основы алгоритмизации и программирования

СОРТИРОВКА «ГНОМЬЯ»

ПЛАН ЛЕКЦИИ

Часть 1:

1. Понятие алгоритма
2. Идея алгоритма
3. Визуализация
4. Словесное представление алгоритма

Часть 2:

5. Блок-схема
6. Подробный разбор элементов блок-схемы
7. Программная реализация на языках программирования C, C++; C#; Java

ПОНЯТИЕ АЛГОРИТМА

Гномья сортировка (англ. *Gnome sort*) — алгоритм сортировки, похожий на сортировку вставками, но в отличие от последней перед вставкой на нужное место происходит серия обменов, как в сортировке пузырьком. Название происходит от предполагаемого поведения садовых гномов при сортировке линии садовых горшков.

Алгоритм концептуально простой, не требует вложенных циклов. Время работы $O(n^2)$. На практике алгоритм может работать так же быстро, как и сортировка вставками.

ИДЕЯ АЛГОРИТМА

Идея алгоритма очень проста. Пусть имеется массив A размером N, тогда сортировка выбором сводится к следующему:

- Смотрим на текущий и предыдущий элемент массива:
  - если они в правильном порядке, шагаем на один элемент вперед,
  - иначе меняем их местами и шагаем на один элемент назад.
- Граничные условия:
  - если нет предыдущего элемента, шагаем вперёд;
  - если нет следующего элемента, стоп.

Это оптимизированная версия с использованием переменной j, чтобы разрешить прыжок вперёд туда, где он остановился до движения влево, избегая лишних итераций и сравнений.

СЛОВЕСНОЕ ПРЕДСТАВЛЕНИЕ

arr – массив, N- длина массива, i,j- индексы массивов, min – индекс локального минимума

1 Сортировка начинается со второго и третьего элементов i=1, j=2;

2 Если i < N, то к пункту **3**, иначе к пункту **9**

3 если arr[i - 1] > arr[i], то к пункту **4**, иначе к пункту **7**

4 Меняем местами значения arr[i] и arr[i - 1]

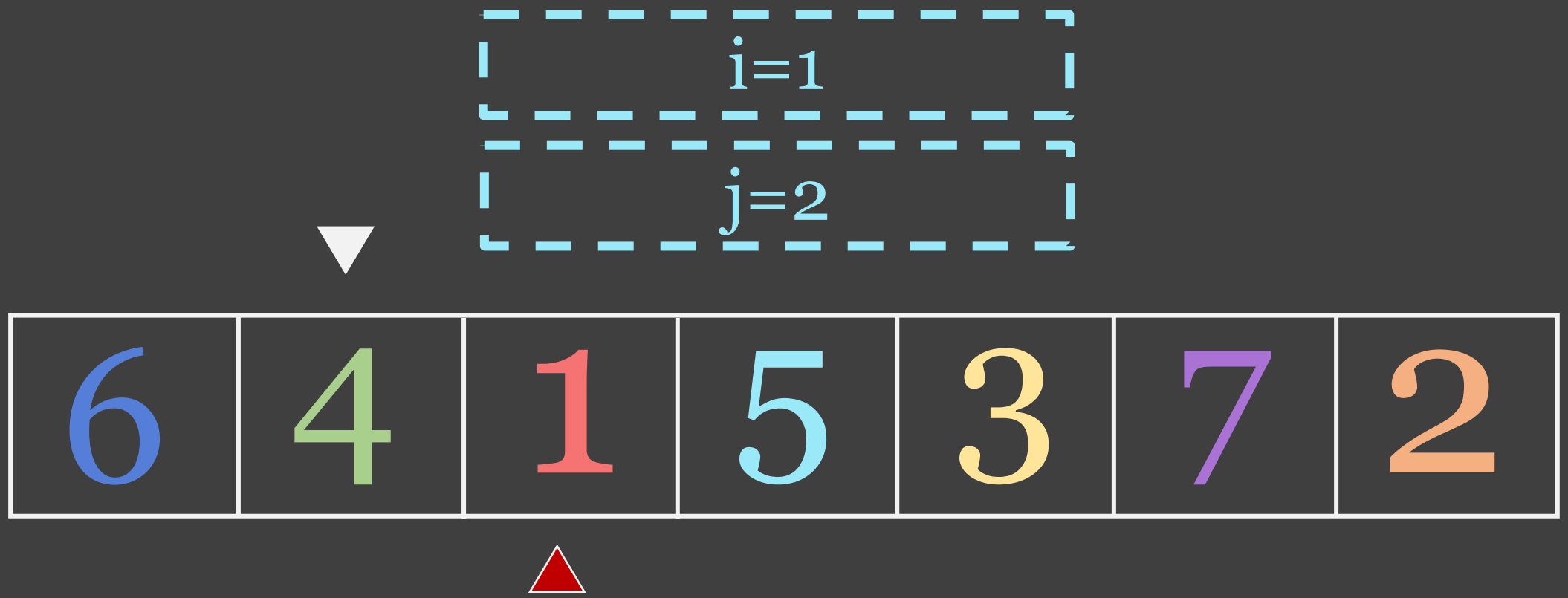
5 Шагаем на один элемент назад i--

6 Если i > 0, то к пункту **2**(используя оператор continue), иначе к пункту **7**

