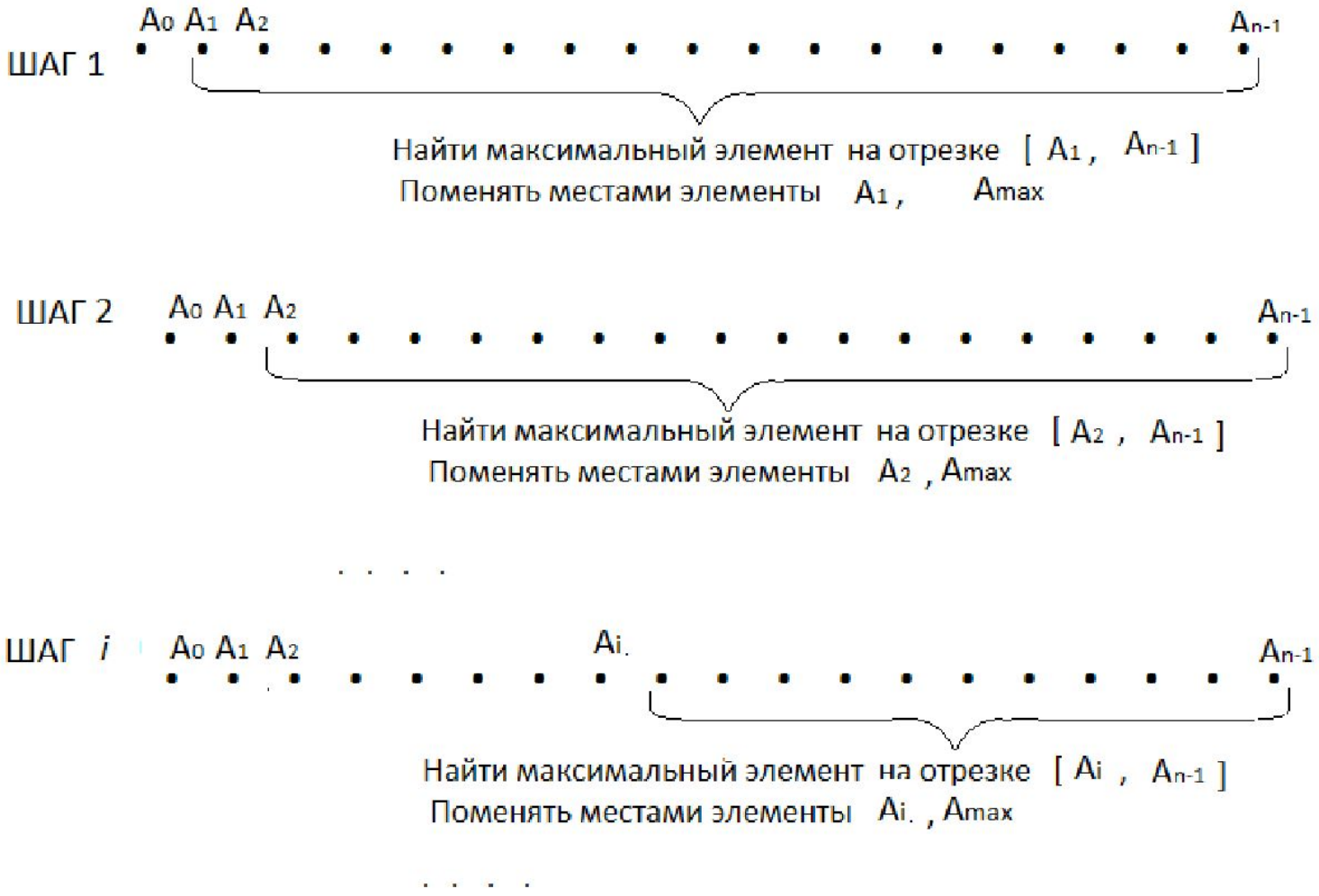
