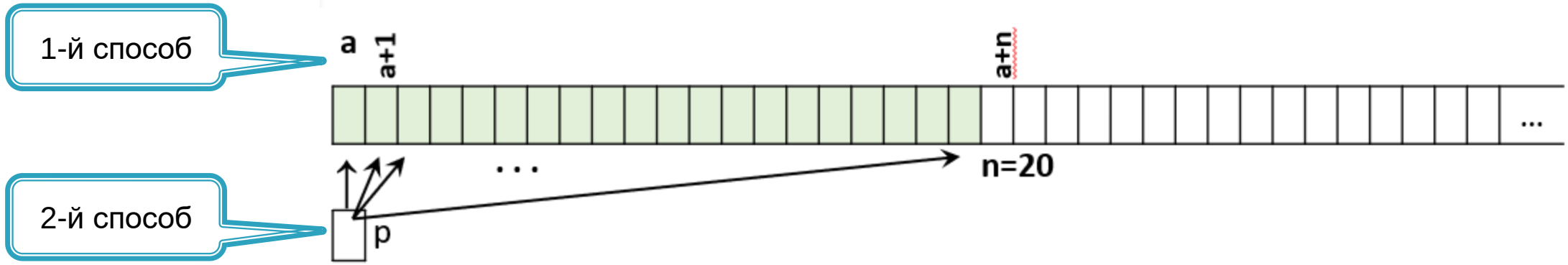
Информационные технологии
и программирование

Язык Си
Массивы. Часть 2

Ракова Ирина Константиновна
каф.О7

Указатели при работе с массивом



2 способа:

- с помощью указателя-константы, содержащего адрес первого элемента массива, прибавляя к нему смещение для вычисления адреса любого элемента
- с помощью указателя-переменной, изменяя её значение для перемещения по элементам массива

```
for (i = 0; i < n; i++ )
    scanf ("%d", &a[i] );
for (i = 0, S = 0; i < n; i++ )
    S += a[i];
```

```
for (i = 0; i < n; i++ )
    scanf ("%d", a+i );
for (i = 0, S = 0; i < n; i++ )
    S += *(a+i);
```

```
for (p = a; p < a+n; p++ )
    scanf ("%d", p );
for (p = a, S = 0; p < a+n; p++ )
    S += *p;
```

Пример. Вариант 1

Поменять знак у первого максимального элемента массива.

```
#include <stdio.h>
#define N 50 // Макс.размер массива
int main()
{
int a[N], i, n, imax;
printf("Введите размер массива:\n“);
scanf(“%d”, &n);  // Реальный размер
if (n > N) n = N;
printf("Введите %d элементов:\n“,n);
for( i=0; i < n; i++ )   // Ввод массива
{
  printf (“a[%d] = ", i );
  scanf ("%d", &a[i] );
}
```

```
imax = 0;
// Поиск индекса максимума
for (i = 1; i < n; i++ )
  if (a[i] > a[imax])
        imax = i;

a[imax] = -a[imax];

printf (“Полученный массив:\n”);
for (i = 0; i < n; i++ )   // Вывод
   printf (“%4d”, a[i]);
return 0;
}
```

Пример. Вариант 2

Поменять знак у первого максимального элемента массива.

```
#include <stdio.h>
#define N 50 // Макс.размер массива
int main()
{
int a[N], i, n, imax;
printf("Введите размер массива:\n“);
scanf(“%d”, &n);  // Реальный размер
if (n > N) n = N;
printf("Введите %d элементов:\n“,n);
for( i=0; i < n; i++ )   // Ввод массива
{
  printf (“a[%d] = ", i );
  scanf ("%d", a+i );
}
```

```
imax = 0;
// Поиск индекса максимума
for (i = 1; i < n; i++ )
  if (*(a+i) > *(a+imax))
        imax = i;

*(a+imax) = - *(a+imax);

printf (“Полученный массив:\n”);
for (i = 0; i < n; i++ )   // Вывод
   printf (“%4d”, *(a+i));
return 0;
}
```

