

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №1**  
**по дисциплине «Качество и метрология программного обеспечения»**  
**ТЕМА: «Расчет метрических характеристик качества разработки**  
**программ по метрикам Холстеда»**

Студентка гр. 6304

Прозорова А. Д.

Преподаватель

Кирияничков В.А.

Санкт-Петербург

2020

## Задание

Для заданного варианта программы обработки данных, представленной на языке Паскаль, разработать вычислительный алгоритм и также варианты программ его реализации на языках программирования Си и Ассемблер. Добиться, чтобы программы на Паскале и Си были работоспособны и давали корректные результаты (это потребуется в дальнейшем при проведении с ними измерительных экспериментов).

Для каждой из разработанных программ (включая исходную программу на Паскале) определить следующие метрические характеристики (по Холстеду):

### 1. Измеримые характеристики программ:

- число простых(отдельных)операторов, в данной реализации;
- число простых (отдельных) операндов, в данной реализации;
- общее число всех операторов в данной реализации;
- общее число всех операндов в данной реализации;
- число вхождений  $j$ -го оператора в тексте программы;
- число вхождений  $j$ -го операнда в тексте программы;
- словарь программы;
- длину программы.

### 2. Расчетные характеристики программы:

- длину программы;
- реальный и потенциальный объемы программы;
- уровень программы;
- интеллектуальное содержание программы;
- работу программиста;
- время программирования;
- уровень используемого языка программирования;
- ожидаемое число ошибок в программе.

Для характеристик длина программы, уровень программы, время программирования следует рассчитать как саму характеристику, так и ее оценку.

## Ход работы

### Вариант 12. Быстрая сортировка (нерекурсивный вариант).

1. Определение метрических характеристик для программы на Pascal.

Код программы представлен в приложении А.

Ручной расчёт измеримых характеристик представлен в таблице 1.

Таблица 1 – Ручной расчёт измеримых характеристик (Pascal)

№	Оператор	Количество	№	Операнд	Количество
1	;	40	1	max	3
2	:=	26	2	ary	1
3	() или begin end	32	3	i	27
4	[]	44	4	j	19
5	+	9	5	n	6
6	-	8	6	left	9
7	>	6	7	right	10
8	<	9	8	hold	3
9	>=	2	9	p	3
10	for to do	1	10	q	3
11	If then	3	11	sp	22
12	If then else	3	12	mid	9
13	sort	2	13	pivot	9
14	swap	4	14	l	20
15	randomize	1	15	0	1
16	random	1	16	80	1
17	div	1	17	20	1
18	while	4	18	2	1
19	end	8	19	5	1
20	or	7	20	x	28
Всего		211	Всего		177

Файл с результатами программных расчётов представлен в приложении

Б.

Таблица 2 – Программный расчёт измеримых характеристик (Pascal)

№	Оператор	Количество	№	Операнд	Количество
1	()	25	1	0	1
2	+	9	2	1	20
3	-	8	3	100	1
4	/	1	4	2	1
5	;	51	5	20	1
6	<	9	6	5	1
7	=	27	7	80	1
8	>	6	8	ary	1
9	>=	2	9	hold	3
10	[]	42	10	i	27
11	and	5	11	j	19
12	ary	2	12	left	9
13	Const	1	13	max	3
14	for	14	14	mid	9
15	if	6	15	n	6
16	integer	4	16	nonrecursive_quicksort	1
17	or	2	17	p	3
18	procedure	2	18	pivot	9
19	program	1	19	q	3
20	random	1	20	right	10
21	randomize	1	21	sp	22
22	real	4	22	x	28
23	sort	2	Всего		179
24	swap	5			
25	type	1			
26	while	4			
Всего		222			

Определение расчетных характеристик представлено в таблице 3.

