+МИНИСТЕРСТВО ТРАНСПОРТА И КОММУНИКАЦИЙ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Белорусский государственный университет транспорта»

Кафедра «Информационно-управляющие системы и технологии»

Отчет  
по практическим работам  
по дисциплине «Основы алгоритмизации и программирования»

Выполнил Проверил

студент группы ГИ-11 ст. преп. каф. ИУСиТ

Зайцев И. С. Голдобина Т. А.

Гомель, 2020

Содержание

[Практическая работа №13 АЛГОРИТМЫ СОРТИРОВКИ 3](#_Toc34586561)

[Задания 3](#_Toc34586562)

[Контрольные вопросы 24](#_Toc34586563)

# Практическая работа №13 АЛГОРИТМЫ СОРТИРОВКИ

### **Цель**

### Разработать алгоритмы сортировки.

## Задания

### Условие

Разработать проект, включающий главную функцию, содержащую меню, и функции-подпрограммы:

1. Ввода или генерации одномерного массива. Количество элементов - не менее 10. Диапазон - на выбор. Вариативность оценивается (тип и диапазон значений массива, количество элементов).
2. Функция, позволяющая поменять местами значения двух чисел (не включается в меню) swap(a, b)
3. Функция сортировки пузырьком
4. Функция сортировки выбором
5. Функция сортировки вставками
6. Функция быстрой сортировки
7. Функция пирамидальной сортировки
8. Функция шейкер-сортировки
9. Функция сортировки Шелла
10. Функция сортировки слиянием
11. Функция сортировки подсчетом
12. Другие функции сортировки (3)
13. Функция вывода одномерного массива

**Программный код**

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

#include <malloc.h>

#include <conio.h>

void display(int num[11]) // Вывод одномерного массива

{

int i;

for (i=0; i<10; i++) {

num[i] = rand() % 201-100;

printf("%d ", num[i]);

}

}

void swap(int \*x, int \*y) // Замена чисел

{int c;

c=\*x;

\*x=\*y;

\*y=c;

}

void bubble(int \*num, int size) // Сортировка пузырьком

{

for (int i=0; i<size-1; i++)

{

for (int j=(size-1); j>i; j--)

{

if (num[j-1]>num[j])

{

int temp=num[j-1];

num[j-1]=num[j];

num[j]=temp;

}

}

}

}

void selectionSort(int \*num, int size) // Сортировка выбором

{

int min, temp;

for (int i = 0; i < 10 - 1; i++)

{

min = i;

for (int j = i + 1; j <10; j++)

{

if (num[j] < num[min])

min = j;

}

temp = num[i];

num[i] = num[min];

num[min] = temp;

}

}

void insertsort(int \*num, int size) // Сортировка вставками

{

int i, j, x;

for (i=0; i<size; i++)

{

x = num[i];

for (j=i-1; j>=0 && num[j]>x; j--) num[j+1] = num[j];

num[j+1] = x;

}

}

void quickSort(int \*numbers, int left, int right) // Быстрая сортировка

{

int pivot;

int l\_hold = left;

int r\_hold = right;

pivot = numbers[left];

while (left < right)

{

while ((numbers[right] >= pivot) && (left < right))

right--;

if (left != right)

{

numbers[left] = numbers[right];

left++;

}

while ((numbers[left] <= pivot) && (left < right))

left++;

if (left != right)

{

numbers[right] = numbers[left];

right--;

}

}

numbers[left] = pivot;

pivot = left;

left = l\_hold;

right = r\_hold;

if (left < pivot)

quickSort(numbers, left, pivot - 1);

if (right > pivot)

quickSort(numbers, pivot + 1, right);

}

void siftDown(int \*numbers, int root, int bottom) // Пирамидальная сортировка

{

int maxChild;

int done = 0;

while ((root \* 2 <= bottom) && (!done))

{

if (root \* 2 == bottom)

maxChild = root \* 2;

else if (numbers[root \* 2] > numbers[root \* 2 + 1])

maxChild = root \* 2;

else

maxChild = root \* 2 + 1;

if (numbers[root] < numbers[maxChild])

