МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**Сравнение методов упорядочивания (сортировки)**

**массивов данных**

**Выполнил:** студент группы 0813-1

В.И. Репин

**Проверил:**

Доцент, к.ф.-м.н.

Баркалов А.В.

Нижний Новгород  
2016

Содержание

Введение . . . . . . . . . 3

1. Постановка учебно-практической задачи . . . 3

2. Руководство пользователя . . . . . 4

3. Руководство программиста . . . . . 5

3.1. Описание алгоритмов сортировки . . . 5

3.2. Описание структур данных . . . . 6

3.3. Описание структуры программного комплекса . 6

Заключение . . . . . . . . . 9

Список литературы . . . . . . . 9

Приложение . . . . . . . . 9

**Введение**

Задача поиска в программировании является крайне важной, при этом поиск в упорядоченном наборе данных гораздо эффективнее, поэтому важно не только изучить различные алгоритмы, позволяющие упорядочивать данные, но и сравнить их по эффективности, в зависимости от количества данных. Простейшим критерием эффективности является время работы алгоритма.

В отчёте приводится постановка задачи, описание алгоритмов сортировки, описание программы и правила её использования, а также прилагается код программы, решающей поставленную задачу.

**1. Постановка учебно-практической задачи**

*Формулировка задачи:*

Необходимо разработать программу, позволяющую сравнить время сортировки различными алгоритмами массива из заданного количества элементов.

*Исходные данные:*

arr\_cur – массив целых чисел;

n\_curr – число элементов в массиве;

*Требуемый результат:*

Вывод фрагмента arr\_curr упорядоченного, количества его элементов n\_curr, название используемого алгоритма сортировки, времени сортировки time. Поставленную задачу требуется решить для типов сортировки: пузырьком, выбором, вставками (с бинарным поиском), слиянием, Шелла, быстрая. Разработать методику сравнения результатов и возможность повторить тесты для различных массивов. Сортировка производится по возрастанию.

*Контрольный пример:*

На рис. 1 результаты всех сортировок для произвольных массивов в 50000 и 25000 элементов.



Рис . 1 Результаты сортировок произвольных массивов

**2. Руководство пользователя**

Для выполнения лабораторной работы по исследованию влияния типа алгоритма сортировки и количества сортируемых данных на длительность необходимо запустить на выполнение файл **compare\_of\_sorting.exe**. На экране появится меню (рис. 2).



Рис . 2 Меню программы

Пользователь может воспользоваться массивом по умолчанию, или сформировать новый, выбрав **1** пункт меню. На экране появится запрос:  
«Введите количество элементов 1000..50000». Следует ввести количество элементов в массиве **n**. Формирование массива производится программно с помощью генератора псевдослучайных чисел в интервале **–n .. n**. Текущий массив в шапке меню обновится.

Пункт **2** меню позволяет просмотреть результаты всех сортировок в форме таблицы, для последующего сравнения.

Пункты **3-8** запускают соответственно сортировку текущего массива выбранным способом. В качестве результата на экран выводится:

* Фрагмент отсортированного массива;
* Название алгоритма сортировки;
* Количество элементов сортируемого массива;
* Время сортировки.



C:\Users\Владимир\Desktop\Без имени-5.jpg

Рис . 3 Результат одной сортировки



После выполнения пункта **1-8** пользователь возвращается в меню.

Для выхода из программы необходимо выбрать **9** пункт меню.

**3. Руководство программиста**

**3.1 Описание алгоритмов сортировки**

1. Сортировка пузырьком (bubble)

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

1. Сортировка выбором (select)

Находим номер минимального значения в текущем массиве, производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции). Теперь сортируем остаток массива, исключив из рассмотрения уже отсортированные элементы.

1. Сортировка вставками (insert)

Элементы массива просматриваются по одному, и для каждого находится позиция в уже отсортированной части массива. Элементы, начиная с этой позиции сдвигаются на 1 позицию, а текущий элемент помещается в отсортированную часть массива.

1. Сортировка слиянием (merge)

Рекурсивный алгоритм, массив делится на две части, которые сортируются отдельно этим же способом, затем они сливаются в сортированный массив. Деление происходит до одного элемента, так как массив из одного элемента считается упорядоченным.

1. Сортировка Шелла (shell)

Выбирается некое значение d, формируются списки элементов, находящихся на расстоянии в d позиций, они сортируются отдельно между собой, но внутри массива. Значение d уменьшается. Повторяется до тех пор, пока d>=1.

1. Быстрая сортировка (quick)

Рекурсивный алгоритм, выбирается опорный элемент d, наиболее эффективны будут значения близкие к медиане массива. Элементы массива упорядочиваются на те, которые <d, и те, которые >=d. Далее тот же алгоритм применяется к этим двум частям массива.