Таблица 3 – Расчёт расчетных характеристик (Pascal)

Характеристика	Ручной расчёт	Программный расчёт
Число простых операторов $n_1$	20	26
Число простых операндов $n_2$	20	22
Общее число всех операторов $N_1$	211	222
Общее число всех операндов $N_2$	177	179
Словарь $n$	40	48
Длина $N_{\text{опыт}}$	388	401
Теоретическая длина $N_{\text{теор}}$	172.877	220.319
Объём $V$	2064.91	2239.57
Потенциальный объём $V^*$	11.609	11.609
Уровень программы $L$	0.0056	0.00518
Оценка уровня программы $L^{\sim}$	0.011	0.0095
Интеллектуальное содержание $I$	23.33	21.173
Работа программирования $E$	367267	432026
Время программирования $T$	36726.7	43202.7
Уровень языка $\lambda$	0.0653	0.0602
Ожидаемое число ошибок в программе $B$	3	3

2. Определение метрических характеристик для программы на Си.

Код программы представлен в приложении В.

Ручной расчёт измеримых характеристик представлен в таблице 4.

Таблица 4 – Ручной расчёт измеримых характеристик (Си)

№	Оператор	Количество	№	Операнд	Количество
1	;	38	1	hold	2
2	=	22	2	p	3
3	() или {}	39	3	q	3
4	[]	45	4	x	28
5	for	1	5	n	2
6	if	3	6	i	28

7	If else	3	7	j	18
8	<	10	8	left	9
9	>	6	9	right	10
10	>=	2	10	sp	21
11	+	7	11	pivot	9
12	++	3	12	mid	9
13	-	9	13	0	6
14	--	1	14	1	12
15	*	7	15	20	2
16	&	8	16	100	1
17	Return	1	17	80	3
18	&&	5	18	5	1
19		2	19	2	1
20	swap	5	20	NULL	1
21	srand	1	Всего		169
22	time	1			
23	rand	1			
24	%	2			
25	sort	2			
26	while	4			
Всего		228			

Программный расчёт измеримых характеристик представлен в таблицу  
5. Файл с результатами программных расчётов представлен в приложении Г.

Таблица 5 – Программный расчёт измеримых характеристик (Си)

№	Оператор	Количество	№	Операнд	Количество
1	%	2	1	0	6
2	&&	5	2	1	12
3	()	17	3	100	1
4	+	7	4	2	1
5	++	3	5	20	2
6	,	10	6	5	1
7	-	8	7	80	3

8	--	1	8	NULL	1
9	;	40	9	hold	2
10	<	10	10	i	28
11	=	22	11	j	18
12	>	6	12	left	9
13	>=	2	13	mid	9
14	[]	42	14	n	2
15	_&	8	15	p	3
16	__*	4	16	pivot	9
17	_-	1	17	q	3
18	_[]	3	18	right	10
19	__*	3	19	sp	21
20	float	6	20	x	28
21	for	1	Всего		169
22	if	6			
23	int	6			
24	main	1			
25	rand	1			
26	return	1			
27	sort	2			
28	srand	1			
29	swap	5			
30	time	1			
31	void	2			
32	while	4			
33		2			
Всего		233			

Определение расчетных характеристик представлено в таблице 6.

Таблица 6 – Расчёт расчетных характеристик (Си)

Характеристика	Ручной расчёт	Программный расчёт
----------------	---------------	--------------------

Число простых операторов $n_1$	26	33
Число простых операндов $n_2$	20	20
Общее число всех операторов $N_1$	228	233
Общее число всех операндов $N_2$	169	169
Словарь $n$	46	53
Длина $N_{\text{опыт}}$	397	402
Теоретическая длина $N_{\text{теор}}$	208.65	252.904
Объём $V$	2192.85	2302.62
Потенциальный объём $V^*$	11.609	11.609
Уровень программы $L$	0.00529	0.00504
Оценка уровня программы $L^{\sim}$	0.0091	0.0072
Интеллектуальное содержание $I$	19.96	16.52
Работа программирования $E$	414191	456696
Время программирования $T$	41419.1	45669.6
Уровень языка $\lambda$	0.0614	0.0585
Ожидаемое число ошибок в программе $B$	3	3

### 3. Определение метрических характеристик для программы на Ассемблере.

Код программы представлен в приложении Д.

Ручной расчёт измеримых характеристик представлен в таблице 7.