7 i = j++

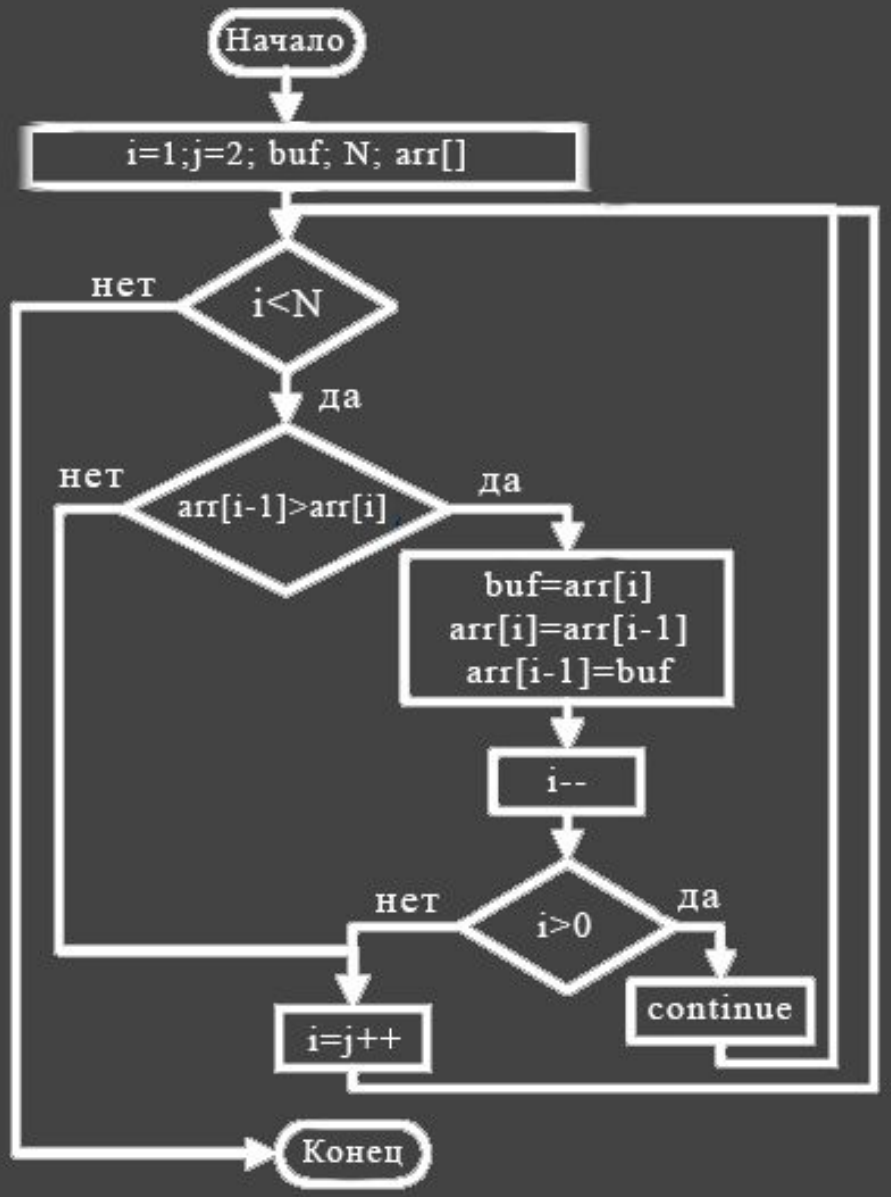
8 К пункту 2.

9 Конец алгоритма



Смотрим на текущий и предыдущий элемент массива:
если они в правильном порядке, шагаем на один элемент вперед, иначе меняем их местами и шагаем на один элемент назад.

БЛОК-СХЕМА


Начало
i=1;j=2; buf; N; arr[]
нет
i<N
да
нет
arr[i-1]>arr[i]
да
buf=arr[i]
arr[i]=arr[i-1]
arr[i-1]=buf
i--
нет
i>0
да
continue
i=j++
Конец

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

1. Первый элемент («пуск») мы используем для обозначения начала работы алгоритма.