{

int temp = numbers[root];

numbers[root] = numbers[maxChild];

numbers[maxChild] = temp;

root = maxChild;

}

else

done = 1;

}

}

void heapSort(int \*numbers, int array\_size) // Пирамидальная сортировка

{

for (int i = (array\_size / 2) - 1; i >= 0; i--)

siftDown(numbers, i, array\_size - 1);

for (int i = array\_size - 1; i >= 1; i--)

{

int temp = numbers[0];

numbers[0] = numbers[i];

numbers[i] = temp;

siftDown(numbers, 0, i - 1);

}

}

void shaker(int \*num, int size) // Шейкер сортировка

{

int tmp = 0, high = 0, low = (10 - 1);

while (high < low)

{

for (int i = high; i < low; i++)

{

if (num[i] > num[i + 1])

{

tmp = num[i];

num[i] = num[i + 1];

num[i + 1] = tmp;

}

}

low--;

for (int j = low; j > high; j--)

{

if (num[j] < num[j - 1])

{

tmp = num[j];

num[j] = num[j - 1];

num[j - 1] = tmp;

}

}

high++;

}

}

void shellSort(int \*num, int size) // Сортировка Шелла

{

int increment = 3;

while (increment > 0)

{

for (int i=0; i<10; i++)

{

int j = i;

int temp = num[i];

while ((j >= increment) && (num[j - increment] > temp))

{

num[j] = num[j - increment];

j = j - increment;

}

num[j] = temp;

}

if (increment > 1)

increment = increment / 2;

else if (increment == 1)

break;

}

}

void mergeSort(int \*num, int size) // Сортировка слиянием

{

int step = 1;

int \*temp = (int\*)malloc(size \* sizeof(temp));

while (step < size)

{

int index = 0;

int l = 0;

int m = l + step;

int r = l + step \* 2;

do

{

m = m < size ? m : size;

r = r < size ? r : size;

int i1 = l, i2 = m;

for (; i1 < m && i2 < r; )

{

if (num[i1] < num[i2]) { temp[index++] = num[i1++]; }

else { temp[index++] = num[i2++]; }

}

while (i1 < m) temp[index++] = num[i1++];

while (i2 < r) temp[index++] = num[i2++];

l += step \* 2;

m += step \* 2;

r += step \* 2;

} while (l < size);

for (int i = 0; i < size; i++)

num[i] = temp[i];

step \*= 2;

}

}

int main ()

{

int t[11], i, a, b, \*x, \*y, variant, N=0;

setlocale(LC\_ALL,"");

x=&a;

y=&b;

display(t);

printf("\nВведите номер задания: ");

scanf("%d", &variant);

switch(variant){

case 3: printf("Функция сортировки пузырьком:\n");{

bubble(t, 10);

for (int i = 0; i<10; i++)

printf("%d ", t[i]);}

break;

case 4: printf("Функция сортировки выбором:\n"); {

selectionSort(t, 10);

for (int i = 0; i<10; i++)

printf("%d ", t[i]);}

break;

case 5: printf("Функция сортировки вставками:\n");{

insertsort(t, 10);

for (int i = 0; i<10; i++)

printf("%d ", t[i]);}

break;

case 6: printf("Функция быстрой сортировки:\n");

quickSort(t, 0, 10-1);

for (int i = 0; i<10; i++)

printf("%d ", t[i]);

break;

case 7: printf("Функция пирамидальной сортировки:\n");

heapSort(t, 10);

for (int i = 0; i<10; i++)

printf("%d ", t[i]);

break;

case 8: printf("Функция шейкер-сортировки:\n");

shaker(t, 10);

for (int i = 0; i<10; i++)

printf("%d ", t[i]);

break;

case 9: printf("Функция сортировки Шелла:\n");

shellSort(t, 10);

for (int i = 0; i<10; i++)

printf("%d ", t[i]);

break;

case 10: printf("Функция сортировки слиянием:\n");

mergeSort(t, 10);

for (int i = 0; i<10; i++)

printf("%d ", t[i]);

break;

case 11: printf("Не найдено!\n"); break;

case 12: printf("Не найдено!\n"); break;

case 13: printf("Функция вывода одномерного массива:\n");

display(t);

break;