Таблица 7 – Ручной расчёт измеримых характеристик (Ассемблер)

№	Оператор	Количество	№	Операнд	Количество
1	pushq	3	1	%rbp	15



2	movq	38	2	%rsp	8
3	subq	3	3	0	2
4	movss	28	4	%rcx	15
5	nop	3	5	\$16	2
6	addq	27	6	16(%rbp)	28
7	popq	3	7	%rdx	45
8	ret	3	8	24(%rbp)	5
9	movl	85	9	%rax	62
10	subl	10	10	(%rax)	20
11	jmp .L3	3	11	%xmm0	32
12	cltq	36	12	-4(%rbp)	25
13	cmpl	8	13	\$224	2
14	jl .L4	1	14	%edx	34
15	leaq	26	15	\$0	6
16	sarl	2	16	-112(%rbp)	1
17	shrl	1	17	%eax	83
18	andl	1	18	\$1	11
19	jle .L5	1	19	-192(%rbp)	1
20	comiss	10	20	-12(%rbp)	19
21	jbe .L6	1	21	-112(%rbp,%rax,4)	7
22	ja .L8	1	22	-192(%rbp,%rax,4)	8
23	jbe .L9	2	23	-8(%rbp)	16
24	call swap	4	24	0(,%rax,4)	24
25	jmp .L5	1	25	-16(%rbp)	8
26	jbe .L12	1	26	\$31	2
27	ja .L14	1	27	-20(%rbp)	8
28	jbe .L5	1	28	\$5	2
29	jmp .L17	1	29	%xmm1	14
30	jns .L26	1	30	1(%rax)	6
31	call __main	1	31	-1(%rax)	2
32	call time	1	32	\$368	2
33	call srand	1	33	368	1
34	jmp .L32	1	34	128(%rsp)	1
35	call rand	1	35	128	1

36	imulq	1	36	236(%rbp)	4
37	shrq	1	37	\$1374389535	1
38	cltd	1	38	\$100	1
39	imull	1	39	-96(%rbp,%rax,4)	1
40	cvtsi2ssl	1	40	\$79	1
41	jle .L33	1	41	-96(%rbp)	1
42	call sort	1	42	\$80	1
43	swap	1			
Всего		319	Всего		528

Определение расчетных характеристик представлено в таблице 8.

Таблица 8 – Расчёт расчетных характеристик (Ассемблер)

Характеристика	Ручной расчёт
Число простых операторов $n_1$	43
Число простых операндов $n_2$	42
Общее число всех операторов $N_1$	319
Общее число всех операндов $N_2$	528
Словарь $n$	85
Длина $N_{\text{опыт}}$	847
Теоретическая длина $N_{\text{теор}}$	459.81
Объём $V$	5428.75
Потенциальный объём $V^*$	11.609
Уровень программы $L$	0.0021
Оценка уровня программы $L^{\sim}$	0.0038
Интеллектуальное содержание $I$	20.085
Работа программирования $E$	2538525
Время программирования $T$	253852.6
Уровень языка $\lambda$	0.0248
Ожидаемое число ошибок в программе $B$	6

#### 4. Сравнение результатов определения метрических характеристик.

Таблица 9 – Сводная таблица расчетов на трех языках

Характеристика	Ручной расчёт Pascal	Програм- мный расчёт Pascal	Ручной расчёт Си	Програм- мный расчёт Си	Ручной расчёт Ассемблер
Число простых операторов $n_1$	20	26	26	33	43
Число простых операндов $n_2$	20	22	20	20	42
Общее число всех операторов $N_1$	211	222	228	233	319
Общее число всех операндов $N_2$	177	179	169	169	528
Словарь $n$	40	48	46	53	85
Длина $N_{\text{опыт}}$	388	401	397	402	847
Теоретическая длина $N_{\text{теор}}$	172.877	220.319	208.65	252.904	459.81
Объём $V$	2064.91	2239.57	2192.85	2302.62	5428.75
Потенциальный объём $V^*$	11.609	11.609	11.609	11.609	11.609
Уровень программы	0.0056	0.00518	0.00529	0.00504	0.0021
Оценка уровня программы $L^{\sim}$	0.011	0.0095	0.0091	0.0072	0.0038
Интеллектуальное содержание $I$	23.33	21.173	19.96	16.52	20.085
Работа программирования $E$	367267	432026	414191	456696	2538525
Время программирования $T$	36726.7	43202.7	41419.1	45669.6	253852.6
Уровень языка $\lambda$	0.0653	0.0602	0.0614	0.0585	0.0248
Ожидаемое число ошибок в программе $B$	3	3	3	3	6

Результаты сравнения показывают, что самый низкий уровень у программы на Ассемблере, а самый высокий у программы на Pascal. Наибольшие показатели

времени программирования, работы программирования и ожидаемого числа ошибок, наоборот, соответствуют Ассемблеру, а наименьший – Pascal.

