Лабораторная работа № 2

“Исследование улучшенных алгоритмов сортировки”

Задание:

1. Разработать проект для исследования алгоритмов сортировки в соответствии с вариантом (диаграмма вариантов использования проекта представлена на рис. 1)

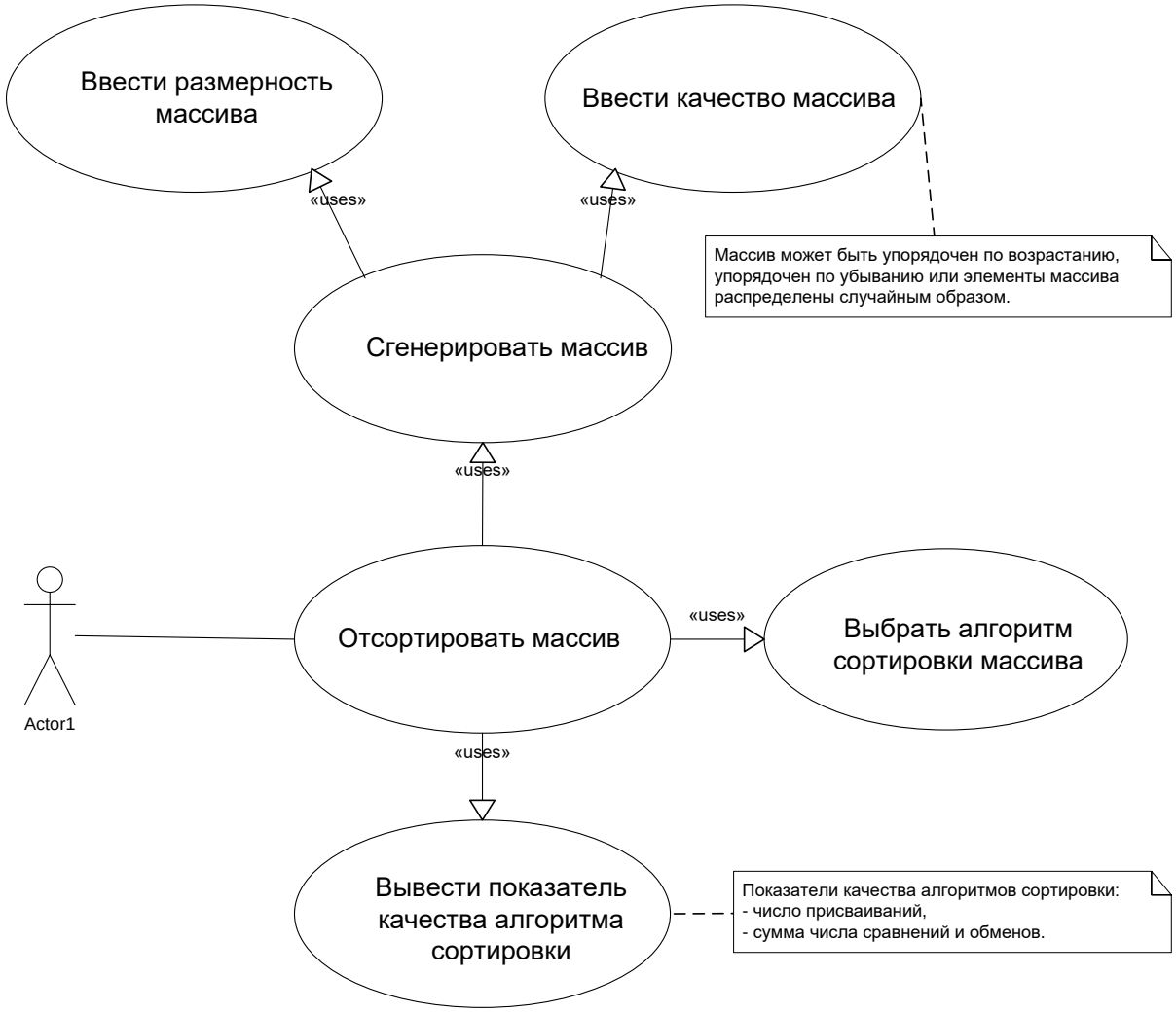


Рис. 1. Диаграмма вариантов использования проекта лабораторной работы № 2.