}

printf("\n");

printf("\na=");

scanf("%d", &a);

printf("b=");

scanf("%d", &b);

swap(&a, &b);

printf("a=%d, b=%d", a, b);

return 0;

}

### **Задание 1.2**

### Условие

Функция, позволяющая поменять местами значения двух чисел (не включается в меню) swap(a, b)

### Программный код

void swap(int \*x, int \*y) // Замена чисел

{int c;

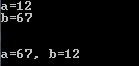
c=\*x;

\*x=\*y;

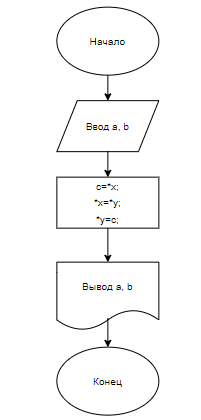
\*y=c;

printf(“a=%d, b=%d”, \*x, \*y);

}

  
Рисунок 1 – Решение задания 1.2

**Блок-схема:**



### **Задание 1.3**

### Условие

Функция сортировки пузырьком

### Программный код

void bubble(int \*num, int size) // Сортировка пузырьком

{

for (int i=0; i<size-1; i++)

{

for (int j=(size-1); j>i; j--)

{

if (num[j-1]>num[j])

{

int temp=num[j-1];

num[j-1]=num[j];

num[j]=temp;

}

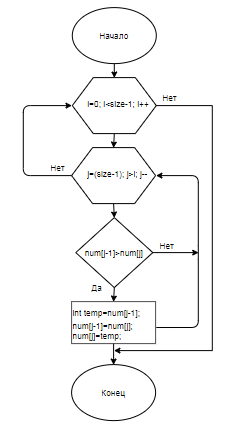
}

}

}

  
Рисунок 1 – Решение задания 1.3

**Блок-схема:**



### **Задание 1.4**

### Условие

Функция сортировки выбором

### Программный код

void selection(int \*num, int size) // Сортировка выбором

{

int min, temp;

for (int i=0; i<size-1; i++)

{

min = i;

for (int j=i+1; j<size; j++)

{

if (num[j]<num[min])

min=j;

temp=num[i];

num[i]=num[min];

num[min]=temp;

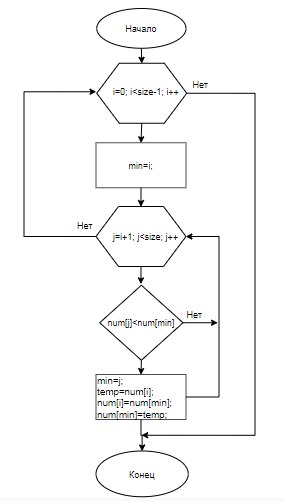
}

}

}

  
Рисунок 1 – Решение задания 1.4

**Блок-схема:**



### **Задание 1.5**

### Условие

Функция сортировки вставками

### Программный код

void insertsort(int \*num, int size) // Сортировка вставками

{

int i, j, x;

for (i=0; i<size; i++)

{

x = num[i];

for (j=i-1; j>=0 && num[j]>x; j--) num[j+1] = num[j];

num[j+1] = x;

}

}

  
Рисунок 1 – Решение задания 1.5

**Блок-схема:**

### 

### **Задание 1.6**

### Условие

Функция быстрой сортировки

### Программный код

void quickSort(int \*numbers, int left, int right)

// Быстрая сортировка

{

int pivot;

int l\_hold = left;

int r\_hold = right;

pivot = numbers[left];

while (left < right)

{

while ((numbers[right] >= pivot) && (left < right))

right--;

if (left != right)

{

numbers[left] = numbers[right];

left++;

