## Практическая работа №2

**«Определение сложности алгоритмов»**

**Цель работы:** Научиться оценивать временную сложность алгоритмов

**Задания для практического занятия:**

**Оцените сложность следующих алгоритмов сортировки и сделайте сравнительный анализ:**

1. сортировка пузырьковым методом;  
   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. Очевидно, не более чем после n итераций массив будет отсортирован. Таким образом, асимптотика в худшем и среднем случае – O(n2), в лучшем случае – O(n).

void bubblesort(int\* l, int\* r) {

int sz = r - l;

if (sz <= 1) return;

bool b = true;

while (b) {

b = false;

for (int\* i = l; i + 1 < r; i++) {

if (\*i > \*(i + 1)) {

swap(\*i, \*(i + 1));

b = true;

}

}

r--;

}

}

1. сортировка вставкой;
2. сортировка посредством выбора;
3. сортировка Хоара;

Алгоритм быстрой сортировки является рекурсивным, поэтому для простоты процедура на вход будет принимать границы участка массива от l включительно и до r не включительно. Понятно, что для того, чтобы отсортировать весь массив, в качестве параметра l надо передать 0, а в качестве r — n, где по традиции n обозначает длину массива.  
  
В основе алгоритма быстрой сортировке лежит процедура partition. Partition выбирает некоторый элемент массива и переставляет элементы участка массива таким образом, чтобы массив разбился на 2 части: левая часть содержит элементы, которые меньше этого элемента, а правая часть содержит элементы, которые больше или равны этого элемента. Такой разделяющий элемент называется **пивотом**.

**int** partition(a: **T**[n], **int** l, **int** r)

**T** v = a[(l + r) / 2]

**int** i = l

**int** j = r

**while** (i ⩽⩽ j)

**while** (a[i] < v)

i++

**while** (a[j] > v)

j--

**if** (i ⩾⩾ j)

**break**

swap(a[i++], a[j--])

**return** j

1. сортировка Шелла.

Сначала сравниваются и сортируются между собой значения, стоящие друг от друга на некотором расстоянии - **d**. После этого расстояние **d** уменьшается и процедура повторяется до тех пор, пока значение **d** не станет минимальным, т.е. **d = 1**. Это означает, что сортировка достигла последнего шага. А на последнем шага элементы сортируются обычной сортировкой вставками.

Первоначально было предложено расчитывать расстояние между сравниваемыми элементами следующим образом:

1. первая итерация - d1 = N/2, где N - размер массива;
2. последующие итерации - di = di-1/2;
3. последняя итерация - dk = 1

program ShellSorting;

uses crt;

type massiv=array[1..100] of integer;

var i, j, n, d, count: integer;

A: massiv;

procedure Shell(A: massiv; n: integer); {сортировка Шелла}

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

count:=A[j];

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

A[j+d]:=count;

j:=j-1;

end;

end;

d:=d div 2;

end;

writeln;

for i:=1 to n do write(' ', A[i]); {вывод массива}

end;

{основной блок программы}

begin

write('Размер массива > '); read(n);

for i:=1 to n do {ввод массива}

begin write(i, ' элемент > '); readln(A[i]); end;

write('Результирующий массив: ');

Shell(A, n);

end.

**Контрольные вопросы:**

1. Как оценивается сложность алгоритма?

Ответ: различают два вида сложности алгоритма: временной сложностью алгоритма и объемом-сложностью алгоритма.

1. Что называют временной сложностью алгоритма?

Ответ: это функция от размера входных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера.

1. Что подразумевают под объемом-сложностью алгоритма?

Ответ: это функция от размера входных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера.

1. Каким образом оценивается временная сложность алгоритма?

Ответ: если обозначить значение пара­метра временной сложности алгоритма α символом Tα, а буквой V обозначить некоторый числовой параметр, характеризующий ис­ходные данные, то временную сложность можно представить как функцию Tα (V).