2. Во втором элементе, называемом «процесс», мы задаем переменные, которые в дальнейшем будем применять для работы алгоритма.


```
Int N, min, buf, i, j, arr[] = { 6, 4, 1, 5, 3, 7, 2 };
```

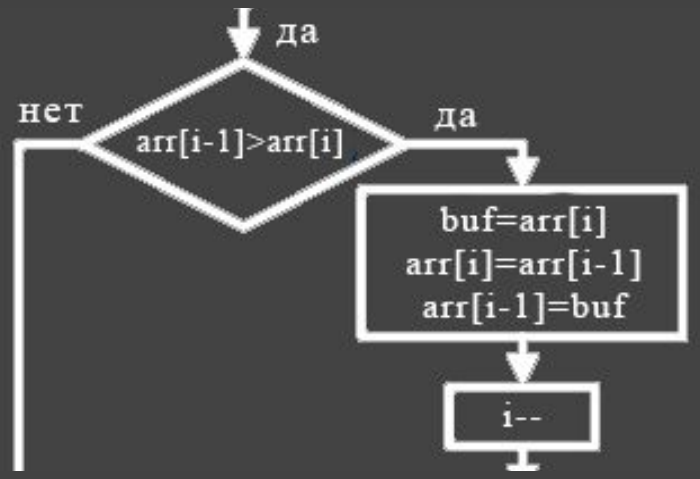
РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

3 Третьим по очереди мы применяем элемент «цикл» для проведения нескольких одинаковых действий и проверки условия: от первого элемента массива до последнего проводится проверка внутри цикла. От «цикла» отходит две ветви: по первой мы идем в случае, если результат проверки условия положительный (заход в цикл и «проход» по всему массиву и процесс сортировки), по второй – если отрицательный (элементы массива закончились).


```
While (i<N) {
…
}
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

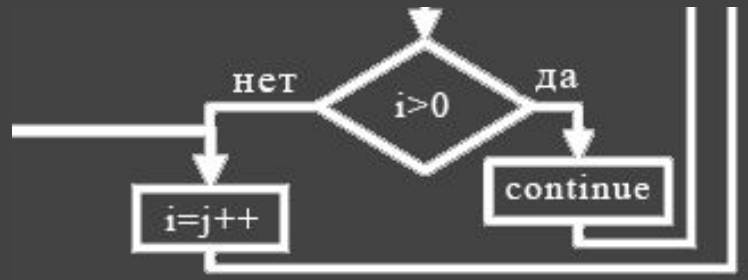
4 Следующим мы используем такой элемент блок-схемы как «цикл» для того, чтобы произвести несколько раз одно и то же действие с выполнением некоторого условия.




```
if (a[i-1]>a[i]){
        B=a[i];
        a[i]=a[i-1];
        a[i-1]=B;
        i--;
        ...
    }
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

5 Далее мы еще раз используем элемент «условие» для проверки условия того, не является ли элемент первым с конца. От «условия» отходит две ветви: по первой мы идем в случае, если результат проверки условия положительный, по второй – если отрицательный, при котором мы рассматриваем следующий элемент массива, приравнивая новый номер к заранее сохраненному элементу .




```
if (i>0) continue;
      }
      i=j++;
    }
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

6 Последний элемент («пуск») мы применяем для обозначения конца работы алгоритма.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ С

```
int main() {
  int i=1, j=2, buf, N=7;
  int arr[]={6, 4, 1, 5, 3, 7, 2};

  while (i<N) {
       if (arr[i-1]>arr[i]) {
           buf=arr[i];
           arr[i]=arr[i-1];
           arr[i-1]=buf;
           i--;
           if (i>0) continue;

       }
          i=j++;
    }
   return 0;
}
```

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ C++

```
#include <iostream>
using namespace std;

int main()
 {
   int N, j=2, i=1, buf;
   arr = new int[6, 4, 1, 5, 3, 7, 2];
   N=7;

   while (i<N) {
        if (arr[i-1]>arr[i]) {
            buf=arr[i];
            arr[i]=arr[i-1];
            arr[i-1]=buf;
            i--;
            if (i>0) continue;

        }
           i=j++;
     }
    return 0;
}
```

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ C#

```
    static void Main(string[] args)
        {
int[] arr = {6, 4, 1, 5, 3, 7, 2};

    int i, j, buf;
   while (i<arr.Length) {
       if (arr[i-1]>arr[i]) {
           buf=arr[i];
           arr[i]=arr[i-1];
           arr[i-1]=buf;
           i--;
           if (i>0) continue;

       }
          i=j++;
    }
  Console.ReadKey();
 }
}
```

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ JAVA

```
public static void sort(int[] arr) {
            while (i < l) {
               if (i > 0 && arr[i - 1] > arr[i]) {
                  [arr[i], arr[i - 1]] = [arr[i - 1], arr[i]];
                  i--;
                  }
              else {
                 i++;
                   }

          } return arr;
}
```