}

while ((numbers[left] <= pivot) && (left < right))

left++;

if (left != right)

{

numbers[right] = numbers[left];

right--;

}

}

numbers[left] = pivot;

pivot = left;

left = l\_hold;

right = r\_hold;

if (left < pivot)

quickSort(numbers, left, pivot - 1);

if (right > pivot)

quickSort(numbers, pivot + 1, right);

}

  
Рисунок 1 – Решение задания 1.6

### **Задание 1.7**

### Условие

Функция пирамидальной сортировки

### Программный код

void siftDown(int \*numbers, int root, int bottom) // Пирамидальная сортировка

{

int maxChild;

int done = 0;

while ((root \* 2 <= bottom) && (!done))

{

if (root \* 2 == bottom)

maxChild = root \* 2;

else if (numbers[root \* 2] > numbers[root \* 2 + 1])

maxChild = root \* 2;

else

maxChild = root \* 2 + 1;

if (numbers[root] < numbers[maxChild])

{

int temp = numbers[root];

numbers[root] = numbers[maxChild];

numbers[maxChild] = temp;

root = maxChild;

}

else

done = 1;

}

}

void heapSort(int \*numbers, int array\_size) // Пирамидальная сортировка

{

for (int i = (array\_size / 2) - 1; i >= 0; i--)

siftDown(numbers, i, array\_size - 1);

for (int i = array\_size - 1; i >= 1; i--)

{

int temp = numbers[0];

numbers[0] = numbers[i];

numbers[i] = temp;

siftDown(numbers, 0, i - 1);

}

}

  
Рисунок 1 – Решение задания 1.7

**Блок-схема:**

### 

### **Задание 1.8**

### Условие

Функция шейкер-сортировки

### Программный код

void shaker(int \*num, int size) // Шейкер сортировка

{

int tmp = 0, high = 0, low = (10 - 1);

while (high < low)

{

for (int i = high; i < low; i++)

{

if (num[i] > num[i + 1])

{

tmp = num[i];

num[i] = num[i + 1];

num[i + 1] = tmp;

}

}

low--;

for (int j = low; j > high; j--)

{

if (num[j] < num[j - 1])

{

tmp = num[j];

num[j] = num[j - 1];

num[j - 1] = tmp;

}

}

high++;

}

}

  
Рисунок 1 – Решение задания 1.8

**Блок-схема:**

### 

### **Задание 1.9**

### Условие

Функция сортировки Шелла

### Программный код

void shellSort(int \*num, int size) // Сортировка Шелла

{

int increment = 3;

while (increment > 0)

{

for (int i=0; i<10; i++)

{

int j = i;

int temp = num[i];

while ((j >= increment) && (num[j - increment] > temp))

{

num[j] = num[j - increment];

j = j - increment;

}

num[j] = temp;

}

if (increment > 1)

increment = increment / 2;

else if (increment == 1)

break;

}

}

  
Рисунок 1 – Решение задания 1.9

**Блок-схема:**

### 

### **Задание 1.10**

### Условие

Функция сортировки слиянием

### Программный код

void mergeSort(int \*num, int size) // Сортировка слиянием

{

int step = 1;

int \*temp = (int\*)malloc(size \* sizeof(temp));

while (step < size)

{

int index = 0;

int l = 0;

int m = l + step;

int r = l + step \* 2;

{

m = m < size ? m : size;

r = r < size ? r : size;

int i1 = l, i2 = m;

for (; i1 < m && i2 < r; )

{

if (num[i1] < num[i2]) { temp[index++] = num[i1++]; }

else { temp[index++] = num[i2++]; }

}

while (i1 < m) temp[index++] = num[i1++];

while (i2 < r) temp[index++] = num[i2++];

l += step \* 2;

m += step \* 2;

r += step \* 2;

} while (l < size);

for (int i = 0; i < size; i++)

num[i] = temp[i];

step \*= 2;

