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

March 2, 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.

В данной работе реализован рекурсивный метод обхода дерева.

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     tdirs =(left,right,up);
13
14     table=record
15         size:longint;
16         Time: array[menu3] of extended;
17     end;
18
19     TableType=array [menu2] of table;
20
21     PTreeType=^rectree;
22
23      rectree=record
24         data:longint;
25         left, right,up:PTreeType;
26     end;
27 (*Объявление глобальных переменных*)
28 var
29    IndPrintTree :longint;
30
31 (*Подключение библиотеки, для работы с сис. таймером*)
32
33 {$L timer.o}
34 Procedure init_timer; cdecl; external;
35 Function get_timer: longint; cdecl; external;
36 {$LinkLib c}
37
38 (*Проверка на сортировку*)
39 function TestSort(a:arr):boolean;
40 var
41     i:longint;
42     sorttest:boolean;
43 begin
44     sorttest:=true;
45     for i:=0 to high(a)-1 do
46         if a[i] < a[i+1] then begin
47         sorttest:=false;
48         break;
49     end;
50     TestSort:=sorttest;
51
52 end;
53
54 (*Копирование массива*)
55 procedure CopyArr(a,b:arr);
56 var
57     i:longint;
58 begin
59     for i:=low(a) to high(a) do
60         b[i]:=a[i];
61     i:=random(high(a)-1)+1;
62     if a[i]<> b[i] then writeln('ERR_COPY');
63
64     end;
65
66 (*процедура смены x2 элтов- массива*)
67 procedure exch(var ar:arr;a,b:longint);
68 var
69     tmp:longint;
70 begin
71     tmp:=ar[a];
72     ar[a]:=ar[b];
73     ar[b]:=tmp;
74 end;
75
76 (*Быстрая сортировка, метод Хоара*)
```

```
77  procedure SortQuick(var a:arr;l,u:longint);
78  var
79      i,j:longint;
80  begin;
81      i:=l;
82       j:=u;
83      repeat
84          while i<>j do begin
85              if a[i]>=a[j] then
86                  dec(j)
87              else begin
88                  exch(a,i,j);
89                  break
90              end;
91          end;
92          while i<>j do begin
93              if a[i]>=a[j] then
94                  inc(i)
95              else begin
96                  exch(a,i,j);
97                  break
98              end;
99          end;
100     until i=j;
101     if i-1>l then
102                     SortQuick(a,l,i-1);
103     if j+1<u then
104                     SortQuick(a,j+1,u);
105 end;
106
107 (*Сортировка бинарным деревом*)
108 (*Обход дерева*)
109 procedure SurTree(node:PTreeType; var  a:arr; i:longint);
110 begin
111         if node<>nil then begin
112         SurTree(node^.right,a,i);
113         a[IndPrintTree]:=node^.data;
114         inc(IndPrintTree);
115         SurTree(node^.left,a,i);
116         end;
117 end;
118
119 (*Поиск со вставкой элта-*)
120 procedure SearchInsert(root:PTreeType; val:longint);
121 var
122     current:PTreeType;
123     prev:PTreeType;
124     pnew:PTreeType;
125 begin
126     current:=root;
127    while(current<>nil)do begin
128         prev:=current;
129         if val< current^.data then current:=current^.left
130         else current:=current^.right;
131     end;
132     new(pnew);
133     pnew^.data:=val;
134     pnew^.left:=nil;
135     pnew^.right:=nil;
136     if val<prev^.data then begin
137              prev^.left:=pnew;
138              pnew^.up:=prev^.left;
139         end
140         else begin
141             prev^.right:=pnew;
142             pnew^.up:=prev^.right;
143        end;
144 end;
145
146
147
148 (*Сортировка бинарным деревом*)
149 procedure SortTree(var  a:arr);
150 var
151 i:longint;
152 root: PTreeType;