2. Разработать интерфейс проекта, *позволяющий*:
   - задавать размерность и качество массива;
   - осуществлять выбор алгоритма сортировки для исследования;
   - осуществлять вывод информации о результатах исследования алгоритма сортировки (исходный и отсортированный массив, пошаговую работу алгоритма сортировки (при небольшой размерности массива), показатели качества работы алгоритма сортировки).
   - Создать подпрограмму для генерации целочисленного массива различного качества. В подпрограмме предусмотреть:
   - задание размерности массива;

- задание диапазона чисел массива;
- создание массива с равномерно распределенными случайными числами;
- создание упорядоченного по возрастанию массива;
- создание упорядоченного по убыванию массива.

3. Создать подпрограмму, реализующую алгоритм сортировки в соответствии с вариантом. В подпрограмме *предусмотреть*:
   - определение числа операций алгоритма сортировки;
   - сортировку массива по возрастанию.

Письменный отчет по лабораторной работе должен содержать:

1. Титульный лист. (Название лабораторной работы. Фамилия, имя, отчество, номер группы исполнителя, дата сдачи.)
2. Математическую постановку задачи.
3. Распечатку текстов подпрограмм для генерации массива и для исследования алгоритмов сортировки (обязательны комментарии к программе).
4. Исследование программной реализации улучшенных алгоритмов сортировки, содержащее следующие материалы, таблицы и графики:

- *примеры пошаговой работы* исследуемых алгоритмов сортировки для небольшой размерности задачи (n = 7, диапазон чисел 0..10 – для сортировки подсчетом и 0..100 – для поразрядных сортировок);
- сведенную в *таблицу* зависимость количества операций (или времени выполнения) исследуемых алгоритмов сортировки от размерности задачи n для прямого, обратного и случайного расположения элементов в массиве;
- сведенную в *таблицу* зависимость количества операций (или времени выполнения) исследуемых алгоритмов сортировки от максимального числа исследуемого массива k для случайного расположения элементов в массиве – *для сортировки подсчетом*;
- сведенную в *таблицу* зависимость количества операций (или времени выполнения) исследуемых *поразрядных* алгоритмов сортировки от разрядности чисел массива (p = 1…5) для случайного расположения элементов в массиве;
- сведенную в *таблицу* зависимость количества операций (или времени выполнения) исследуемых алгоритмов сортировки от размерности задачи n и от порядка выбора элемента разбиения для прямого, обратного и случайного расположения элементов в массиве – *для быстрой сортировки* (всего в отчете должно быть 8 таблиц);
- *графики* зависимости показателя качества (количества операций (суммы числа сравнений и обменов) или времени выполнения) исследуемых алгоритмов сортировки от размерности задачи для прямого, обратного и случайного расположения элементов в массиве (5 графиков);

Пример таблицы:

Таблица 1. Быстрая сортировка, массив упорядочен по возрастанию

n	5	10	15	20	25	30	35	40	45	50
Sr										
Ob										
Sum										

- *график* зависимости показателя качества исследуемого алгоритма сортировки подсчетом от максимального числа исследуемого массива *k* для случайного расположения элементов в массиве (1 график);
- *график* зависимости показателя качества исследуемых алгоритмов поразрядной сортировки от разрядности чисел массива *p* для случайного расположения элементов в массиве (1 график);

5. *Выводы* по лабораторной работе (в выводах провести сравнительную характеристику исследованных алгоритмов сортировки).