}

}

  
Рисунок 1 – Решение задания 1.10

**Блок-схема:**

### 

### **Задание 1.13**

### Условие

Функция вывода одномерного массива

**Программный код**

void display(int num[11]) // Вывод одномерного массива

{

int i;

for (i=0; i<10; i++) {

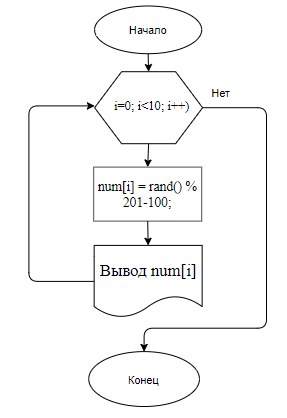
num[i] = rand() % 201-100;

printf("%d ", num[i]);

}}

  
Рисунок 1 – Решение задания 1.13

**Блок-схема:**



## Контрольные вопросы

1. Показатели эффективности алгоритмов – время выполнения программы и требуемый объём памяти.

От чего зависит время выполнения алгоритмов?

1. От сложности алгоритмов, от количества обрабатываемых данных.
2. Асимптотический анализ сложности алгоритмов

Это Оценка времени исполнения алгоритмов. О(n) - n операций, О(n²) - n² операций

1. Сложность алгоритма и понятие О-нотации

Запись вида f(n) = O(g(n)) означает, что ф-ия f(n) возрастает медленнее чем ф-ия g(n) при с = с1 и n = N, где c1 и N могут быть сколь угодно большими числами, т.е. При c = c1 и n >= N, c\*g(n) >=f(n).

О-означает верхнее ограничение сложности алгоритма.

1. Классы сложности

f(n) = O(1) константа  
f(n) = O(log(n)) логарифмический рост  
f(n) = O(n) линейный рост  
f(n) = O(n\*log(n)) квазилинейный рост  
f(n) = O(n^m) полиномиальный рост  
f(n) = O(2^n) экспоненциальный рост

1. Описание алгоритма сортировки выбором

В неотсортированном массиве ищется локальный максимум (минимум).

Найденный максимум (минимум) меняется местами с последним (первым) элементом в массиве.

Если в массиве остались неотсортированные элементы — повторяем действия.

1. На 6-10. Описание алгоритма сортировки быстрой

Из массива выбирается некоторый опорный элемент a[i],

запускается процедура разделения массива, которая перемещает все ключи, меньшие, либо равные a[i], влево от него, а все ключи, большие, либо равные a[i] - вправо,

теперь массив состоит из двух подмножеств, причем левое меньше, либо равно правого,

для обоих подмассивов: если в подмассиве более двух элементов, рекурсивно запускаем для него ту же процедуру.

1. На 8-10. Описание алгоритма сортировки шейкер

От последней перестановки до конца (начала) массива находятся отсортированные элементы. Учитывая данный факт, просмотр осуществляется не до конца (начала) массива, а до конкретной позиции. Границы сортируемой части массива сдвигаются на 1 позицию на каждой итерации.

Массив просматривается поочередно справа налево и слева направо.

Просмотр массива осуществляется до тех пор, пока все элементы не встанут в порядке возрастания (убывания).

Количество просмотров элементов массива определяется моментом упорядочивания его элементов.

1. На 9-10. Описание алгоритма сортировки слиянием

Разделим массив пополам, рекурсивно отсортируем части, после чего выполним процедуру слияния: поддерживаем два указателя, один на текущий элемент первой части, второй – на текущий элемент второй части. Из этих двух элементов выбираем минимальный, вставляем в ответ и сдвигаем указатель, соответствующий минимуму.

1. На 10. Описание алгоритма сортировки на выбор (отличной от предыдущих)

Сортировка пузырьком:

Будем идти по массиву слева направо. Если текущий элемент больше следующего, меняем их местами. Делаем так, пока массив не будет отсортирован. Заметим, что после первой итерации самый большой элемент будет находиться в конце массива, на правильном месте. После двух итераций на правильном месте будут стоять два наибольших элемента, и так далее.

Вывод по работе

Разработал алгоритмы сортировки.