Основы использования массивов в программировании. Ввод данных и сортировка массивов.

Лабораторная работа № А-7.

ЦЕЛЬ РАБОТЫ

Закрепление знаний о работе с одномерными массивами в PHP, получение навыков алгоритмического мышления и способах отладки программы на PHP.

ПРОДОЛЖИТЕЛЬНОСТЬ

4 академических часа (2 занятия)

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу http документы, обеспечивающие возможность пользователю задать произвольной длинны массив чисел и отсортировать его различными алгоритмами с выводом состояния массива на каждом шаге работы.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Работа выполняется в виде двух *php*-файлов: для ввода массива и для отображения процесса его сортировки. Первый файл содержит статически заданную форму, содержащую следующие элементы.

- Одно поле для ввода значения элемента массива.
- Селектор с опциями:
    - сортировка выбором;
    - пузырьковый алгоритм;
    - алгоритм Шелла;
    - алгоритм садового гнома;
    - быстрая сортировка;
    - встроенная функция PHP для сортировки списков по значению.
- Кнопки:
    - добавить еще один элемент;
    - сортировать массив.

При нажатии кнопки "Добавить еще один элемент" на форме с помощью *JavaScript*, без перезагрузки страницы, добавляется еще одно поле для ввода элемента массива (с номером элемента слева от него). При нажатии кнопки "Сортировать массив" в отдельном окне (вкладке) загружается вторая страница, в которой данные формы обрабатываются.

В качестве результата обработки страница должна содержать название алгоритма сортировки, входные данные (введённый массив чисел), результат проверки валидности входных данных (все ли элементы массива числа). Если среди элементов массива есть не числа – сортировка не выполняется, вместо нее выводится соответствующее предупреждение. Если входных данных нет – также выводится предупреждение, сортировка не выполняется.

В процессе сортировки выводится информация о каждой итерации алгоритма:

- номер итерации (сквозная нумерация даже для вложенных циклов);
- текущее состояние массива.

В конце работы алгоритма выводится надпись: "Сортировка завершена, проведено *N* итераций. Сортировка заняла *T* секунд". (где *N* – количество проделанных итераций, *T* – время выполнения сортировки в секундах или долях секунды).

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

Первый файл, несмотря на расширение "*.php*", не содержит *PHP*-кода, зато содержит код *JavaScript*. Он необходим для формирования произвольного количества полей с элементами массива. Такой прием часто используется в экранных формах, когда число строк (элементов) в форме заранее неизвестно. JavaScript позволяет реализовать такую функцию различными способами, рассмотрим в качестве примера один из них.

Листинг А-7. 1

```
<script>
       function addElement(table_name, amount)  // функция добавляет еще один элемент
       {
           var t = document.getElementById(table_name); // объект таблицы

           for(var i=0; i<amount; i++)
           {
                var index=t.rows.length;        // индекс новой строки
                    var row=t.insertRow(index); // добавляем новую строку

                    var cel = row.insertCell(0);     // добавляем в строку ячейку
                    cel.className='element_row';     // определяем css-класс ячейки

                    // формируем html-код содержимого ячейки
                    var celcontent='<input type="text" name="element0">';

                    // добавляем контент в ячеку таблицы
                    setHTML(cel, celcontent);
           }
           // в скрытом поле записываем количество полей (строк таблицы)
           document.getElementById('arrLength').value=t.rows.length;
       }
</script>

<table id="elements">
     <tr><td class="element_row"><input type="text" name="element0"></td></tr>
</table>

<input type="hidden" id="arrLength">
<input type="button" value="Добавить еще один элемент"
       onClick="addElement('elements', 1);">
```

Статический HTML-код на представленном листинге формирует согласно требованиям лабораторной работы кнопку и одно поле для ввода элемента массива. Реализованный в представленной *JavaScript*-программе подход к решению задачи предполагает размещение этого поля в таблице. Тогда каждый раз при нажатии кнопки, с помощью разработанной функции `addElement()` в таблицу добавляется еще одна строка с еще одним полем. Рассмотрим работу указанной функции более подробно.

Обратите внимание, что в качестве аргумента в функцию была передана строка с *id* таблицы, а не соответствующий ей объект. Поэтому первая строка тела функции с помощью `getElementById()` определяет соответствующий таблице объект по ее имени. В дальнейшем работа JavaScript-программы происходит уже именно с этим объектом.

