202              for t2:=t250 to t16000 do begin
203                 case t2 of
204                     t250:max:=250;
205                     t500: max:=500;
206                     t1000: max:=1000;
207                     t2000: max:=2000;
208                     t4000: max:=4000;
209                     t8000: max:=8000;
210                     t16000: max:=16000;
211                 end;
212
213                 setlength(mas1,max);
214                 setlength(mas2,max);
215                 TableBinTreeSort[t2].size:=max;
216                 TableQSort[t2].size:=max;
217                 dec(max);
218                 case t3 of
219                     SORTED: begin
220                         mas1[0]:=maxnumber;
221                         for i:=1 to high(mas1) do
222                             mas1[i]:=mas1[i-1]-random(maxnumber div (max+1))-1;
223                     end;
224                     ASORTED: begin
225                         mas1[0]:=1;
226                         for i:=1 to high(mas1) do
227                             mas1[i]:=mas1[i-1]+random(maxnumber div (max+1))-1;
228                     end;
```

```
229                     CHILD: for i:=0 to high(mas1) do
230                         mas1[i]:=random(12)+1;
231                     RAND: for i:=low(mas1) to high(mas1) do
232                         mas1[i]:=random(maxnumber)+1;
233                 end;
234
235                 CopyArr(mas1,mas2);
236
237                 init_timer();
238                     SortTree(mas1);
239                     TableBinTreeSort[t2].Time[t3]:=get_timer()/1000;
240                     if(testsort(mas1)=false) then begin
241                         writeln ('Ошибка: массив не был отсортирован');
242                         exit;
243                     end;
244                 CopyArr(mas2,mas1);
245                     init_timer();
246                     SortQuick(mas1,0,high(mas1)+1);
247                     TableQSort[t2].Time[t3]:=get_timer()/1000;
248                     if(testsort(mas1)=false) then begin
249                         writeln ('Ошибка: массив не был отсортирован');
250                         exit;
251                     end;
252             end;
253
254         print(TableBinTreeSort,TableQSort);
255     end.
```

Результат выполнения программы:

Сортировка бинарным деревом / Быстрая сортировка мс()

Размер	Упорядоченные	Обратный порядок	Вырожденные	Случайные
250	0.3510 / 0.1830	0.2670 / 0.1250	0.0440 / 0.0240	0.0380 / 0.0270
500	1.3500 / 0.7170	1.0500 / 0.4880	0.1410 / 0.0740	0.0820 / 0.0600
1000	4.9090 / 1.9660	4.0690 / 1.9160	0.4530 / 0.2090	0.1870 / 0.1290
2000	16.2800 / 7.4920	15.8900 / 7.3480	1.6460 / 0.6820	0.4210 / 0.2910
4000	63.6210 / 29.8030	60.6020 / 27.9920	5.9130 / 2.4730	0.8990 / 0.6400
8000	247.5150 / 119.1000	197.1270 / 96.9020	25.2400 / 9.6470	1.9190 / 1.3750
16000	987.0480 / 481.5840	409.8360 / 211.9910	108.1150 / 36.6040	4.6170 / 3.0400

4 Описание исследовательской программы

4.1 Модули

timer.o – объектный файл, содержащий в себе процедуры для работы с системным таймером: **init_timer()** и **get_timer()**.

4.2 Константы

maxnumber = 70000. Применяется в подготовке исходных данных (при заполнении массива). Обозначает максимально возможное значение элемента массива;

4.3 Типы

arr. Массив целых чисел. Этим типом определяется исходный массив;

menu1. Тип, определяемый переменную перечисляемого типа (BINTREE,QSORT). Используется при выборе алгоритма сортировки исходного массива;

menu2. Тип, определяемый переменную перечисляемого типа (t250,t500,t1000,t4000,t8000,t16000). Используется при выборе размера исходного массива;

menu3. Тип, определяемый переменную перечисляемого типа (SORTED,ASORTED,CHILD,RAND). Используется при выборе типа заполнения исходного массива (упорядоченный, обратный порядок, вырожденный, случайный) ;

tdirs. Тип, определяемый переменную перечисляемого типа (left, right, up). Применяется в алгоритме сортировки методом бинарного дерева;

rectree. Тип, определяемый как запись из: **data** - числа целого типа, обозначаемого значение листа, и **dirs** - массива, типа tdirs , обозначаемого индексы узлов-соседей в алгоритме сортировки бинарным деревом.