## **Выводы**

В результате выполнения данной лабораторной работы была изучена система метрик Холстеда. Было проведено сравнение программ, реализующих алгоритм быстрой сортировки (нерекурсивный вариант), на языках Pascal, Си и Ассемблер.

## ПРИЛОЖЕНИЕ А

### Код программы на Pascal.

```
program nonrecursive_quicksort;

const max = 80;
type ary = array[1..max] of real;
var x : ary; i,n : integer;

procedure sort(var x: ary; n: integer);
    var    left,right    : array[1..20] of integer;
          i,j,sp,mid     : integer;
          pivot          : real;

    procedure swap(var p,q: real);
        var    hold : real;
        begin
            hold:=p;
            p:=q;
            q:=hold
        end;

    begin
        left[1]:=1;
        right[1]:=n;
        sp:=1;
        while sp > 0 do
            begin
                if left[sp]>=right[sp] then sp:=sp-1
                else
                    begin
                        i:=left[sp];
                        j:=right[sp];
                        pivot:=x[j];
                        mid:=(i+j)div 2;
                        if (j-i)>5 then
                            if ((x[mid]>pivot)and(x[mid]<x[i])) or
                                ((x[i]>x[mid])and(x[i]<pivot)) or
                                ((x[mid]<pivot)and(x[mid]>x[i])) or
                                ((x[i]<x[mid])and(x[i]>pivot))
                                then swap(x[mid],x[j])
                                else if((x[i]<x[mid])and(x[i]>pivot)) or
                                    ((x[mid]<pivot)and(x[mid]>x[i])) or
                                    ((x[i]>x[mid])and(x[i]<pivot))
                                    then swap(x[i],x[j]);
                                pivot:=x[j];

                        while i<j do
                            begin
                                while x[i]<pivot do
                                    i:=i+1;
                                j:=j-1;
                                while (i<j)and(pivot<x[j]) do
                                    j:=j-1;
                                if i<j then swap(x[i],x[j])
                            end;

                        j:=right[sp];
                        swap(x[i],x[j]);
                        if i-left[sp]>=right[sp]-i then
                            begin
                                { put shorter part first }
                                left[sp+1]:=left[sp];
                                right[sp+1]:=i-1;
                                left[sp]:=i+1
                            end
                    end
            end
        end
```

```

                                end
                                else
                                    begin
                                        left[sp+1]:=i+1;
                                        right[sp+1]:=right[sp];
                                        right[sp]:=i-1
                                    end;
                                sp:=sp+1
                                end
                                end
                                end;

begin
    n:=max;
    randomize;
    for i:=1 to n do
        x[i]:= random(100);
    sort( x,n );
end.

```

## ПРИЛОЖЕНИЕ Б

### Результаты parser\_pas.exe

Statistics for module pascal.lxm

```
=====
The number of different operators   : 26
The number of different operands    : 22
The total number of operators       : 222
The total number of operands        : 179

Dictionary      ( D) : 48
Length          ( N) : 401
Length estimation ( ^N) : 220.319
Volume          ( V) : 2239.57
Potential volume (*V) : 19.6515
Limit volume     (**V) : 38.2071
Programming level ( L) : 0.00877467
Programming level estimation ( ^L) : 0.00945423
Intellect       ( I) : 21.1734
Time of programming ( T) : 14179.5
Time estimation   ( ^T) : 7230.58
Programming language level (lambda) : 0.172435
Work on programming ( E) : 255231
Error            ( B) : 1.34122
Error estimation  ( ^B) : 0.746523
```

Table:

=====

Operators:

1	25	()
2	9	+
3	8	-
4	1	/
5	51	;
6	9	<
7	27	=
8	6	>
9	2	>=
10	42	[]
11	5	and
12	2	ary
13	1	const
14	1	for
15	6	if
16	4	integer
17	2	or
18	2	procedure
19	1	program
20	1	random
21	1	randomize
22	4	real
23	2	sort
24	5	swap
25	1	type
26	4	while