**3.2 Описание структур данных и типов**

Для реализации программы были созданы следующие структуры данных:

const

num\_max = 50000; {Максимальное число элементов в массиве}

num\_min = 1000; {Минимальное число элементов в массиве}

num\_menu = 9; {Число пунктов меню}

type

sort\_types = { Используемые сортировки}

(bubble,

select,

insert,

merge,

shell,

quick);

nvect = array [1..num\_max] of integer {Для сортируемых массивов}

pmenu = array [1..num\_menu] of string; {Для заголовков меню}

psort = array [byte] of sort\_types; {Для типов сортировки}

info\_sort = array[byte] of word; {Для количества элементов, времени сортировки}.

**3.3 Описание структуры программного комплекса.**

{Процедура генерации массива по умолчанию}

procedure default\_gen(

var a: nvect); {Массив для заполнения}

{Процедура вызова меню}

function menu(

names: pmenu; {Названия пунктов меню}

amount: byte; {Количество пунктов меню}

const a: nvect; {Текущий массив для вывода}

n: word) {Количество элементов в текущем массиве}

: char; {Выбранный пользователем пункт меню}

{Процедура генерации массива с заданным числом элементов}

procedure arr\_gen(

var na: word; {Необходимое количество элементов}

var a: nvect); {Массив для заполнения}

{ Процедура вывода таблицы результатов }

procedure print\_res(

count: byte; {Количество тестов}

n: info\_sort; {Массив количества элементов для каждого теста}

names: psort; {Массив названий сортировок для каждого теста}

time: info\_sort); {Массив времени сортировок для каждого теста}

{ Процедура сортировки bubble}

procedure bubble\_sort(

n: word; {Количество элементов}

var a: nvect); {Сортируемый массив}

{ Процедура сортировки select}

procedure select\_sort(

n: word; {Количество элементов}

var a: nvect); {Сортируемый массив}

{ Процедура сортировки insert}

procedure insert\_sort(

n: word; {Количество элементов}

var a: nvect); {Сортируемый массив}

{ Процедура сортировки merge}

procedure merge\_sort(

st, fn: word; {Индексы первого и последнего элемента}

var a: nvect); {Сортируемый массив}

{ Процедура сортировки shell}

procedure shell\_sort(

n: word; {Количество элементов}

var a: nvect); {Сортируемый массив}

{ Процедура сортировки quick}

procedure quick\_sort(

st, fn: word; {Индексы первого и последнего элемента}

var a: nvect); {Сортируемый массив}

{Процедура сортировки в общем виде}

procedure sort(

dif: sort\_types; {Выбор процедуры для сортировки}

n: word; {Количество элементов}

var a: nvect; {Сортируемый массив}

var name: sort\_types; {Имя сортировки}

var time: word); {Время сортировки}

{Основная программа}

var

menu\_names: pmenu = ( {Пункты меню}

'Сформировать новый массив',

'Результаты сортировок',

'Сортировка пузырьком',

'Сортировка выбором',

'Сортировка вставками(бинарный поиск)',

'Сортировка слиянием',

'Сортировка Шелла',

'Быстрая сортировка',

'Выход');

menu\_ch: char; {Переменная выбора пункта меню}

n\_curr: word; {Количество элементов в текущем массиве}

arr\_curr, dup: nvect; {Текущий массив}

test\_n: info\_sort; {Количество элементов для прошлых тестов}

test\_names: psort; {Название сортировок прошлых тестов}

test\_time: info\_sort; {Время прошлых тестов}

test\_amount: byte; {Количество тестов}

Основная программа состоит из вызова процедуры генерации массива по умолчанию, вызова процедуры меню и обработки выбора пользователя.

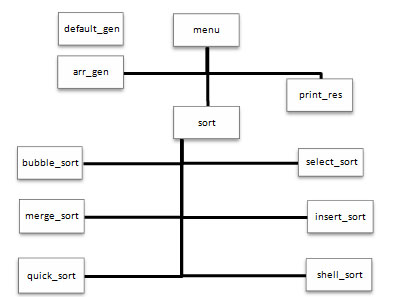


Рис . 4 Схема взаимодействия процедур и функций

Заключение

В лабораторной работе были реализованы алгоритмы сортировки массивов шестью различными методами. Представленный в отчёте программный комплекс позволяет в режиме диалога задавать размер массива, формировать массив целых чисел, распечатывать результаты всех тестов за текущую сессию, печатать результат сортировки и время для каждого отдельного теста с различными исходными данными. По результатам тестов, наиболее эффективными показали себя сортировки слиянием и вставками (рис. 1).

**Список литературы**

1. Д. Кнут Искусство программирования, Т. 3. Сортировка и поиск. - М.: Вильямс, 2007.
2. В. Фаронов Turbo Pascal -СПб.: БХВ-Петербург, 2006.