```

```
153 begin
154 IndPrintTree:=0;
155     new(root);
156     root^.data:=a[0];
157     root^.left:=nil;
158     root^.right:=nil;
159     root^.up:=nil;
160     for i:=1 to high(a) do
161         SearchInsert(root, a[i]);
162     SurTree(root,a,0);
163 end;
164
165
166
167 (*Процедура печати таблицы*)
168 procedure print(a:TableType;b:TableType );
169 var
170     i:menu2;
171 begin
172     writeln();
173     writeln('␣':15, 'Сортировка␣бинарным␣деревом␣/␣Быстрая␣сортировка␣мс()');
174     write('Размер' :15,'␣␣␣␣␣␣␣␣Упорядоченные':15, '␣␣␣␣␣␣␣Обратный␣порядок␣␣␣␣␣␣␣␣');
175     writeln('Вырожденные␣␣␣␣␣␣␣␣␣␣␣Случайные');
176     for i:=t250 to t16000 do begin
177         write(a[i].size :7,'␣␣␣␣', a[i].Time[SORTED]:10:4 ,'␣/␣', b[i].Time[SORTED]:5:4,'␣␣␣␣');
178         write( a[i].Time[ASORTED]:10:4 ,'␣/␣', b[i].Time[ASORTED]:5:4,'␣␣␣␣');
179         write( a[i].Time[CHILD]:10:4 ,'␣/␣', b[i].Time[CHILD]:5:4,'␣␣␣␣');
180         writeln( a[i].Time[RAND]:10:4 ,'␣/␣', b[i].Time[RAND]:5:4,'␣␣␣␣');
181
182
183
184     end;
185 end;
186
187 (*Главноая часть программы*)
188 var
189     mas1,mas2:arr;
190     i,max :longint;
191     t2:menu2;
192     t3:menu3;
193     TableBinTreeSort:TableType;
194     TableQSort: TableType;
195 begin
196     randomize;
197     for t3:=SORTED to RAND do
198          for t2:=t250 to t16000 do begin
199             case t2 of(*Определение размера массива*)
200                 t250:max:=250;
201                 t500: max:=500;
202                 t1000: max:=1000;
203                 t2000: max:=2000;
204                 t4000: max:=4000;
205                 t8000: max:=8000;
206                 t16000: max:=16000;
207             end;
208             setlength(mas1,max);
209             setlength(mas2,max);
210             TableBinTreeSort[t2].size:=max;
211             TableQSort[t2].size:=max;
212             dec(max);
213             case t3 of(*Определение типа заполнения массива*)
214                 SORTED: begin
215                     mas1[0]:=maxnumber;
216                     for i:=1 to high(mas1) do
217                         mas1[i]:=mas1[i-1]-random(maxnumber div (max+1))-1;
218                 end;
219                 ASORTED: begin
220                     mas1[0]:=1;
221                     for i:=1 to high(mas1) do
222                         mas1[i]:=mas1[i-1]+random(maxnumber div (max+1))-1;
223                 end;
224                 CHILD: for i:=0 to high(mas1) do
225                     mas1[i]:=random(12)+1;
226                 RAND: for i:=low(mas1) to high(mas1) do
227                     mas1[i]:=random(maxnumber)+1;
228             end;
```

```
229
230                 CopyArr(mas1,mas2);(*копирование массива*)
231                 init_timer();(*инициализация таймера*)
232                     SortTree(mas1);(*сортировка по алгоритму 1*)
233                     TableBinTreeSort[t2].Time[t3]:=get_timer()/1000;(*сохранение результата времени сортировки
234                     if(testsort(mas1)=false) then writeln ('Ошибка: массив не был отсортирован');(*проверка н
235                 CopyArr(mas2,mas1);(*восстановление массива из копии*)
236                     init_timer();
237                     SortQuick(mas1,0,high(mas1)+1);
238                     TableQSort[t2].Time[t3]:=get_timer()/1000;
239                     if(testsort(mas1)=false) then writeln ('Ошибка: массив не был отсортирован');
240             end;
241         print(TableBinTreeSort,TableQSort);(*печать таблицы с результатами*)
242     end.