Пример. Вариант 3

Поменять знак у первого максимального элемента массива.

```
#include <stdio.h>
#define N 50 // Макс.размер массива
int main()
{
int a[N], n, *p, *pmax;
printf("Введите размер массива:\n“);
scanf(“%d”, &n);  // Реальный размер
if (n > N) n = N;
printf("Введите %d элементов:\n“,n);
for( p=a; p < a+n; p++ )
// Ввод массива
  scanf ("%d", p );
```

```
pmax = a;
// Поиск адреса максимума
for ( p=a+1; p < a+n; p++ )
  if (*p > *pmax)
        pmax = p;

*pmax = - *pmax;

printf (“Полученный массив:\n”);
for (p=a; p < a+n; p++ )   // Вывод
   printf (“%4d”, *p);
return 0;
}
```

Динамический массив

Динамическое распределение памяти используется для эффективного использования памяти компьютера.

Для работы с динамическими массивами используются **указатели**.

- Размер массива может быть задан во время выполнения программы.
- Размер массива можно изменять.
- Максимально возможный размер массива определяется объемом свободной оперативной памяти.
- Именем массива служит имя указателя, в котором хранится адрес начала блока (массив описывается как указатель).
- После того, как массив перестал быть нужным, память следует освободить.

Динамическое выделение памяти

Функции выделения памяти (в библиотеке *stdlib* или *alloc*):

- **Выделение блока под массив**
  calloc (количество_элементов, размер_элемента)

Функция *calloc* выделяет участок памяти достаточный для размещения n объектов заданного размера size, и возвращает указатель на него или NULL. Выделенная память автоматически заполняется (инициализируется) нулями.

- **Выделение произвольного блока памяти**
  malloc (размер_блока_в_байтах)

Выделяет память для одного объекта указанного размера, для массива указывают размер (n * sizeof (тип)), но память не заполняется нулями. Возвращает указатель на блок памяти или NULL

Если памяти не хватает, то функции возвращают NULL

Работа с динамической памятью

Выделение блока памяти с указанием кол-ва элементов

```
int *pa; // Объявление указателя на массив
pa = (int*) calloc (n, sizeof (int));
или
pa = (int*) malloc (n*sizeof (int));
```

Выделение блока памяти с указанием общего размера

Функции работы с памятью (в библиотеке *stdlib* или *alloc*):

- **Повторное выделение памяти нового размера**
  realloc (адрес_имеющегося_блока, новый_размер)

Функция *realloc* выделяет новый участок памяти нового размера, и возвращает указатель на него или NULL. Старые данные сохраняются, дополнительный участок ничем не заполняется.

- **Освобождение блока памяти**
  free (адрес_блока)

Освобождает область памяти, которая была ранее выделена. После освобождения памяти указатель нужно обнулить.

Пример

Дан целочисленный массив a. Сформировать новый массив b из элементов массива a с четными порядковыми номерами.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{  int *a, *b, n, k, i;
printf("vv n\n");
scanf("%d", &n);  // Размер массива
a=(int*) malloc (n * sizeof (int));
if (a==NULL)  // Контроль выдел. памяти
{ printf(" Ошибка выделения памяти a");
  return 0; }
b=(int*) calloc (n/2, sizeof (int));
if (b == NULL) // Контроль выдел.памят
{ printf("Ошибка выдел. памяти под b");
  return 0; }
for (i=0; i<n; ++i) // Ввод массива
   scanf("%d", a+i);
```

Выделение памяти под массив a

Выделение памяти под массив b