table. Тип, определяемый как запись из: **size** - числа целого типа, обозначаемого текущий размер исходного массива , и **Time** - вещественного числа, обозначающего время сортировки в миллисекундах (мс).

4.4 Подпрограммы

4.4.1 TestSort(a:arr)

Функция проверки переданного ей массива a на отсотированность: если все элементы с большими индексами больше элементов с меньшими индексами, то $sorttest := true$ иначе, $sorttest := false$. Возвращает значения, относительно переменной $sorttest$.

4.4.2 CopyArr(a,b:arr)

Процедура копирования, переданного ей массива a в переданный массив b : в цикле каждый элемент a присваивается элементу массива b с соответствующим индексом. Затем выполняется проверка на равность элементов массивов a и b, со случайным индексом, максимальное значение которого не превышает размерность aи b, и, в случае ошибки, выводится сообщение об ошибке копирования.

4.4.3 exch(var ar:arr;a,b:longint)

Процедура меняет значения элементов с индексами a и b массива arr, используя временную переменную для обмена - tmp типа longint.

4.4.4 SortQuick(var ar:arr;l,u:longint)

Процедура, выполняющая быструю сортировку по алгоритму Хоара. Процедура принимает следущие параметры: **a** - массив, в котором проводится сортитровка, **l** - первый элемент интервала сортировки, **u** - количество элементов, участвующих в сортировке. Так как алгоритм подразумевает рекурсию, переменные l и u не могут быть заданы в теле процедуры явным образом. В первоначальном вызове процедуры $l = 0$, а $u = high(ar) + 1$;