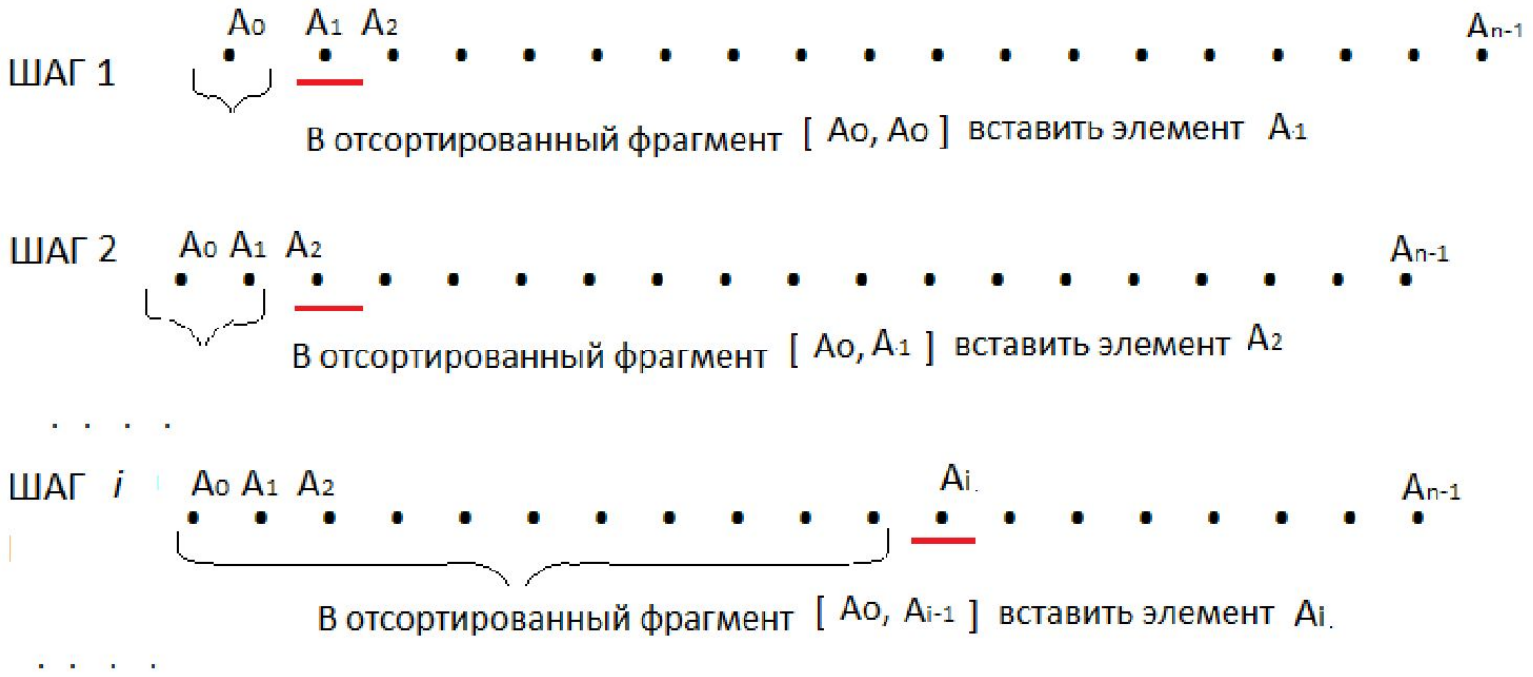
Алгоритмы сортировки