```
printf("Массив a\n"); // Вывод массива a
for (i = 0; i < n; i++)
     printf ("%d\t", *(a+i));
//  Формирование нового массива
for (k=0, i =1; i < n; i+=2, k++)
     b[k] = a[i];
 printf(" \n Массив b\n ");
for (i = 0; i < k ;i++) // Вывод массива b
     printf(" %d ", b[i]);
free(a); a = NULL;
free(b); b = NULL;
return 0;
}
```

Освобождение памяти

Способы обращения к элементам массива не зависят от способа выделения памяти

Пример

Дан целочисленный массив a из n элементов (n<=20). Сформировать новый массив b из элементов массива a, меньших 5.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{  int *a, *b, *pa, *pb, n, k;
printf("vv n\n");
scanf("%d", &n);  // Размер массива
a=(int*) malloc (n * sizeof (int));
if (a==NULL)  // Контроль выдел. памяти
{ printf(" Ошибка выделения памяти a");
  return 0; }
b=(int*) calloc (n, sizeof (int));
if (b == NULL) // Контроль выдел.памят
{ printf("Ошибка выдел. памяти под b");
  return 0; }
for (pa=a; pa<a+n; pa++) // Ввод массива
   scanf("%d", pa);
```

Выделение памяти под массив a

Выделение памяти под массив b

```
printf("Массив a\n"); // Вывод массива a
for (pa = a; pa<a+n; pa++)
     printf("%d ",*pa);
//  Формирование нового массива
for (pb = b, pa = a; pa < a+n; pa++)
  if (*pa < 5)
  {    *pb =*pa;
       pb++;    }
k = pb-b;
b=(int*) realloc( b, k* sizeof (int));
printf(" \n Массив b\n ");
// Вывод массива b
for ( pb = b; pb < b+k; pb++)
    printf(" %d ", *pb);
free(a); a = NULL;
free(b); b = NULL;
return 0;
}
```

Освобождение памяти

Сортировка массивов

Сортировка - процесс упорядочения множества объектов по заданному признаку

Сортировка - это классическая задача программирования. 25% времени работы компьютеров приходится на реализацию этого процесса. Данные могут быть отсортированы:

- По возрастанию - каждый следующий элемент больше предыдущего (a[1]<a[2]…<a[n]);
- По неубыванию - каждый следующий элемент не меньше предыдущего, т. е. больше или равен (a[1]<=a[2]…<=a[n]);
- По убыванию - каждый следующий элемент меньше предыдущего (a[1]>a[2]…>a[n]);
- По невозрастанию - каждый следующий элемент не больше предыдущего (a[1]>=a[2]…>=a[n]);

Цель сортировки - облегчить последующие поиск, обновление, исключение, включение элементов в структуру данных.

Сортировка массивов

Основными требованиями к алгоритмам сортировки, как и ко всяким алгоритмам, являются требования по памяти и по времени выполнения. Это предполагает, что сортировка элементов массива выполняется на месте, без передачи их в результирующий массив.

Самыми простыми методами сортировки являются прямые методы сортировки, они требуют порядка $O(n^2)$ сравнений значений.

Прямые методы можно разбить на три категории:

1. сортировка методом прямого включения (сортировка вставками);
2. сортировка методом прямого выбора (минимума или максимума);
3. сортировка методом прямого обмена (попарно-обменных перестановок).

Сложность алгоритма $O(n^2)$ означает, что время работы алгоритма имеет квадратичную зависимость от количества элементов массива.