```

Результат выполнения программы:

Сортировка бинарным деревом / Быстрая сортировка мс()

Размер	Упорядоченные	Обратный порядок	Вырожденные	Случайные
250	0.1960 / 0.1190	0.1600 / 0.1230	0.0350 / 0.0280	0.0300 / 0.0260
500	0.7090 / 0.4680	0.6580 / 0.4890	0.1150 / 0.0700	0.1000 / 0.0590
1000	2.9610 / 1.8630	2.9240 / 1.8770	0.3740 / 0.2040	0.1780 / 0.1320
2000	11.9810 / 7.4480	11.9870 / 7.4240	1.3910 / 0.7020	0.3780 / 0.2910
4000	46.6130 / 29.6600	46.6950 / 27.7340	5.6300 / 2.3240	0.8220 / 0.6340
8000	184.4960 / 118.2820	173.3080 / 95.6250	22.7970 / 9.8340	1.7980 / 1.3840
16000	739.9360 / 478.4660	498.1160 / 210.0860	93.7220 / 40.4560	3.8360 / 3.0040

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) ;

PTreeType.Тип, определяющий указатель на структуру **rectree**. Применяется в алгоритме сортировки методом бинарного дерева;

rectree. Тип, определяемый как запись из: **data** - числа целого типа, обозначаемого значение листа, и 3х указаителей на структуры тогоже типа : **left, right, up** обозначающих 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 $u = high(ar) + 1$;

Были добавлены рекурсивные функции для дерева. Потом добавлю