Вторым аргументом функции, введенным для универсальности, является `amount` – число добавляемых в таблицу строк (полей). Т.к. в лабораторной работе требуется при нажатии кнопки добавление только одной строки, то при вызове функции этот аргумент равняется одному. В теле функции этот аргумент используется как предел цикла со счетчиком – в этом цикле последовательно на каждой итерации в таблицу добавляется одна строка. Следовательно, в таблицу будет добавлено ровно столько строк, сколько указано в переменной `amount`.

Сам процесс добавления строки происходит следующим образом. Сначала определяется количество уже присутствующих в таблице строк и сохраняется в переменной `index`. Затем с помощью функции `insertCell()` на последнее место в таблице добавляется новая строка. Ее

номер будет соответствовать количеству строк в таблице до ее добавления (нумерация начинается с нуля), т.е. значению в переменной `index`. Обратите внимание: если в качестве аргумента функции `insertCell()` указать ноль – строка будет добавлена на первом месте, если `index-1` – то на предпоследнем.

Добавление в таблицу строки не означает автоматическое добавление в них ячеек – это необходимо сделать явно с помощью функции `insertCell()`. Получив таким образом объект, идентифицирующий ячейку, мы можем изменять ее свойства (например, присвоить имя *CSS*-класса) и определять ее содержимое. Контент ячейки предварительно формируется в переменной `celcontent`, а затем добавляется в нее с помощью пользовательской функции `setHTML()`.

Обратите внимание – каждая строка будет содержать поле с одним и тем же именем. Чтобы обработчик формы получил все элементы массива имена полей должны отличаться. Самостоятельно доработаете JavaScript-программу так, чтобы итоговые поля назывались "*element*X", где *X* – номер поля по порядку начиная с нуля.

В конце программы общее количество строк таблицы, а значит общее количество полей с элементами массива, а значит и длина массива будет присвоена значению скрытого поля с *id*="*arrLength*" – это несколько упростит дальнейшую обработку массива *PHP*-программой из второго файла.

Для завершения первой части лабораторной работы необходимо лишь разработать код функции `setHTML()`, которая бы для указанного объекта (в данном случае для ячейки таблицы) устанавливала бы внутреннее содержимое (например, для объекта-блока с *id* "*block*", заданного кодом `<div id="block">This is the innerContent</div>` внутренним содержимым будет строка "*This is the innerContent*"). В JavaScript у объектов есть свойство `innerHTML`, присвоение которому *HTML*-кода и означает требуемую операцию. Но, к сожалению, оно доступно не во всех браузерах, поэтому и есть необходимость определить подобную функцию.

Листинг А-7. 2

```
function setHTML(element, txt)
{
	if(element.innerHTML)
		element.innerHTML = txt;
	else
	{
		var range = document.createRange();
		range.selectNodeContents(element);
		range.deleteContents();
		var fragment = range.createContextualFragment(txt);
		element.appendChild(fragment);
	}
}
```

Представленный код первой страницы практически закончен. Но для выполнения условий лабораторной работы самостоятельно модифицируйте *HTML*-код и *JavaScript*-программу так, чтобы слева от поля в отдельной колонке выводился номер (ключ) элемента массива; также модифицируйте и упростите функцию `addElement()` так, чтобы она всегда добавляла в таблицу только одну строку; добавьте в статический *HTML*-код теги формы и ее не указанных элементов.

Второй файл, содержащий PHP-программу для обработки данных, также достаточно прост. Сформулируем словесное описание алгоритма следующим образом.

1. Если данные не переданы в обработчик – вывести сообщение, прекратить работу.
2. Если среди переданных элементов есть не число – вывести сообщение, прекратить работу.
3. Выбрать алгоритм сортировки.
4. Вывести название алгоритма, входные данные, сообщение о прохождении валидации элементов массива.
5. Сохранить текущее время.
6. Провести сортировку массива выбранным алгоритмом с выводом информации о ходе алгоритма.
7. Засечь разницу между текущим временем и сохраненным ранее.

8. Вывести сообщение о завершении алгоритма, количество итераций алгоритма, затраченное время.

Переведем описание на русском языке на язык *PHP*. Для простоты описания будем считать, что все алгоритмы сортировки уже известны и определяются номером от нуля до одного (для краткости рассмотрим вариант с двумя алгоритмами). Тогда функции `sort_0()` и `sort_1()` сортируют массив и выводят информацию о ходе процесса по соответствующим алгоритмам.