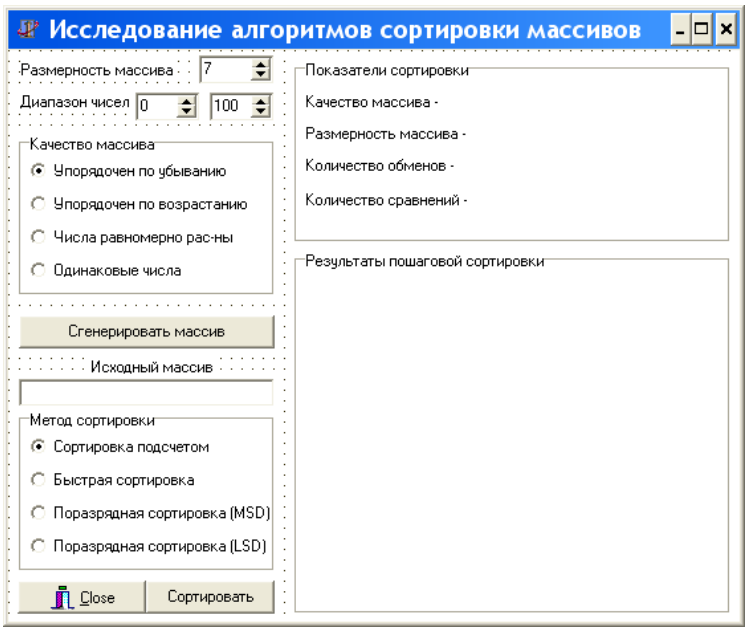
Варианты задач по лабораторной работе:

1. Сортировка подсчетом.
2. Поразрядная сортировка – *LSD*.
3. Поразрядная сортировка – *MSD*.
4. Быстрая сортировка.

Варианты исследуемых алгоритмов сортировки

Вариант	Интервал [*A, B*]	Варианты задач
1.	[0, 100]	1, 2
2.	[0, 200]	1, 3
3.	[0, 0.002]	1, 2
4.	[0.1, 0.003]	1, 3
5.	[0, 500]	1, 2
6.	[100, 200]	1, 3
7.	[0, 0.001]	1, 2
8.	[0, 0.002]	1, 3
9.	[0,300]	1, 2
10.	[0,400]	1, 3
11.	[0.2, 0.005]	1, 2
12.	[0, 0.006]	1, 3
13.	[0,700]	1, 2
14.	[0,800]	1, 3
15.	[0.1, 0.009]	1, 2
16.	[0,999]	1, 3
17.	[1,900]	1, 2
18.	[0, 0.001]	1, 3
19.	[0, 0.001]	1, 2
20.	[10, 1000]	1, 3

Вариант формы проекта при использовании визуальной среды проектирования



Краткие теоретические сведения

1. Сортировка подсчетом.

Пусть исходная последовательность записана в массиве $A[1..n]$, $C[1..k]$ – вспомогательный массив (k – мощность диапазона чисел, используемых в исходном массиве $A[1..n]$), отсортированная последовательность записывается в массив $B[1..n]$.

Counting_Sort (*A, B, k*)

```
1.  for i:=1 to k
2.                  do C[i]:=0
3. for j:=1 to length(A)
4.                  do C[A[j]]:=C[A[j]]+1
5. // C[i]  равно количеству элементов, равных i
6. for i:=2 to k
7.                  do C[i]:=C[i]+C[i-1]
8. // C[i]  равно количеству элементов, не превосходящих i
9. for j:= length(A) downto 1
10.                 do B[C[A[j]]] :=A[j]
11.                     C[A[j]]:= C[A[j]]-1
```

После инициализации (строки 1 - 2) сначала помещают в $C[i]$ количество элементов массива A, равных i (строки 3 - 4), а затем, находя частичные суммы последовательности $C[1]$,$C[2]$, ... , $C[k]$ – количество элементов, не превосходящих i (строки 6 - 7). В строках 9-10 каждый из элементов массива A помещается на нужное место в массиве B. В самом деле, если все n элементов различны, то в отсортированном массиве B число $A[j]$ должно стоять на месте $C[A[j]]$, ибо именно столько элементов массива A не превосходят числа $A[j]$, если в массиве A встречаются повторения, то после каждой записи числа $A[j]$ в массив B число $C[A[j]]$ уменьшается на единицу (строка 11), так что при следующей встрече с числом, равным $A[j]$, оно будет записано на одну позицию левее.

Пример: исходная последовательность A = | 3 | 4 | 6 | 3 | 1 | 4 | 4 | 1 | 2 |, k=6 – мощность чисел в массиве A, размерность массива A – n=9.