**Приложение**

program compare\_of\_sort;

uses

crt;

const

num\_max = 50000; //наибольшее число элементов в массиве

num\_min = 1000; // наименьшее число элементов в массиве

num\_menu = 9;//число пунктов меню

type

sort\_types = ( //типы сортировок

bubble,

select,

insert,

merge,

shell,

quick);

nvect = array [1..num\_max] of integer;//для сортируемых массивов

pmenu = array [1..num\_menu] of string;//для заголовков меню

psort = array [byte] of sort\_types;//для хранения информации о том, какие были сортировки

info\_sort = array[byte] of word;//для хранения информации о времени сортировок, количестве элементов

//------Генерация массива по умолчанию--

procedure default\_gen(var a: nvect);

var

i: word;

begin

for i := 1 to num\_max do

a[i] := num\_max - i + 1;

end;

//--------------------------

//--------Меню--------------

function menu(names: pmenu; amount: byte; const a: nvect; n: word): char;

var

i: byte;

k: char;

begin

clrscr;

writeln('Текущий массив из ', n, ' элементов: ');

writeln(a[1], ' ', a[2], ' . . .', a[n - 1], ' ', a[n]);

writeln('------Меню------');

for i := 1 to amount do

writeln(i, '. ', names[i]);

write('Выберите действие: ');

readln(k);

writeln;

menu := k;

end;

//--------------------------

//---------Генерация массива с заданным числом элементов--

procedure arr\_gen(var na: word; var a: nvect);

var

i: word;

temp: string;

err: integer;

begin

repeat

clrscr;

write('Введите количество элементов ', num\_min, '..', num\_max, ': ');

readln(temp);

val(temp, na, err);

if (na > num\_max) or (na < num\_min) or (err > 0) then

begin

writeln('Неверное количество, повторите ввод.');

readln;

end;

until (na <= num\_max) and (na >= num\_min) and (err = 0);

randomize;

for i := 1 to na do

begin

a[i] := random(2 \* na) - na + 1;

end;

end;

//--------------------------

//---------Вывод таблицы результатов--

procedure print\_res(count: byte; n: info\_sort; names: psort; time: info\_sort);

var

i: byte;

begin

clrscr;

writeln('Элементов Тип сортировки Время');

for i := 1 to count do

begin

gotoxy(3, i + 1);

write(n[i]);

gotoxy(19, i + 1);

write(names[i]);

gotoxy(35, i + 1);

write(time[i]);

end;

writeln;

write('Для выхода в меню нажмите любую клавишу.');

readln;

end;

//--------------------------

//------Bubble--------------

procedure bubble\_sort(n: word; var a: nvect);

var

swaps: boolean;

i, j: word;

begin

i:=0;

swaps := true;

while (i < n - 1) and swaps do

begin

swaps := false;

inc(i);

for j := 1 to n - i do

if a[j] > a[j + 1] then

begin

swaps := true;

swap(a[j],a[j+1]);

end;

end;

end;

//--------------------------

//------Select--------------

procedure select\_sort(n: word; var a: nvect);

var

i, min, j: word;

temp: integer;

begin

for i := 1 to n - 1 do

begin

min := i;

for j := i + 1 to n do

if a[j] < a[min] then

min := j;

if min <> i then

begin

temp := a[i];

a[i] := a[min];

a[min] := temp;

end;

end;

end;

//--------------------------

//------Insert-------------

procedure insert\_sort(n: word; var a: nvect);

function bin\_search(n: word; const a: nvect; el: integer): word;

var

st, mid, fn: word;

begin

st := 1;

fn := n;

while(st <= fn) do

begin

mid := (st + fn) div 2;

if el > a[mid] then

st := mid + 1

else fn := mid - 1;

end;

bin\_search := st;

end;

procedure shift(k: word; fn: word; var a: nvect);

var

i: word;

begin

for i := fn downto k do

a[i + 1] := a[i];

end;

var

i, pos: word;

r: integer;

begin

for i := 2 to n do

begin

r := a[i];

pos := bin\_search(i - 1, a, r);

shift(pos, i - 1, a);

a[pos] := r;

end;

end;

//-------------------------

var cop:nvect;

//------Merge--------------

procedure merge\_sort(st, fn: word; var a: nvect);

var

i, j, k,d, mid: word;

begin

mid := (st + fn - 1) div 2;

if mid - st > 0 then

merge\_sort(st, mid, a);

if fn - mid > 1 then

merge\_sort(mid + 1, fn, a);

j := mid + 1; i := st;

k := st - 1;

for d:=st to fn do cop[d]:=a[d];

while(i <= mid) and (j <= fn) do

begin

inc(k);

if cop[i] < cop[j] then

begin

a[k] := cop[i];

inc(i);

end

else

begin

a[k] := cop[j];

inc(j);

end;

end;

if i <= mid then

for j := i to mid do

begin

inc(k);

a[k] := cop[j];

end

else

for i := j to fn do

begin

inc(k);

a[k] := cop[i];

end;

end;