Листинг А-7. 3

```
if( !isset($_POST['element0']) )     // если данных нет
{
     echo 'Массив не задан, сортировка невозможна'; // сообщение
     exit();                                        // и завершение программы
}

for($i=0; $i<$_POST['arrLength']; $i++)              // для всех элементов массива
     if( arg_is_not_Num( $_POST['element'.$i] ) )    // если элемент массива не число
     {
          // выводим сообщение и завершаем программу
          echo 'Элемент массива "'. $_POST['element'.$i].'" - не число';
          exit();
     }

// определим реализуемый алгоритм и выведем его название
if( $_POST['algoritm']==='0' )
     echo '<h1>Алгоритм 0</h1>';
else
if( $_POST['algoritm']==='1' )
     echo '<h1>Алгоритм 1</h1>';

$arr = array(); // создаем пустой массив для формирования сортируемого списка
echo 'Исходный массив<br>----------------------------<br>';

for($i=0; $i<$_POST['arrLength']; $i++)         // для всех элементов массива
{
      echo '<div class="arr_element">'.$i.': '.
           $_POST['element'.$i].'</div>'; // выводим текущий элемент и его номер

     $arr[] = $_POST['element'.$i]; // добавляем элемент в массив для сортировки
}

// сообщение об успешной валидации
echo '<br>----------------------------<br>Массив проверен, сортировка возможна';

$time = microtime(true); // засекаем время начала сортировки
if( $_POST['algoritm']==='0' )
     $n = sort_0( $arr );       // запускаем сортировку по первому алгоритму
else
if( $_POST['algoritm']==='1' )
      $n = sort_1( $arr );          // запускаем сортировку по второму алгоритму

// выводим сообщение о завершении сортировки
echo 'Сортировка завершена, проведено '.$n.' итераций. ';

// считаем и выводим затраченое на сортировку время
echo 'Затрачено '.($time- microtime(true)).' микросекунд!';
```

Приведенную программу вполне можно оптимизировать, но в приведенном виде она довольно точно соответствует описанному алгоритму. Итак, в первых строках программы осуществляется проверка наличия в массиве с *POST*-параметрами первого (нулевого) элемента массива. Если его нет – это означает что данная страница была вызвана не как результат работы формы, а просто через *URL* без каких-либо *GET*- или *POST*-параметров. В этом случае обработка невозможно, о чем и выводится соответствующее предупреждение.

Далее необходимо проверить все переданные элементы массива на их соответствие целому числу. Использовать функцию `is_integer()` в этом случае нельзя – параметры передаются в массив как строки. Напишем собственную функцию `arg_is_not_Num()`, которая возвращает `true`

если аргумент не целое число (рассмотрим эту функцию более подробно ниже в листинге А-7.4). Тогда, перебирая в цикле все полученные параметры с элементами массива и проверяя их этой функцией мы осуществим валидацию входных данных.

Обратите внимание: мы точно знаем сколько элементов массива передано для обработки, т.к. передали их количество через скрытое поле с именем "*arrLength*". Кроме того, нет необходимости всегда перебирать все элементы – первый же невалидный элемент массива останавливает работу программы.

Далее, в программе определяется выбранный через селектор тип алгоритма, который и выводится в *HTML*-коде. Для вывода исходного массива (входных данных) также используется цикл: все элементы "оборачиваются" в блок `<div>` – определив *CSS*-стиль для этого блока набор элементов массива можно вывести очень компактно и наглядно (используйте свойства `float`, `margin`, `padding`). Параллельно в этом же цикле элементы массива вынимаются из ассоциированного массива `$_POST` с ключами в виде строк и добавляются в специально созданные ранее список `$arr`.

Теперь осталось собственно отсортировать массив, что и происходит с помощью вызова соответствующей функции. Предварительно в переменной `$time` засекается системное время (с точностью до микросекунды) до начала выполнения алгоритма. Разница между системным временем после выполнения алгоритма и значением переменной `$time` и будет определять длительность выполнения сортировки. Число итераций алгоритма будем брать как результат работы соответствующей функции.