Сортировка выбором



Сортировка вставками



Сортировка слиянием

Сортировка фрагмента массива [A_p , A_q] определяется рекурсивно.

Функция MergeSort (p, q)

Если (p==q) выход;

Если (q − p ==1) упорядочить фрагмент из двух элементов; выход;

r = (p+q)/2

MergeSort (p,m)

MergeSort (p+1,q)

Соединить два упорядоченных фрагмента в один (функция Merge(p, q, r))

```
#include <Windows.h>          // Файл 1.h с реализацией методов сортировки
#include <iostream>
#include <stdlib.h>
#include "time.h"

const int N =  100;          // 1000;  100 000
int Arr__[N], Arr[N];        //  Arr - массив сортируемых значений.
                             //  Arr__  массив используемый для инициализации  Arr
void Out(int K, int M, int* A)                    // Вспомогательная функция, выводит K начальных
 {  int i = 0;                                    //значений массива,  K значений  из середины
   if (N < 3 * K)                                 // K последних значений
   {   int i = 0;
       while (i < N) {  if (i % M == 0)  printf("\n");          // Если размер массива < 3*K , то
выводится                          printf("%8d  ", A[i]); i++;        }      // весь массив
      printf("\n");                                            //M - число элементов в строке вывода
   }
    else
   {  int ind[3] = { 0,    N/2-K/2,    N-K  };                //начальные индексы выводимых фрагментов
      for (int i = 0; i < 3; i++)
      { int beg = ind[i];
        for (int j = beg; j < beg + K; j++)    {  if ((j - beg) % M == 0)  printf("\n");
                                                   printf("%8d  ", A[j]);                    }
        printf("\n");
   }   }
}
```

```
void FillArray()                                    // заполнение Arr__ случайными
числами
{   for (int i = 0; i < N; i++)        Arr__[i] = rand();
}

void FillArray__()                                  // другой вариант заполнение Arr__
случайными
{                                                   // числами
time_t     ltime = NULL;                            // соответствует дате 1970.1.1  и времени 0:0:0
time(&ltime);                          // получает число секунд от  0:0:0, 1970.1.1
printf("ltime = %u\n", (unsigned int) ltime);
tm* lt = localtime(&ltime);                        // формирует объект,  содержащий информацию
                                    //о текущей дате и  местном времени
int y = lt->tm_year + 1900;                  // год вычисляется от 1900, т.е. 0-йгод= 1900,
                                 //122-й год = 1900+122 =2022
int M = lt->tm_mon;
int d = lt->tm_mday;
int h = lt->tm_hour;
int m = lt->tm_min;
int s = lt->tm_sec;
printf("%d.%d.%d  %d:%d:%d \n", y, M, d, h, m, s);
srand((unsigned)ltime);
for (int i = 0; i < N; i++)
{ double xi = (double)rand() / RAND_MAX;
            Arr__[i] = (int)(xi * 10000);
```

```
void CopyArray(int* A)                          // Копирование данных из дополнительного-
{                                               //массива  Arr__ в  сортируемый массив  Arr
for (int i = 0; i < N; i++)   A[i] = Arr__[i];
Out(20, 10,  A );
}
//-------------------------------------------------------------
void Sort1 (int* A)                             //Сортировка выбором
{
DWORD start_t = GetTickCount();                 //Текущий тик процессора
int ind, amax;
for (int i = 0; i < N - 1; i++)
{
. . .
}
Out(20, 10, A);                          //вывод фрагментов отсортированного массива
printf("N= %d,  T = %d ", N, GetTickCount()-start_t);
                                         //вывод времени выполнения (число тиков процессора
                                         //на выполнение данного метода )
}
//-------------------------------------------------------------
void Sort2 (int* A)                             //Сортировка вставками
{ . . .

}
```

```
//-----------------------------------------------------------// Сортировка слиянием
void MergeSotr  (int p, int q)
{  if (p < q)
   {   int r = (p + q) / 2;
       . . .
   }
}
int B[N];
void Merge(int p,   int q,  int r)
{ //[p, r], [r+1, q] - уже упорядоченные фрагменты,  нужно сделать один упорядоченный фрагмент
  // пусть B- глоб.  переменная-массив для промежуточных результатов в теле данной функции
        int i1 = p,    i2 = r + 1;                  // начальные индексы фрагментов
        int NN = q - p + 1;                         // всего элементов на отрезке  [p, q]
        bool t1,   t2;
        for (int i = 0; i < NN; i++ )               // слияние двух упорядоченных фрагментов в один
        {       t1 = i1 <= r;                       // проверка, есть ли еще данные в каком либо
фрагменте
                t2 = i2 <= q;
                if (!t2 || t1 && A[i1] > A[i2])          { B[r + i] = A[i1]; i1++;    }
                else                                     { B[r + i] = A[i2]; i2++;    }
        }
        for (int i = 0; i < NN; i++) A[i+r] = B[i+r];     //перезапись фрагмента A[p], A[q] в массиве
A
}
void  Sort3( )     {   MergeSort(0, N - 1);          //  массив  A передается как глобальная
```

```
// Сортировка комбинированная (в порядке возрастания)
void Sort4( int *A )
{  int offset = N;
   bool replace = true;
   while (offset > 1 || replace == true)
  {   offset = offset * 8 / 11;
      if (offset == 0)offset = 1;
       replace = false;
      for (int i = 0; i < N - offset; i++)
      {  int t = A[i];
        if (t > A[i + offset]) {  A[i] =A[i + offset];
                        A[i + offset] = t;
                        replace = true;
                    }
     }
  }
}
```

Суть сортировки: выбирается фрагмент массива A[0] ... A[N-offset-1] (при уменьшении параметра offset размер фрагмента увеличивается).
Элемент A[i] фрагмента сравнивается с элементом A[i+offset] и может быть перестав-лен (внутренний цикл при фиксированном значениии offset).
Внешний цикл - изменение параметра offset. Выход - параметр offset <= 1 и в последнем внутреннем цикле не было ни одной перестановки.

// Сортировка Шелла (в порядке возрастания)

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на расстоянии R, например, начальное значение R=N .

После этого процедура повторяется для некоторых меньших значений R, например R=R/2. Завершается сортировка Шелла упорядочиванием элементов при R=1 (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места .