C = | 2 | 1 | 2 | 3 | 0 | 1 | , число присваиваний Пр=9, новое значение $C[i]$ равно количеству элементов, непревосходящих i ($C[i]:=C[i]+C[i-1]$),

C = | 2 | 3 | 5 | 8 | 8 | 9 |. Пр=Пр+(k -1)=9+(6-1)=14.

1. элемент отсортированного массива $B[C[A[9]]] = A[9]$, то есть $B[C[2]] = B[3]=A[9] = 2$, значит $B[3] = 2$ (Пр=Пр+1=15), далее $C[A[9]]=C[2]=C[2]-1=3-1=2$ (Пр=Пр+1=16).

C = | 2 | 2 | 5 | 8 | 8 | 9 |

2. $B[C[A[8]]]=B[C[1]]= B[2]=1$, то есть $B[2]=1$, $C[A[8]]=C[1]=C[1]-1=2-1=1$, (Пр=Пр+2=18)

C = | 1 | 2 | 5 | 8 | 8 | 9 |

3. $B[C[A[7]]]=B[C[4]]=B[8]=A[7]=4$, то есть $B[8]=4, C[A[7]]=C[4]=C[4]-1=7$, (Пр=Пр+2=20)

$C =$

1	2	5	7	8	9

4. $B[C[A[6]]] = B[C[4]] = B[7]=A[6]=4$, то есть $B[7]=4$, $C[4]=C[4]-1=6$, (Пр=Пр+2=22)

$C =$

1	2	5	6	8	9

5. $B[C[A[5]]] = B[C[1]] = B[1]=A[5]=1$, то есть $B[1]=1$, $C[1]=C[1]-1=0$, (Пр=Пр+2=24)

$C =$

0	2	5	6	8	9

6. $B[C[A[4]]] = B[C[3]] = B[5]=A[4]=3$, то есть $B[5]=3$, $C[3]=C[3]-1=5-1=4$, (Пр=Пр+2=26)

$C =$

0	2	4	6	8	9

И так далее.

Количество присваиваний для реализации сортировки подсчетом массива A размерности n =9 и при k =6 – Пр=n+(k-1)+n*2=9+(6-1)+2*9=32. Количество сравнений для поиска величины k равно (n - 1)=8. Общее количество операций Оп= n+(k-1)+n*2+(n-1)=4*n+k-2=36+6-2=40.

Таким образом, результирующий массив будет иметь вид

B=

1	2	3	4	5	6	7	8	9
1	1	2	3	3	4	4	4	6

2. Поразрядная сортировка (*LSD*).

LSD (least significant digit radix sort) - поразрядная сортировка сначала по младшей цифре.

Пусть в общем случае сортируемые числа $A=(a_1, a_2, \ldots, a_n)$ являются целыми и состоят из p цифр (более короткие числа дополняются нулями) с основанием системы счисления r. Число a_i имеет следующее представление: $a_i=(a_{i,p}, a_{i,p-1}, \ldots, a_{i,1})$, где $a_{i,p}$ – цифра старшего разряда, $a_{i,1}$ – цифра младшего разряда числа a_i. Поразрядная сортировка основана на том свойстве, что числа можно полностью отсортировать, выполняя сортировку по отдельным разрядам, начиная с самого младшего.

Алгоритм поразрядной сортировки состоит в следующем:

1) k=1. Выбираем младшую цифру числа.

2) Берем каждое число из массива A и помещаем его в конец одной из r очередей в зависимости от значений цифры в позиции k.

3) Восстанавливаем каждую очередь в массив A, начиная с очереди чисел с цифрой 0 и кончая очередью чисел с (r-1)-й цифрой. (После распределения массива A по очередям (по “карманам”) выполняется операция конкатенации всех списков в один список.)

4) $k=k+1$. Выполняем пункты 2, 3 пока ($k \le p$), то есть до старшей значащей цифры числа.

(При k=1 помещаем каждое сортируемое число a_i в очередь с номером (a_i *mod* r). При k=2 число помещается в очередь с номером $\lfloor a_i / r \rfloor$, то есть номер очереди

равен наибольшему целому числу, равному или меньшему $\lfloor a_i / r \rfloor$, (другими словами, скобки здесь обозначают операцию взятия целой части числа).)