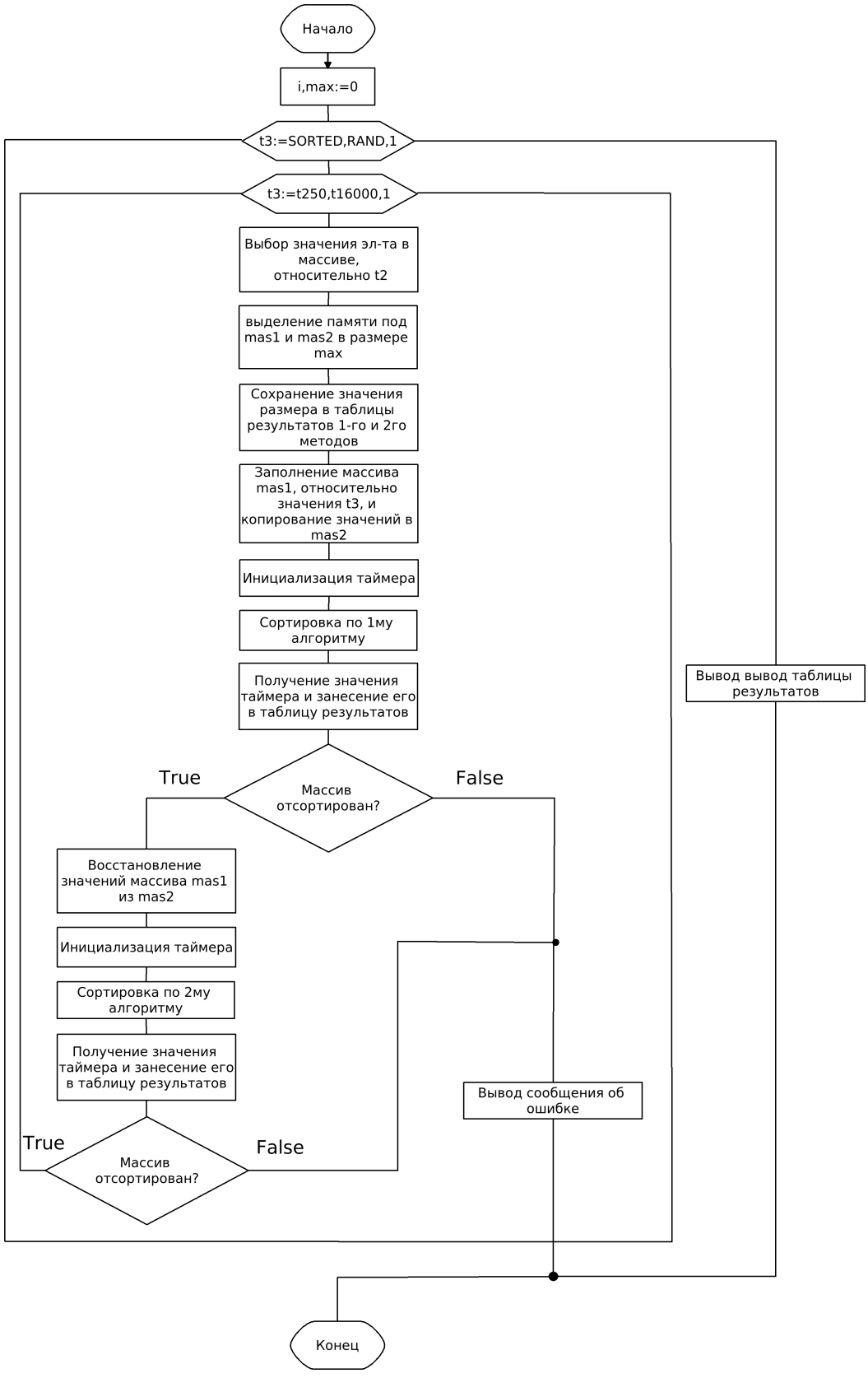
4.4.5 SortTree(a:arr)

Процедура, выполняющая сортировку методом бинарного дерева. Здесь в массиве **tree** из $high(a)+1$ элементов, каждый из которых - запись типа $rectree$, выстраивается дерево из элементов исходного массива **a** по определенному правилу: левый преемник должен быть больше, а правый – меньше или равен предшественнику. Расположение каждого элемента определяет переменная **dir** типа $dirs$. Затем, совершается обход дерева и заполнение массива a уже отсортированными данными.

4.4.6 print(a:TableType;b:TableType)

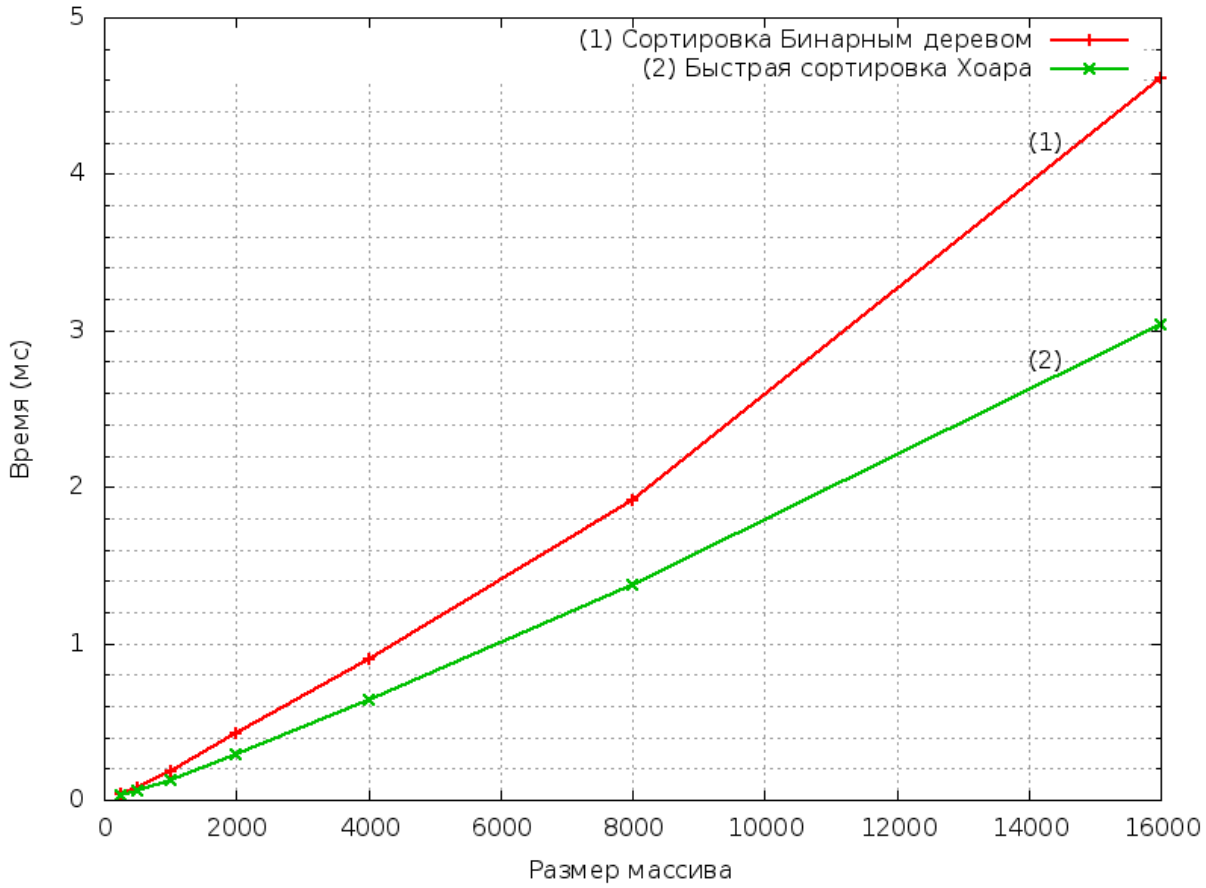
Процедура, выполняющая печать заранее приготовленных таблиц с результатами: a содержит результаты сортировки бинарным деревом, а b результаты быстрой сортировки Хоара.

4.5 Блок-схема программы



5 Таблица и график результатов исследования

Сортировка бинарным деревом / Быстрая сортировка (мс)				
Размер	Упорядоченные	Обратный порядок	Вырожденные	Случайные
250	0.3510 / 0.1830	0.2670 / 0.1250	0.0440 / 0.0240	0.0380 / 0.0270
500	1.3500 / 0.7170	1.0500 / 0.4880	0.1410 / 0.0740	0.0820 / 0.0600
1000	4.9090 / 1.9660	4.0690 / 1.9160	0.4530 / 0.2090	0.1870 / 0.1290
2000	16.2800 / 7.4920	15.8900 / 7.3480	1.6460 / 0.6820	0.4210 / 0.2910
4000	63.6210 / 29.8030	60.6020 / 27.9920	5.9130 / 2.4730	0.8990 / 0.6400
8000	247.5150 / 119.1000	197.1270 / 96.9020	25.2400 / 9.6470	1.9190 / 1.3750