#### Operands:

1	1	0
2	20	1
3	1	100
4	1	2
5	1	20
6	1	5
7	1	80
8	1	ary
9	3	hold
10	27	i
11	19	j
12	9	left
13	3	max
14	9	mid
15	6	n
16	1	nonrecursive_quicksort
17	3	p
18	9	pivot
19	3	q
20	10	right
21	22	sp
22	28	x

#### Summary:

=====

The number of different operators	: 26
The number of different operands	: 22
The total number of operators	: 222
The total number of operands	: 179

Dictionary	( D )	: 48
Length	( N )	: 401
Length estimation	( ^N )	: 220.319
Volume	( V )	: 2239.57
Potential volume	( *V )	: 19.6515
Limit volume	( **V )	: 38.2071
Programming level	( L )	: 0.00877467
Programming level estimation	( ^L )	: 0.00945423
Intellect	( I )	: 21.1734
Time of programming	( T )	: 14179.5
Time estimation	( ^T )	: 7230.58
Programming language level	( lambda )	: 0.172435
Work on programming	( E )	: 255231
Error	( B )	: 1.34122
Error estimation	( ^B )	: 0.746523



## ПРИЛОЖЕНИЕ В

### Код программы на Си

```
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

void swap(float *p, float *q) {
    float hold = (*p);
    *p = (*q);
    *q = hold;
}

void sort(float *x, int n) {
    int left[20];
    int right[20];
    int i, j, sp, mid;
    float pivot;

    left[0] = 0;
    right[0] = n-1;
    sp = 0;
    while (sp > -1) {
        if (left[sp] >= right[sp]) sp = sp - 1;
        else {
            i = left[sp];
            j = right[sp];
            pivot = x[j];
            mid = (i + j) % 2;
            if (j - i > 5) {
                if ((x[mid] < pivot && x[mid] > x[i]) || (x[mid] > pivot &&
x[mid] < x[i]))
                    swap(&x[mid], &x[j]);
                else if ((x[i] < x[mid] && x[i] > pivot) || (x[i] > x[mid]
&& x[i] < pivot ))
                    swap(&x[i], &x[j]);
            }
            pivot = x[j];

            while (i < j) {
                while (x[i] < pivot) i++;
                j = j-1;
                while (i < j && pivot < x[j]) j--;
                if (i < j) swap(&x[i], &x[j]);
            }
            j = right[sp];
            swap(&x[i], &x[j]);
            if (i - left[sp] >= right[sp] - i) {
                left[sp+1] = left[sp];
                right[sp+1] = i-1;
                left[sp] = i+1;
            } else {
                left[sp+1] = i+1;
                right[sp+1] = right[sp];
                right[sp] = i-1;
            }
            sp++;
        }
    }
}
```

```
}
```

```
int main(){  
    float x[80];  
    srand(time(NULL));  
    for (int i=0; i <80; i++) {  
        x[i] = rand() % 100;  
    }  
    sort(x,80);  
  
    return 0;  
}
```

## ПРИЛОЖЕНИЕ Г

### Результаты parser\_c.exe

Statistics for module ci.lxm

=====

The number of different operators : 33  
The number of different operands : 20  
The total number of operators : 233  
The total number of operands : 169

Dictionary ( D) : 53  
Length ( N) : 402  
Length estimation ( ^N) : 252.904  
Volume ( V) : 2302.62  
Potential volume ( \*V) : 19.6515  
Limit volume ( \*\*V) : 38.2071  
Programming level ( L) : 0.00853439  
Programming level estimation ( ^L) : 0.00717231  
Intellect ( I) : 16.5151  
Time of programming ( T) : 14989.2  
Time estimation ( ^T) : 11220.7  
Programming language level (lambda) : 0.167713  
Work on programming ( E) : 269805  
Error ( B) : 1.39181  
Error estimation ( ^B) : 0.767541

Table:

=====

Operators:

1	2	%
2	5	&&
3	17	()
4	7	+
5	3	++
6	10	,
7	8	-
8	1	--
9	40	;
10	10	<
11	22	=
12	6	>
13	2	>=
14	42	[]
15	8	_&
16	4	_*
17	1	_ -
18	3	_[]
19	3	_*
20	6	float
21	1	for
22	6	if
23	6	int
24	1	main
25	1	rand
26	1	return
27	2	sort
28	1	srand
29	5	swap
30	1	time
31	2	void

	32		4		while
	33		2		

Operands:

	1		6		0
	2		12		1
	3		1		100
	4		1		2
	5		2		20
	6		1		5
	7		3		80
	8		1		NULL
	9		2		hold
	10		28		i
	11		18		j
	12		9		left
	13		9		mid
	14		2		n
	15		3		p
	16		9		pivot
	17		3		q
	18		10		right
	19		21		sp
	20		28		x

#### Summary:

```

=====
The number of different operators      : 33
The number of different operands      : 20
The total number of operators         : 233
The total number of operands         : 169

Dictionary          ( D)   : 53
Length              ( N)   : 402
Length estimation   ( ^N)   : 252.904
Volume              ( V)   : 2302.62
Potential volume    ( *V)   : 19.6515
Limit volume        (**V)  : 38.2071
Programming level    ( L)   : 0.00853439
Programming level estimation ( ^L) : 0.00717231
Intellect           ( I)   : 16.5151
Time of programming ( T)   : 14989.2
Time estimation      ( ^T)  : 11220.7
Programming language level (lambda) : 0.167713
Work on programming ( E)   : 269805
Error               ( B)   : 1.39181
Error estimation     ( ^B)  : 0.767541

```

## ПРИЛОЖЕНИЕ Д

### Код программы на Ассемблер

```
swap:
    pushq %rbp
    .seh_pushreg %rbp
    movq %rsp, %rbp
    .seh_setframe %rbp, 0
    subq $16, %rsp
    .seh_stackalloc 16
    .seh_endprologue
    movq %rcx, 16(%rbp)
    movq %rdx, 24(%rbp)
    movq 16(%rbp), %rax
    movss (%rax), %xmm0
    movss %xmm0, -4(%rbp)
    movq 24(%rbp), %rax
    movss (%rax), %xmm0
    movq 16(%rbp), %rax
    movss %xmm0, (%rax)
    movq 24(%rbp), %rax
    movss -4(%rbp), %xmm0
    movss %xmm0, (%rax)
    nop
    addq $16, %rsp
    popq %rbp
    ret
    .seh_endproc
    .globl sort
    .def sort; .scl 2; .type 32; .endef
    .seh_proc sort

sort:
    pushq %rbp
    .seh_pushreg %rbp
    movq %rsp, %rbp
    .seh_setframe %rbp, 0
    subq $224, %rsp
    .seh_stackalloc 224
    .seh_endprologue
    movq %rcx, 16(%rbp)
    movl %edx, 24(%rbp)
    movl $0, -112(%rbp)
    movl 24(%rbp), %eax
    subl $1, %eax
    movl %eax, -192(%rbp)
    movl $0, -12(%rbp)
    jmp .L3

.L26:
    movl -12(%rbp), %eax
    cltq
    movl -112(%rbp,%rax,4), %edx
    movl -12(%rbp), %eax
    cltq
    movl -192(%rbp,%rax,4), %eax
    cmpl %eax, %edx
    jl .L4
    subl $1, -12(%rbp)
    jmp .L3

.L4:
    movl -12(%rbp), %eax
    cltq
```

```

movl    -112(%rbp,%rax,4), %eax
movl    %eax, -4(%rbp)
movl    -12(%rbp), %eax
cltq
movl    -192(%rbp,%rax,4), %eax
movl    %eax, -8(%rbp)
movl    -8(%rbp), %eax
cltq
leaq    0(,%rax,4), %rdx
movq    16(%rbp), %rax
addq    %rdx, %rax
movss   (%rax), %xmm0
movss   %xmm0, -16(%rbp)
movl    -4(%rbp), %edx
movl    -8(%rbp), %eax
addl    %eax, %edx
movl    %edx, %eax
sarl    $31, %eax
shrl    $31, %eax
addl    %eax, %edx
andl    $1, %edx
subl    %eax, %edx
movl    %edx, %eax
movl    %eax, -20(%rbp)
movl    -8(%rbp), %eax
subl    -4(%rbp), %eax
cmpl    $5, %eax
jle     .L5
movl    -20(%rbp), %eax
cltq
leaq    0(,%rax,4), %rdx
movq    16(%rbp), %rax
addq    %rdx, %rax
movss   (%rax), %xmm1
movss   -16(%rbp), %xmm0
comiss  %xmm1, %xmm0
jbe     .L6
movl    -20(%rbp), %eax
cltq
leaq    0(,%rax,4), %rdx
movq    16(%rbp), %rax
addq    %rdx, %rax
movss   (%rax), %xmm0
movl    -4(%rbp), %eax
cltq
leaq    0(,%rax,4), %rdx
movq    16(%rbp), %rax
addq    %rdx, %rax
movss   (%rax), %xmm1
comiss  %xmm1, %xmm0
ja      .L8