Процесс поразрядной сортировки для последовательности A=(25, 57, 48, 37, 12, 92, 86, 33) показан в таблице 1 (здесь r =10, p =2, n =8).

Сложность поразрядной сортировки составляет примерно $O(p \cdot n)$.

Для экономии памяти эффективно использовать линейные связанные списки из указателей для представления сортируемых элементов и очередей.

Таблица 1 Пример поразрядной сортировки (*LSD)*

Очереди для k=1		*Очереди для k=2*	
Разряд	Содержимое разряда	Разряд	Содержимое разряда
0		0	
1		1	12
2	12 92	2	25
3	33	3	33 37
4		4	48
5	25	5	57
6	86	6	
7	57 37	7	
8	48	8	86
9		9	92
Восстанавливаем массив *A* {12, 92, 33, 25, 86, 57, 37, 48}		Восстанавливаем массив *A* {12, 25, 33, 37, 48, 57, 86, 92}	

Чтобы увидеть, почему этот алгоритм работает правильно, достаточно заметить, что когда числа помещаются в одну очередь (в один “карманам”), например числа 33 и 37 в карман 3, то они будут располагаться в возрастающем порядке, поскольку в списке {12, 92, 33, 25, 86, 57, 37, 48} они упорядочены по самой правой цифре (k=1). Следовательно, в любом “кармане” числа также будут упорядочены по самой правой цифре. И, конечно, распределение чисел на втором этапе по “карманам” в соответствии с первой цифрой (k=2 в примере) гарантирует, что в объединенном списке все числа будут расставлены в возрастающем порядке.

1. Количество присваиваний при формировании очередей для k=1 равно Оп=n=8.

2. Количество операций при восстановлении массива равно r=10, то есть Оп=n+r=18.

3. Количество присваиваний при формировании очередей для k=2 равно n=8, то есть Оп=n+r+n=26.

4. Количество операций при восстановлении массива равно r=10, то есть Оп = n + r + n + r =36.

3. Поразрядная сортировка (*MSD*).

MSD (*most significant digit radix sort)* - поразрядная сортировка сначала по старшей цифре**.**

Первая часть процесса поразрядной сортировки *MSD* (- проход по старшей цифре ключа) для последовательности *A*=(25, 57, 48, 37, 12, 92, 86, 33) показана в табл. 2 (здесь *r* =10, *p* =2, *n* =8).

Таблица 2. Пример поразрядной сортировки (*MSD)*

Очереди для k=1	
Разряд	*Содержимое разряда*
(0)	
(1)	12
(2)	25
(3)	37 33
(4)	48
(5)	57
(6)	
(7)	
(8)	86
(9)	92
Выполнить операцию конкатенации списков. Результат имеет вид: {12, 25, 37, 33, 48, 57, 86, 92}	

Массив *A* практически отсортирован, за исключением двух чисел {37, 33}, которые можно отсортировать любым простым методом сортировки.

Отсортировать числа в разряде 3 с помощью, например, сортировки простым выбором.

Подсчет количества операций:

1. Количество присваиваний при формировании очередей для старшего разряда равно Пр=n=8. Массив A имеет вид {12, 25, 37, 33, 48, 57, 86, 92}, то есть неотсортированы числа в третьем разряде.

2. Количество операций (сравнений и присваиваний) при сортировке, например, простым выбором подмассива (37, 33) равно – сравнений 1, обменов 1 (то есть присваиваний 1*3=3). Всего операций (сравнений и присваиваний) – 1+3=4.

Всего операций при поразрядной сортировке MSD массива A=(25, 57, 48, 37, 12, 92, 86, 33) равно Пр=8+4=12.

4. Быстрая сортировка.

Для достижения наибольшей эффективности при сортировке Ч.Хоар предложил обеспечить обмен элементов на больших расстояниях.