Пример

Пусть дан список A=(32, 95, 16, 82, 24, 66, 35, 19, 75, 54, 40, 43, 93, 68) (N= 14)

Выполняется его сортировка методом Шелла, а в качестве значений 5,3,1.

На 1-м шаге сортируются подсписки , составленные из всех элементов массива A, различающихся на 5 позиций, то есть:

(32,66,40) - 66 и 40 меняются местами

(95,35,43) - 95 и 35 меняются местами, затем 95 и 43 меняются местами

(16,19,93)

(82,75,68)

(24,54) **Результат:** 32 35 16 69 24 40 43 19 75 54 66 95 93 82

На 2-м шаге сортируются подсписки:

(32 69 43 54 93)

(35 24 19 66 82)

(16 40 75 95) **Результат:** 32 19 16 43 24 40 54 35 75 69 66 95 93 82

Псевдокод для сортировки Шелла

```
ЗАДАЧА Шелл (  a  - массив целый)
Переменные      N,i,j,k,h: целые;
  N  =  РАЗМЕР(a);
  R  =   N/2;
  ПОКА R>0  ВЫПОЛНЯТЬ
       ДЛЯ  i=1 ДО  N-R  ВЫПОЛНЯТЬ
                 j = i;
                 ПОКА (j>=0) И  (a[j]>a[j+R]) ВЫПОЛНЯТЬ
                          a[j] и a[j+R] поменять местами
                          j --;
                  КОНЕЦ_ПОКА;
      КОНЕЦ_ДЛЯ;
      R= R/2
  КОНЕЦ_ПОКА;
КОНЕЦ  Шелл;
```

```
void Test()
{       FillArray__() ;                 //FillArray()
        printf("\n---------%s-------------------\n",    nnn[i]  );
        CopyArray( A);
        Out( 20, 10,  A);
        Sort1 (A);
        Out( 20, 10,  A);

        CopyArray( A);
        Out( 20, 10,  A);
        Sort2 (A);
        Out( 20, 10,  A);

             . . .
}
```

```
ltime = 1645711745
2022.1.25  0:9:5

---------Sort1--------------------

 6331   631  9474  4761  7314  7821  6560   285  2133  3975
 1905  5889  1203  6722  1010  4919  9935  8458  5768  9435
. . . . . .
 6153  3829  8593  1131  7395  8856  3050  1742  5823  6174
 3158  5646   196  8490  8824  4027  9254  7720  4915  5448
. . . . . .
 1150  1521  5812  3540  3123  7374  6706  4191  5099  7883
 1200  6646  7904  4987  5222  2897  1255  7474  3604  9855


   0    0    0    0    0    0    0    0    0    0
   0    0    1    1    1    1    1    1    1    1
. . . . . .
 4999  5000  5000  5000  5000  5000  5001  5001  5001  5001
 5001  5001  5001  5001  5001  5002  5002  5002  5002  5003
. . . . . .
 9997  9997  9997  9997  9997  9997  9998  9998  9998  9998
 9998  9998  9998  9998  9998  9999  9999  9999  9999  9999

 ticks = 10859
```

```
---------Sort2--------------------

6331   631  9474  4761  7314  7821  6560   285  2133  3975
1905  5889  1203  6722  1010  4919  9935  8458  5768  9435
. . . . . .

6153  3829  8593  1131  7395  8856  3050  1742  5823  6174
3158  5646   196  8490  8824  4027  9254  7720  4915  5448
. . . . . .

1150  1521  5812  3540  3123  7374  6706  4191  5099  7883
1200  6646  7904  4987  5222  2897  1255  7474  3604  9855

  0    0    0    0    0    0    0    0    0    0
  0    0    1    1    1    1    1    1    1    1
. . . . . .

4999  5000  5000  5000  5000  5000  5001  5001  5001  5001
5001  5001  5001  5001  5001  5002  5002  5002  5002  5003
. . . . . .

9997  9997  9997  9997  9997  9997  9998  9998  9998  9998
9998  9998  9998  9998  9998  9999  9999  9999  9999  9999
ticks = 8046
```

```
---------Sort3--------------------
. . . . . .

 ticks = 31

---------Sort4--------------------
. . . . . .

 ticks = 360


---------Sort5--------------------
. . . . . .

 ticks = 2313

------------------------------------------------------------------------
---------Sort3--------------------
. . . . . .
 ticks = 325    N=1000000
```

Самостоятельно реализовать:

1. Минимум три алгоритма сортировки из 5 рассмотренных алгоритмов.
   Получить время работы алгоритма для T= 1000, 10000, 50000, 100000, 1000000/

2. Задано натуральное число N.
   Определить количество простых чисел на отрезке [2, N].