16000	987.0480 / 481.5840	409.8360 / 211.9910	108.1150 / 36.6040	4.6170 / 3.0400
N	$\sim N^2$ / $N \ln^2 N$	$N \ln^2 N$ / $N \ln^2 N$	$N \ln(N)$ / N	N/ N



6 Анализ результатов эксперимента

Анализируя приведенные выше результаты, можно прийти к выводу, что **сортировка бинарным деревом** и **быстрая сортировка** выполняются практически за равное время для последовательностей случайных чисел одной длины. Тем не менее, уже при количестве 16000 элементов быстрая сортировка выполняется на 1.6 мсек. (0,0016 сек) быстрее, что уже является поводом для ее выбора. Кроме того, она проще, чем сортировка деревом, как с точки зрения понимания, так и реализации. Так же она не использует много памяти, не храня структуру размером N. Однако, рекурсивный подход может стать причиной ошибок во время реализации.

7 Вывод

При таком, сравнительно небольшом, значении N оба метода показали себя как быстрые алгоритмы сортировки. Эти методы довольно схожи, но обладают разной сложностью реализации.

Быстрая сортировка показалась мне более удобным и оптимальным алгоритмом для сортировки числовых последовательностей. Скорость сортировки и простота реализации предоставляют возможность использовать данный алгоритм даже в крупных проектах. Помимо этого, результат выполнения можно улучшить добавив в алгоритм рандомизированный выбор опорного элемента, что сократит время выполнения алгоритма даже для больших последовательностей.