Идея быстрой сортировки Ч. Хоара такова: в последовательности сортируемых элементов ($a_1,\ldots,a_n$) выберем наугад какой-нибудь элемент (назовем его *X*); будем просматривать слева нашу последовательность до тех пор, пока не найдем элемент $a_i>X$; потом будем просматривать последовательность справа, пока не встретим $a_j<X$; поменяем местами эти два элемента и продолжим процесс просмотра и обмена, пока

эти два просмотра не встретятся в середине последовательности. В результате этого последовательность будет разбита на левую часть, которая имеет ключи меньшие (или равные) *X*, и правую – с ключами большими (или равными) *X*. Сортировку от распределения отделяет только небольшой шаг: необходимо применить этот процесс разбиения к получившимся двум частям, потом к частям частей, и так до тех пор, пока каждая из частей не будет состоять из одного элемента.

Описание алгоритма быстрой сортировки имеет вид:

1. В последовательности элементов $A=(a_1,\ldots,a_n)$ выбрать элемент *X*.
2. Просматривая последовательность *A* слева направо, найти элемент $a_i < X$.
3. Просматривая последовательность *A* справа налево, найти элемент $a_к >= X$.
4. Поменять местами элементы a_i и a_k. Продолжить процесс встречного просмотра и обмена, пока два просмотра не встретятся где-то внутри последовательности элементов *A*. Если *down*>=*up*, то поменять местами *X* и a_{up}, где *down* – текущий индекс при движении слева направо,
5. а *up* - текущий индекс при движении справа налево. В результате элемент *X* помещается в позицию j и выполнятся следующие условия: - элементы в позициях с 1-й по (*j*-1)-ю меньше (равны) *X*, - элементы в позициях с (*j*+1)-й по *n*-ю, больше (равны) *X*. То есть элемент *X* является *j*-м наименьшим элементом в последовательности *A* и *X* останется в позиции j и когда последовательность *A* будет полностью отсортирована.
6. Применить пункты 1,..,5 к подпоследовательностям ($a_1,\ldots, a_j$) ($a_{j-1},\ldots, a_n$) и так далее, пока каждая из подпоследовательностей не будет состоять из одного элемента.

Первая часть процесса быстрой сортировки для последовательности *A*=(11, 7, 4, 49, 9, 18, 2, 5, 11) показана в табл. 3, в качестве элемента разбиения *X* выбран элемент a_5.

Таблица 3 Пример быстрой сортировки

Шаг	*Состояние последовательности*								
1.	11	7	4	49	9	18	2	5	11
2.	5	7	4	49	9	18	2	11	11
3.	5	7	4	2	9	18	49	11	11

В примере $X=a_5=9$. Так как $a_1=11>X$, а $a_8=5<X$, то они меняются местами, что и показано во второй строке таблицы 1. Так как $a_4=49>X$, а $a_7=2<X$, то они тоже меняются местами, что и показано в третьей строке таблицы 1. Третья строка таблицы 1 также показывает, что элемент $X=a_5=9$ находится в своем окончательном месте и разделяет последовательность *A* на две подпоследовательности: с элементами меньше (равно) *X* – (5, 7, 4, 2) и с элементами больше (равно) *X* – (18, 49, 11, 11). Далее процесс сортировки необходимо аналогично применить к этим подпоследовательностям.

Для выбора элемента разбиения *X* можно использовать:

1. Первый элемент последовательности.
2. Последний элемент последовательности.
3. Элемент находящийся посередине последовательности.
4. Медиану последовательности.
5. Медиану среди элементов (a_1, $a_{n/2}$, a_n).
6. Генератор случайных чисел.

Для алгоритма быстрой сортировки имеем следующие выводы:

1. быстрая сортировка работает наилучшим образом для полностью неотсортированных последовательностей;
2. лучший выбор элемента разделения – медиана последовательности;

3. быстрая сортировка работает наихудшим образом для полностью отсортированных последовательностей при выборе в качестве элемента разбиения наименьшего (наибольшего) значения;

4. в среднем (по всем последовательностям размера n) быстрая сортировка имеет сложность $O(n \cdot log_2 n)$;

5. в наихудшем случае быстрая сортировка имеет сложность $O(n^2)$;

6. быстрая сортировка является одним из наилучших алгоритмов сортировки (за счет наименьшей мультипликативной константы C_Q по сравнению с пирамидальной сортировкой).