//-------------------------

//---------Shell-----------

procedure shell\_sort(n: word; var a: nvect);

var

d, i, j: word;

temp: integer;

begin

d := n;

d := d div 2;

while (d > 0) do

begin

for i := 1 to n - d do

begin

j := i;

while ((j > 0) and (A[j] > A[j + d])) do

begin

temp := A[j];

A[j] := A[j + d];

A[j + d] := temp;

j := j - 1;

end;

end;

d := d div 2;

end;

end;

//-------------------------

//---------Quick-----------

procedure quick\_sort(st, fn: word; var a: nvect);

var

l, r, i\_op: word;

temp, op: integer;

begin

if fn - st > 0 then

begin

i\_op := (st + fn - 1) div 2;

op := a[i\_op];

l := st;

r := fn;

while(l < r) do

begin

while a[l] < op do

inc(l);

while a[r] > op do

dec(r);

if l < r then

begin

temp := a[l];

a[l] := a[r];

a[r] := temp;

if l = i\_op then

begin

inc(l);

i\_op := r;

end

else if r = i\_op then

begin

dec(r);

i\_op := l;

end

else begin inc(l); dec(r); end;

end;

end;

end;

if l - st > 1 then

quick\_sort(st, l - 1, a);

if fn - l > 1 then

quick\_sort(l + 1, fn, a);

end;

//-------------------------

//-----Сортировка в общем виде--

procedure sort(dif: sort\_types; n: word; var a: nvect; var name: sort\_types; var time: word);

begin

clrscr;

write('Выполняется сортировка...');

name := dif;

time := milliseconds;

case dif of

bubble: bubble\_sort(n, a);

select: select\_sort(n, a);

insert: insert\_sort(n, a);

merge: merge\_sort(1, n, a);

shell: shell\_sort(n, a);

quick: quick\_sort(1, n, a);

end;

time := milliseconds - time;

clrscr;

writeln('Отсортированный массив: ');

writeln(a[1], ' ', a[2], ' . . .', a[n - 1], ' ', a[n]);

writeln('Сортировка ', dif, ' массива из ', n, ' элементов заняла ', time, ' миллисекунд.');

write('Для выхода в меню нажмите любую клавишу.');

readln;

end;

//--------------------------

var

menu\_names: pmenu = (

'Сформировать новый массив',

'Результаты сортировок',

'Сортировка пузырьком',

'Сортировка выбором',

'Сортировка вставками(бинарный поиск)',

'Сортировка слиянием',

'Сортировка Шелла',

'Быстрая сортировка',

'Выход');

menu\_ch: char; // Переменная выбора пункта меню

n\_curr: word; // Количество элементов в текущем массиве

arr\_curr, dup: nvect; // Текущий массив

test\_n: info\_sort; // Количество элементов для прошлых тестов

test\_names: psort; // Название сортировок прошлых тестов

test\_time: info\_sort; // Время прошлых тестов

test\_amount: byte;// Количество тестов

begin

n\_curr := num\_max; // Генерация массива по умолчанию

default\_gen(arr\_curr); //

repeat

menu\_ch := menu(menu\_names, num\_menu, arr\_curr, n\_curr);

if (menu\_ch >= '3') and (menu\_ch <= chr(ord('0') + ord(num\_menu - 1))) then //Дупликация массива,увеличение кол-ва сортировок,

begin//если в меню была выбрана сортировка

inc(test\_amount);

dup := arr\_curr;

test\_n[test\_amount] := n\_curr;

end;

case menu\_ch of

'1': arr\_gen(n\_curr, arr\_curr);

'2': print\_res(test\_amount, test\_n, test\_names, test\_time);

'3': sort(bubble, n\_curr, dup, test\_names[test\_amount], test\_time[test\_amount]);

'4': sort(select, n\_curr, dup, test\_names[test\_amount], test\_time[test\_amount]);

'5': sort(insert, n\_curr, dup, test\_names[test\_amount], test\_time[test\_amount]);

'6': sort(merge, n\_curr, dup, test\_names[test\_amount], test\_time[test\_amount]);

'7': sort(shell, n\_curr, dup, test\_names[test\_amount], test\_time[test\_amount]);

'8': sort(quick, n\_curr, dup, test\_names[test\_amount], test\_time[test\_amount]);

'9': ;

else

begin

write('Неверный пункт меню, повторите ввод ');

readln;

end;

end;

until menu\_ch = chr(ord('0') + ord(num\_menu));

end.