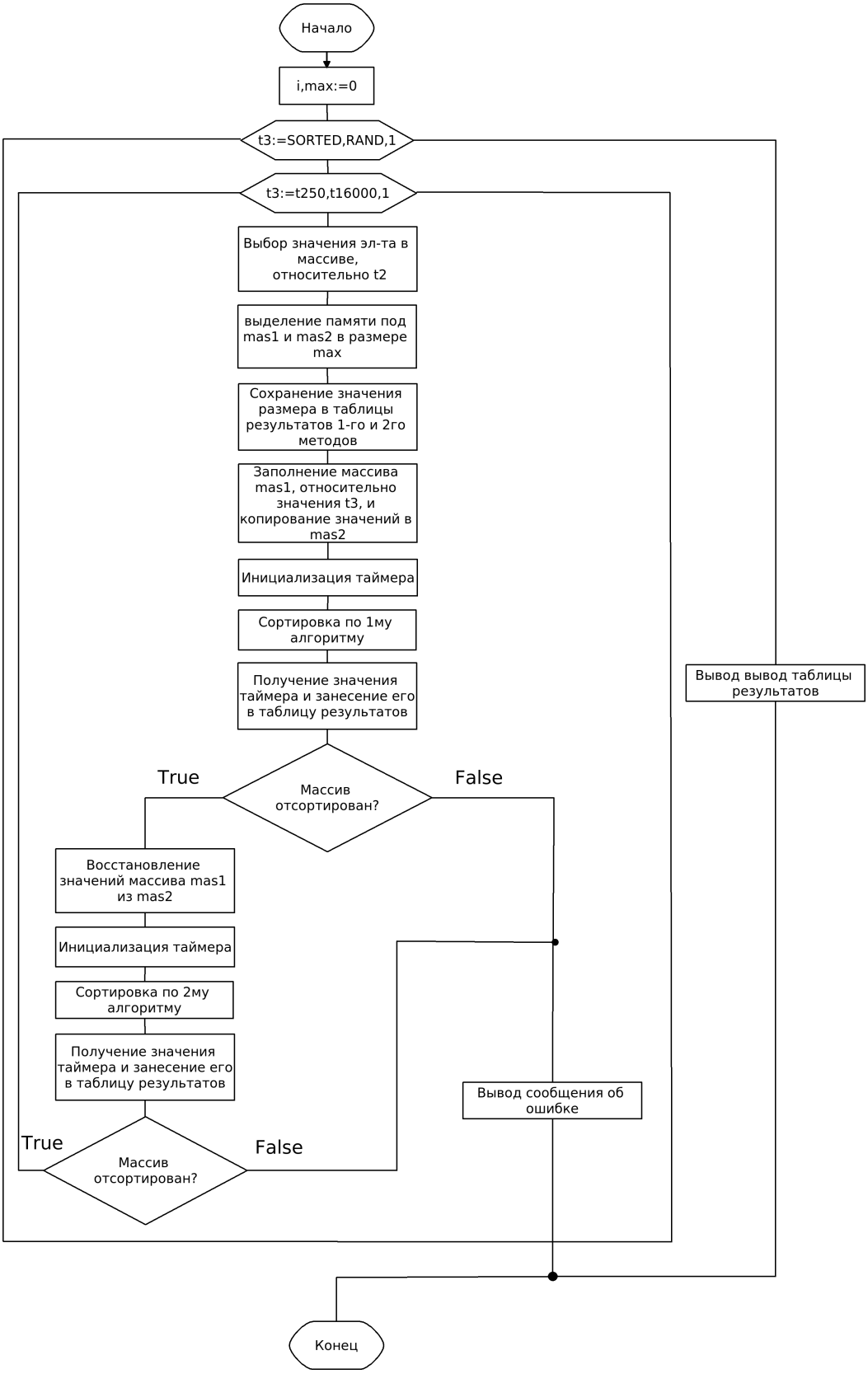
4.4.5 SortTree(a:arr)

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)

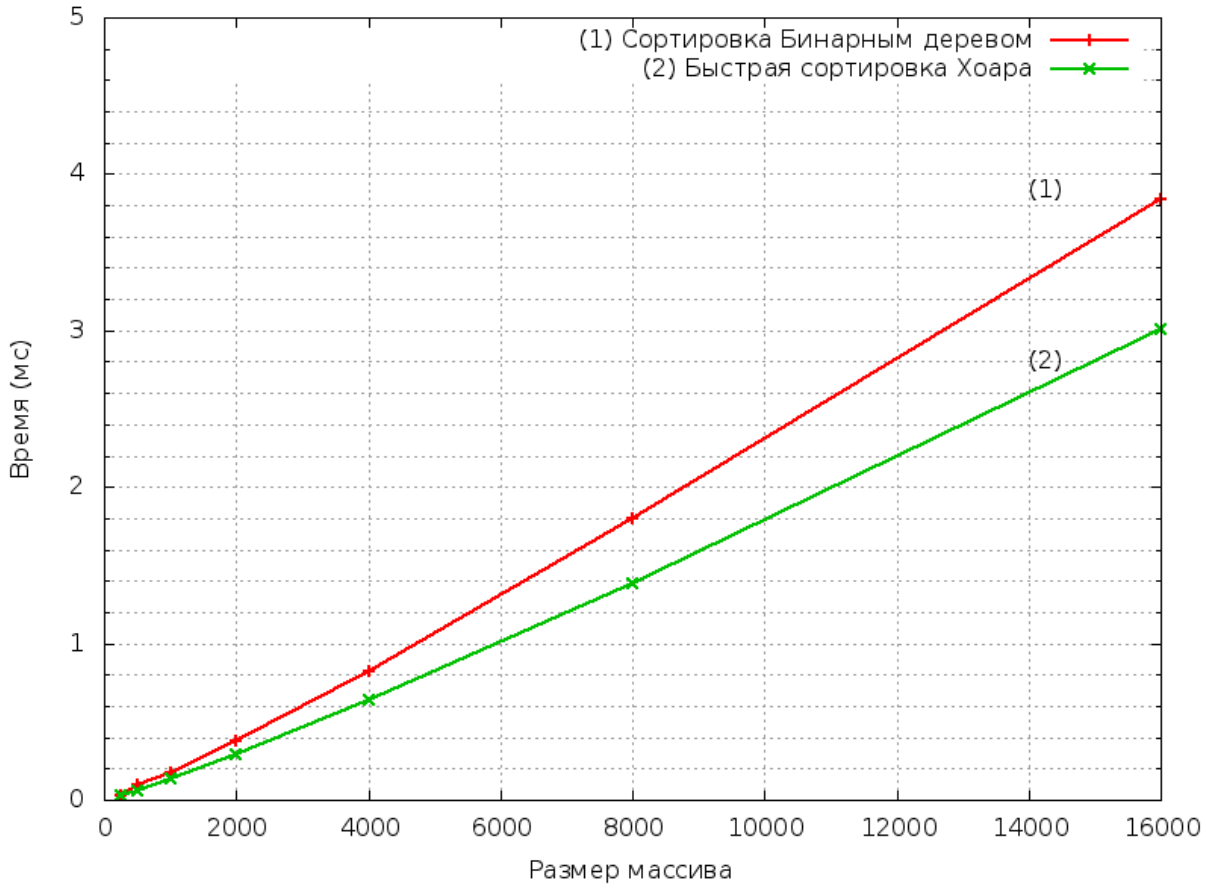
Процедура, выполняющая печать заранее приготовленных таблиц с результатами: a содержит результаты сортировки бинарным деревом, а b результаты быстрой сортировки Хоара.

4.5 Блок-схема программы


Начало
i,max:=0
t3:=SORTED,RAND,1
t3:=t250,t16000,1
Выбор значения эл-та в массиве, относительно t2
выделение памяти под mas1 и mas2 в размере max
Сохранение значения размера в таблицы результатов 1-го и 2го методов
Заполнение массива mas1, относительно значения t3, и копирование значений в mas2
Инициализация таймера
Сортировка по 1му алгоритму
Получение значения таймера и занесение его в таблицу результатов
Вывод вывод таблицы результатов
True
Массив отсортирован?
False
Восстановление значений массива mas1 из mas2
Инициализация таймера
Сортировка по 2му алгоритму
Получение значения таймера и занесение его в таблицу результатов
Вывод сообщения об ошибке
True
Массив отсортирован?
False
Конец

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.