Сортировка массива вставками

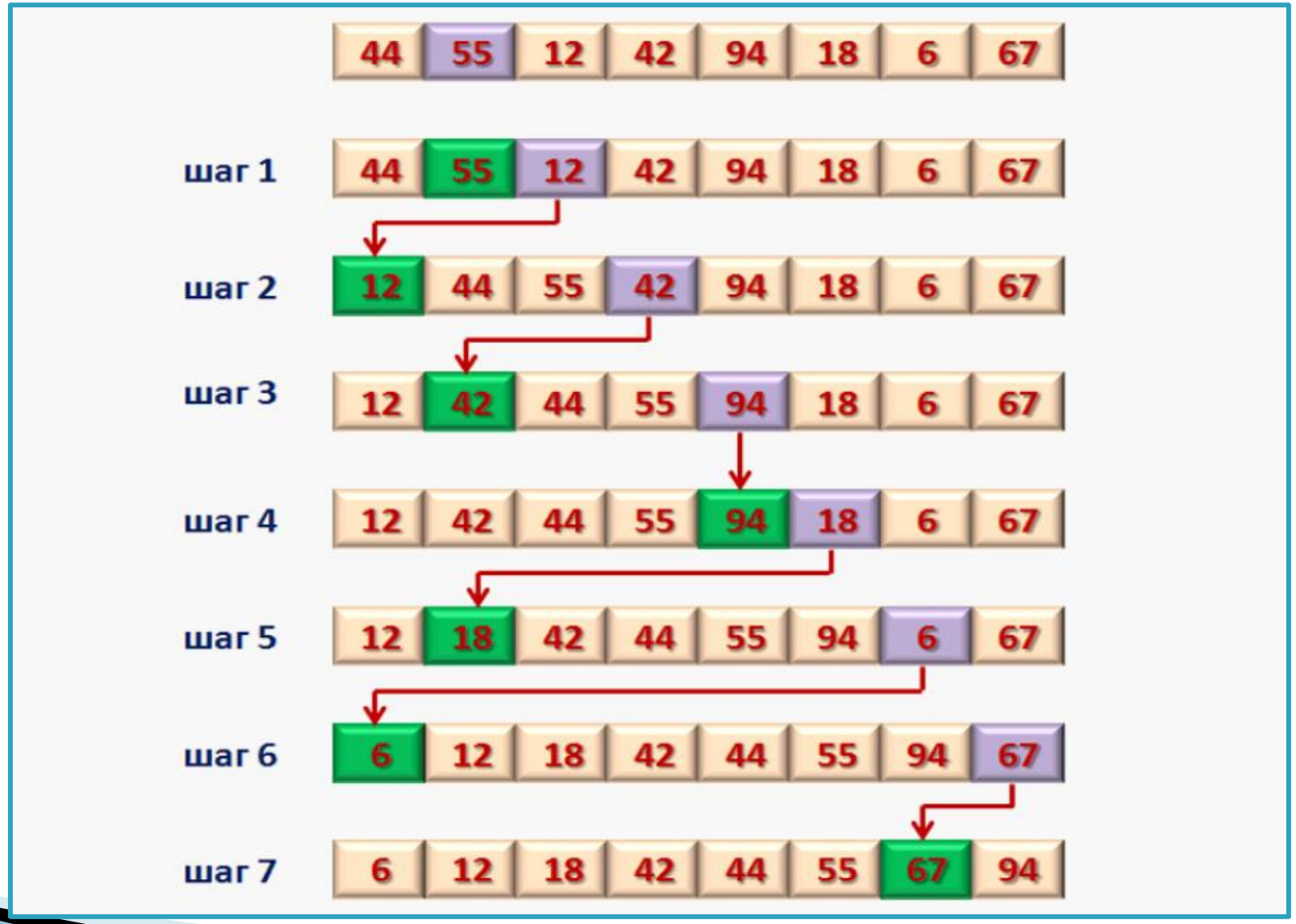
Сортировка вставками делит массив на 2 части — отсортированную и неотсортированную. Из неотсортированной части извлекается очередной элемент. Он вставляется в отсортированную часть массива на то место, где он должен находиться. В результате отсортированная часть массива увеличивается, а неотсортированная уменьшается.

Переброска элементов из неотсортированной части массива в отсортированную продолжается до тех пор, пока в неотсортированной части не останется элементов.

Вставка элемента в отсортированную часть массива может производиться по-разному.

Главное преимущество сортировок вставками - очень быстрая обработка почти упорядоченных массивов.