```

.L6:

```

movl    -20(%rbp), %eax
cltq
leaq    0(,%rax,4), %rdx
movq    16(%rbp), %rax
addq    %rdx, %rax
movss   (%rax), %xmm0
comiss  -16(%rbp), %xmm0
jbe     .L9
movl    -20(%rbp), %eax
cltq

```

```

    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rdx, %rax
    movss   (%rax), %xmm1
    movl    -4(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rdx, %rax
    movss   (%rax), %xmm0
    comiss  %xmm1, %xmm0
    jbe     .L9
.L8:
    movl    -8(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rax, %rdx
    movl    -20(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rcx
    movq    16(%rbp), %rax
    addq    %rcx, %rax
    movq    %rax, %rcx
    call    swap
    jmp     .L5
.L9:
    movl    -4(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rdx, %rax
    movss   (%rax), %xmm1
    movl    -20(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rdx, %rax
    movss   (%rax), %xmm0
    comiss  %xmm1, %xmm0
    jbe     .L12
    movl    -4(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rdx, %rax
    movss   (%rax), %xmm0
    comiss  -16(%rbp), %xmm0
    ja      .L14
.L12:
    movl    -4(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rdx, %rax
    movss   (%rax), %xmm0
    movl    -20(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rdx, %rax
    movss   (%rax), %xmm1

```

```

        comiss %xmm1, %xmm0
        jbe    .L5
        movl   -4(%rbp), %eax
        cltq
        leaq   0(,%rax,4), %rdx
        movq   16(%rbp), %rax
        addq   %rdx, %rax
        movss  (%rax), %xmm1
        movss  -16(%rbp), %xmm0
        comiss %xmm1, %xmm0
        jbe    .L5
.L14:
        movl   -8(%rbp), %eax
        cltq
        leaq   0(,%rax,4), %rdx
        movq   16(%rbp), %rax
        addq   %rax, %rdx
        movl   -4(%rbp), %eax
        cltq
        leaq   0(,%rax,4), %rcx
        movq   16(%rbp), %rax
        addq   %rcx, %rax
        movq   %rax, %rcx
        call   swap
.L5:
        movl   -8(%rbp), %eax
        cltq
        leaq   0(,%rax,4), %rdx
        movq   16(%rbp), %rax
        addq   %rdx, %rax
        movss  (%rax), %xmm0
        movss  %xmm0, -16(%rbp)
        jmp    .L17
.L19:
        addl   $1, -4(%rbp)
.L18:
        movl   -4(%rbp), %eax
        cltq
        leaq   0(,%rax,4), %rdx
        movq   16(%rbp), %rax
        addq   %rdx, %rax
        movss  (%rax), %xmm1
        movss  -16(%rbp), %xmm0
        comiss %xmm1, %xmm0
        ja     .L19
        subl   $1, -8(%rbp)
        jmp    .L20
.L22:
        subl   $1, -8(%rbp)
.L20:
        movl   -4(%rbp), %eax
        cmpl   -8(%rbp), %eax
        jge    .L21
        movl   -8(%rbp), %eax
        cltq
        leaq   0(,%rax,4), %rdx
        movq   16(%rbp), %rax
        addq   %rdx, %rax
        movss  (%rax), %xmm0
        comiss -16(%rbp), %xmm0
        ja     .L22
.L21:

```



```

    movl    -4(%rbp), %eax
    cmpl    -8(%rbp), %eax
    jge     .L17
    movl    -8(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rax, %rdx
    movl    -4(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rcx
    movq    16(%rbp), %rax
    addq    %rcx, %rax
    movq    %rax, %rcx
    call    swap
.L17:
    movl    -4(%rbp), %eax
    cmpl    -8(%rbp), %eax
    jl      .L18
    movl    -12(%rbp), %eax
    cltq
    movl    -192(%rbp,%rax,4), %eax
    movl    %eax, -8(%rbp)
    movl    -8(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rdx
    movq    16(%rbp), %rax
    addq    %rax, %rdx
    movl    -4(%rbp), %eax
    cltq
    leaq    0(,%rax,4), %rcx
    movq    16(%rbp), %rax
    addq    %rcx, %rax
    movq    %rax, %rcx
    call    swap
    movl    -12(%rbp), %eax
    cltq
    movl    -112(%rbp,%rax,4), %eax
    movl    -4(%rbp), %edx
    subl    %eax, %edx
    movl    -12(%rbp), %eax
    cltq
    movl    -192(%rbp,%rax,4), %eax
    subl    -4(%rbp), %eax
    cmpl    %eax, %edx
    jl      .L24
    movl    -12(%rbp), %eax
    leal    1(%rax), %ecx
    movl    -12(%rbp), %eax
    cltq
    movl    -112(%rbp,%rax,4), %edx
    movslq  %ecx, %rax
    movl    %edx, -112(%rbp,%rax,4)
    movl    -12(%rbp), %eax
    leal    1(%rax), %ecx
    movl    -4(%rbp), %eax
    leal    -1(%rax), %edx
    movslq  %ecx, %rax
    movl    %edx, -192(%rbp,%rax,4)
    movl    -4(%rbp), %eax
    leal    1(%rax), %edx
    movl    -12(%rbp), %eax

```

```

        cltq
        movl %edx, -112(%rbp,%rax,4)
        jmp  .L25
.L24:
        movl -12(%rbp), %eax
        leal 1(%rax), %ecx
        movl -4(%rbp), %eax
        leal 1(%rax), %edx
        movslq %ecx, %rax
        movl %edx, -112(%rbp,%rax,4)
        movl -12(%rbp), %eax
        leal 1(%rax), %ecx
        movl -12(%rbp), %eax
        cltq
        movl -192(%rbp,%rax,4), %edx
        movslq %ecx, %rax
        movl %edx, -192(%rbp,%rax,4)
        movl -4(%rbp), %eax
        leal -1(%rax), %edx
        movl -12(%rbp), %eax
        cltq
        movl %edx, -192(%rbp,%rax,4)
.L25:
        addl $1, -12(%rbp)
.L3:
        cmpl $0, -12(%rbp)
        jns  .L26
        nop
        nop
        addq $224, %rsp
        popq %rbp
        ret
        .seh_endproc
        .def __main; .scl 2; .type 32; .endef
        .globl main
        .def main; .scl 2; .type 32; .endef
        .seh_proc main
main:
        pushq %rbp
        .seh_pushreg %rbp
        subq $368, %rsp
        .seh_stackalloc 368
        leaq 128(%rsp), %rbp
        .seh_setframe %rbp, 128
        .seh_endprologue
        call __main
        movl $0, %ecx
        call time
        movl %eax, %ecx
        call srand
        movl $0, 236(%rbp)
        jmp  .L32
.L33:
        call rand
        movslq %eax, %rdx
        imulq $1374389535, %rdx, %rdx
        shrq $32, %rdx
        movl %edx, %ecx
        sarl $5, %ecx
        cltd
        subl %edx, %ecx
        movl %ecx, %edx

```

```

    imull $100, %edx, %edx
    subl %edx, %eax
    movl %eax, %edx
    cvtsi2ssl %edx, %xmm0
    movl 236(%rbp), %eax
    cltq
    movss %xmm0, -96(%rbp,%rax,4)
    addl $1, 236(%rbp)
.L32:
    cmpl $79, 236(%rbp)
    jle .L33
    leaq -96(%rbp), %rax
    movl $80, %edx
    movq %rax, %rcx
    call sort
    movl $0, %eax
    addq $368, %rsp
    popq %rbp
    ret
.seh_endproc

```