Пример. Пошаговая работа алгоритма быстрой сортировки (первое разделение последовательности).

```
X=a(lb)      // X - элемент разделения, для него ищется место окончательной позиции
Up=rb        // Up – правая граница массива
Down=lb      // Down – левая граница массива
While Down< Up do
                    While a(Down)<= X do
                                Down= Down+1
                    End
                    While a(Up)> X do
                                Up= Up-1
                    End
                    If Down< Up then
                            поменять местами a(Down) и a(Up)
                    End If
End
a(lb)= a(Up) // a(Up) меняется местами с a(lb)= X
a(Up)= X
j= Up            // j – окончательная позиция элемента разделения X
```

Пусть имеется массив размерности n=9:

1	2	3	4	5	6	7	8	9
11	7	2	4	18	9	49	5	11

Проверим условие алгоритма: при движении слева-направо – "<=", справа-налево – ">".

X=a(1)=11
Up=n=9
Down=1
a(1)=11<=X=11 Да → Down= Down +1=2
a(2)=7<=X=11 Да → Down= Down +1=3
a(3)=2<=X=11 Да → Down= Down +1=4
a(4)=4<=X=11 Да → Down= Down +1=5
a(5)=18<=X=11 Нет → a(9)=11>X=11 → Нет
Down=5< Up=9 Да → обмен a(5) и a(9)
Получили 11, 7, 2, 4, 11, 9, 49, 5, 18
a(5)=11<=X=11 Да → Down= Down +1=6
a(6)=9<=X=11 Да → Down= Down +1=7
a(7)=49<=X=11 Нет → a(9)=18>X=11 Да → Up= Up-1=8
a(8)=5>X=11 Нет

Down=7< Up =8 Да → обмен a(7)=49 и a(8)=5
Получили 11, 7, 2, 4, 11, 9, 5, 49, 18
a(8)=49<=X=11 Нет → a(8)=49>X=11 Да→ Up= Up-1=7
a(7)=5>X=11 Нет
Down=7< Up =7 Нет → обмен X=a(1)=11 с a(7)=5
Получили (5, 7, 2, 4, 11, 9) (**11**) (49, 18) ← итоговая последовательность.

Элемент разделения X=a(1)=11 занял окончательное место j=7 в почти что отсортированной последовательности, исходная последовательность разделилась на две неравные части (плохая сбалансированность) – первая с элементами <= X=11, вторая – с элементами > X=11.

Список литературы

1. Лэнгсам И., Огенстайн М., Тененбаум А. Структуры данных для персональных ЭВМ. -М.: Мир, 1989.-568 с.(с.437-465).
2. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. -М.: МЦНМО, 2000. - 960 с. (с. 175 – 177, с.171 – 173)
3. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/Поиск. - К.: ДиаСофт, 2001. – 688 с. (с. 295-298, 401 - 437)
4. Ахо А., Хопкрофт Дж., Ульман Д. Структуры данных и алгоритмы. -М.: Вильямс, 2000. – 384 с. (с. 247 - 254)
5. Проценко В.С. та ін. Техніка програмування мовою Сі: Навч. Посібник,-К.:Либідь, 1993.-224 с.(с.72-73).
6. Мейер Б., Бодуэн К. Методы программирования Т.2.-М.:Мир, 1982.-368 с.(с.153-183).
7. Зубов В.С. Справочник программиста. Базовые методы решения графовых задач и сортировки.-М.: Филинъ, 1999.-256 с.(с.55-62).
8. Кнут Д. Искусство программирования для ЭВМ. Т.3. Сортировка и поиск.-М.:Мир, 1978.-844 с.(с.140-151,177-190).
9. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов.-М.:Мир, 1981.-(с.223-240).

Контрольные вопросы и задания

1. В чем состоят задача и цель сортировки?
2. Приведите классификацию алгоритмов внутренней сортировки.
3. Каковы основные показатели качества исследуемых алгоритмов сортировки?
4. Отсортируйте пошагово по возрастанию массив целых чисел размерности 6 с помощью быстрой сортировки, сортировки подсчетом, поразрядной сортировки (*LSD* и *MSD*).
5. Каковы сложности алгоритмов быстрой сортировки, сортировки подсчетом, поразрядной сортировки?
6. Как задается качество массива для исследования алгоритмов сортировки - быстрой сортировки, сортировки подсчетом, поразрядной сортировки?