Сортировка массива вставками



Сортировка массива вставками

```
for(i=1; i<N; i++)
{
        vsp = a[i];
        for(j=i-1; vsp<a[j]&&j>=0; j--)
                a[j+1] = a[j];
        a[j+1] = vsp;
}
```

Исходный массив:

44 55 12 42 94 18 6 67

Сортировка:

i=1	44 55 12 42 94 18 6 67
i=2	12 44 55 42 94 18 6 67
i=3	12 42 44 55 94 18 6 67
i=4	12 42 44 55 94 18 6 67
i=5	12 18 42 44 55 94 6 67
i=6	6 12 18 42 44 55 94 67
i=7	6 12 18 42 44 55 67 94

Результат:

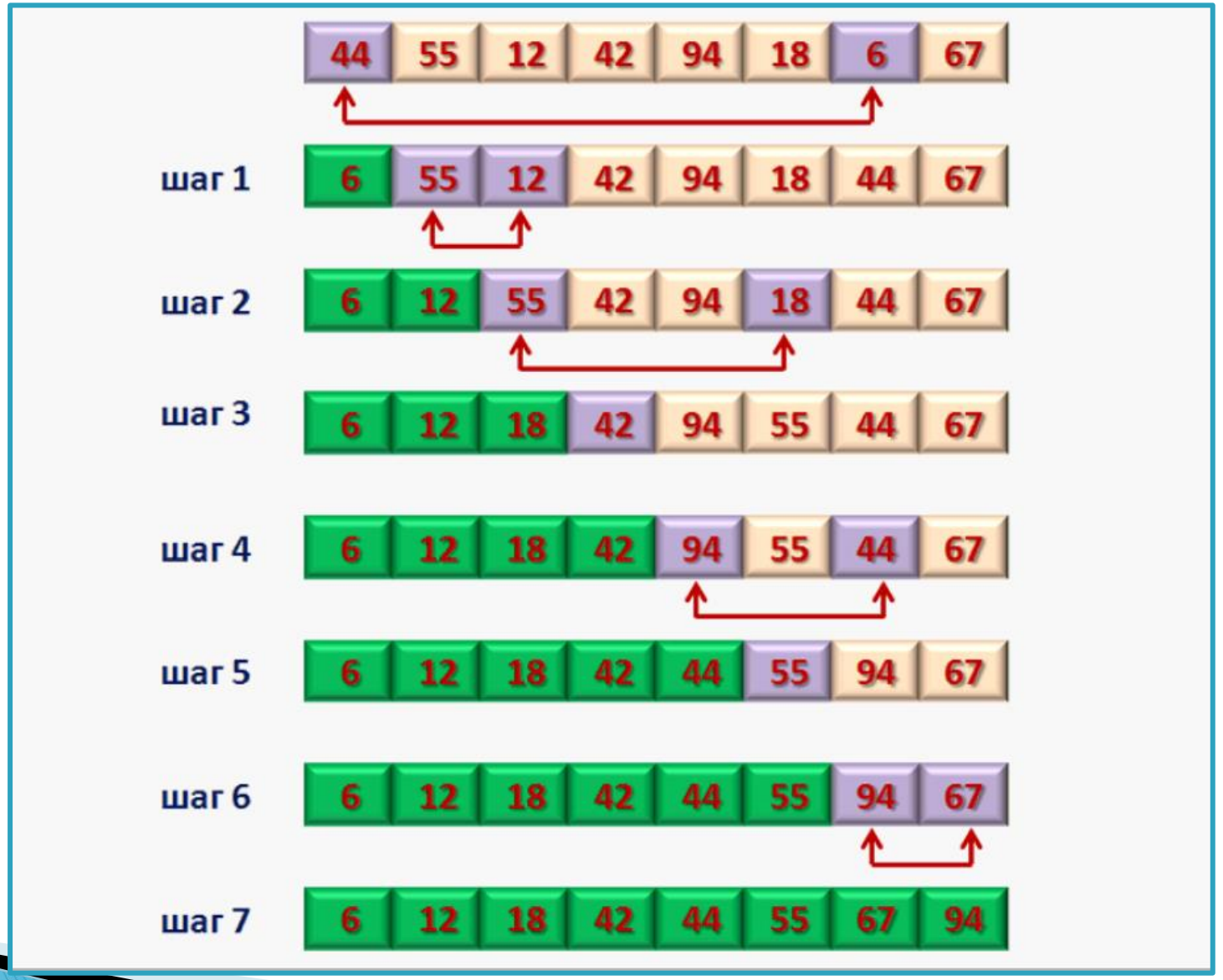
6 12 18 42 44 55 67 94

Сортировка методом выбора

Различают метод минимума и метод максимума. Рассмотрим сортировку в порядке возрастания методом минимума:

- находится наименьший элемент во всем массиве и меняется местами с первым элементом,
- в оставшейся части массива снова ищется наименьший элемент и меняется местами с первым элементом на этом промежутке,
- за один просмотр массива один элемент встанет на свое место, и просматриваемая часть массива уменьшается на один элемент
- так продолжается до тех пор, пока все элементы массива не встанут на свои места, таких проходов требуется *n-1*.

Сортировка методом выбора



Сортировка методом выбора

```
for ( i = 0; i < N-1; ++i )
{
// Поиск мин. от i до N-1 эл.
    min = a[i];
    pmin = i;
    for ( k = i+1; k < N; ++k )
      if ( a [k] < min)
      {
        min = a [k];
        pmin = k;
       }
// Перестановка
     a [pmin] = a[i];
     a [i] = min;
 }
```

Исходный массив:

44 55 12 42 94 18 6 67

Сортировка:

i=0	6 55 12 42 94 18 44 67
i=1	6 12 55 42 94 18 44 67
i=2	6 12 18 42 94 55 44 67
i=3	6 12 18 42 94 55 44 67
i=4	6 12 18 42 44 55 94 67
i=5	6 12 18 42 44 55 94 67
i=6	6 12 18 42 44 55 67 94
i=7	6 12 18 42 44 55 67 94

Результат:

6 12 18 42 44 55 67 94

Сортировка методом обмена

Рассмотрим сортировку в порядке возрастания методом пузырька:

- метод заключается в последовательном просмотре массива снизу вверх и обмене местами соседних элементов в случае, если *a [i] > a [i+1],*
- при первом проходе сравниваются все пары элементов во всем массиве от элемента с номером *n-1* до 0,
- за один просмотр массива один элемент встаёт на свое место (поднимается вверх), и просматриваемая часть массива уменьшается на один элемент,
- так продолжается до тех пор, пока все элементы массива не встанут на свои места, таких проходов требуется *n-1,* но может оказаться, что массив будет упорядочен раньше, тогда проходы по массиву можно прекратить.

Сортировка методом обмена

```
for(i=0; i<N; i++)
{
        for(j=N-1; j>i; j--)
                if(a[j-1]>a[j])
                {
                        vsp = a[j];
                        a[j] = a[j-1];
                        a[j-1] = vsp;
                }
}
```

Исходный массив:

44 55 12 42 94 18 6 67

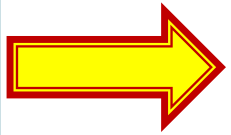
Сортировка:

i=0	6 44 55 12 42 94 18 67
i=1	6 12 44 55 18 42 94 67
i=2	6 12 18 44 55 42 67 94
i=3	6 12 18 42 44 55 67 94
i=4	6 12 18 42 44 55 67 94
i=5	6 12 18 42 44 55 67 94
i=6	6 12 18 42 44 55 67 94
i=7	6 12 18 42 44 55 67 94

Результат:

6 12 18 42 44 55 67 94

За 4 прохода по массиву (i=3) данные упорядочены. Дальнейшие проходы не нужны.



Сортировка методом обмена

```
i=0;
do
{
  f = 0;
  for ( j = N-1; j > i; j--)
     if (a [j-1] > a [j])
     {
        vsp = a[j];
        a [j]=a [j-1];
        a [j-1]=vsp;
        f = 1;
     }
}
while ( ++i < N  &&  f );
```

Перестановок нет

Перестановка произошла

N-1 раз, пока происходят перестановки

Методы сортировки (визуализация):
https://proglib.io/p/sort-gif

Спасибо за внимание