Листинг А-7. 4

```
function arg_is_not_Num( $arg )
{
     if( $arg!=='' ) return true;    // передана пустая строка

     for($i=0; $i<strlen($arg); $i++)  // цикл по всем символам аргумента
          if( $arg[$i]!=='0' && $arg[$i]!=='1' && $arg[$i]!=='2' &&
              $arg[$i]!=='3' && $arg[$i]!=='4' && $arg[$i]!=='5' &&
              $arg[$i]!=='6' && $arg[$i]!=='7' && $arg[$i]!=='8' &&
              $arg[$i]!=='9' ) // если встретилась не цифра
                   return true; // возвращаем true

      // строка состоит из чисел - возвращаем true
      return false;
}
```

Работу используемой выше функции `arg_is_not_Num()` можно реализовать следующим образом. В начале происходит проверка на пустую строку, которая разумеется не является числом. Обратите внимание – в данном случае нельзя проверять условие (`$arg`) – необходимо явное сравнение аргумента с пустой строкой. Это следует делать из-за того, что PHP достаточно вольно преобразует типы. И уверенность в том, что строка "*0*" не будет преобразована сначала в число ноль (при передаче в качестве аргумента функции), а затем и в `false`, отсутствует.

Далее в цикле перебираются все символы строки: если среди них встретился отличный от цифры символ – работа функции завершается, возвращается `true` (аргумент не число). Если все символы аргумента прошли проверку – возвращается `false`. Функцию вполне можно упростить с помощью регулярных выражений, которые будут рассматриваться в последующих лабораторных работах.

Для завершения лабораторной работы самостоятельно переименуйте соответствующие сообщения и функции сортировки согласно названию алгоритмов и напишите их *PHP*-код (используйте раздел "Справочная информация" этой лабораторной работы). Добавьте в программу стандартную функцию сортировки массивов (без учета количества итераций). Сравните время работы реализованных вами функций и встроенной функции *PHP,* сделайте выводы.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Сортировка массива по значению элементов	`sort($arr); // в порядке возрастания`
	`rsort($arr); // в порядке убывания`
	`assort($arr); // по возрастанию с сохранением ключей элементов; для ассоциированных массивов`
	`arsort($arr); // по убыванию с сохранением ключей элементов;для ассоциированных массивов`
	Функции возвращают true в случае успешной сортировки и false в случае ошибки.

Досрочное прекращение выполнения PHP-программы	`exit();` Прекращает выполнение PHP и дальнейшую передачу статического кода в браузер. Фактически в HTML-коде браузера будет только то, что было передано в него (неважно, статически или с помощью PHP) до вызова функции. Удобно использовать для зашиты сайтов от попыток взлома и для прекращения формирования страницы при некорректных данных.
Получение системного времени	`$time = $microtime(true);` Функция `microtime()` возвращает текущую метку веремени *Unix*. Если необязательный аргумент указан и равен `true` – время возвращается в виде вещественного числа, обозначающего секунды с точностью до микросекунды. Если аргумент не указан или равен `false` – метка времени имеет формат "*msec sec*".

АЛГОРИТМ СОРТИРОВКИ МАССИВА ВЫБОРОМ

Сортировка выбором, пожалуй, самый простой алгоритм. Его суть заключается в том, что мы находим в массиве минимальный элемент и меняем его местами с первым элементом массива. Затем ищем минимальный элемент уже начиная со второго и меняем его местами со вторым. И так далее, пока элементы массива не закончатся.

Другими словами, можно представить себе отсортированную и неотсортированную часть массива. В начале работы отсортированной части нет, т.к. весь массив не отсортирован. После первой итерации, когда на первом месте размещен минимальный элемент, можно сказать что отсортированная часть – первый элемент массива, а неотсортированная – массив начиная со второго элемента. После второй, когда найден минимальный элемент неотсортированной части предыдущей итерации, отсортированная часть состоит уже из двух элементов массива. На третьей – из трех и т.д.

Обратите внимание, что нет необходимости проводить интеграции для всех элементов массива: когда неотсортированная часть будет состоять из одного элемента – она автоматически станет также отсортированной (массив из одного элемента отсортирован всегда). Тогда словесное описание такого алгоритма можно сформулировать следующим образом.

В цикле для всех элементов массива, кроме последнего делаем следующие итерации.

- Начиная текущего ищем минимальное значение элемента массива и запоминаем его индекс.
- Если найденный индекс больше текущего, т.е. минимальный элемент не располагается в самом начале неотсортированной части массива – меняем местами текущий элемент и элемент с найденным индексом.

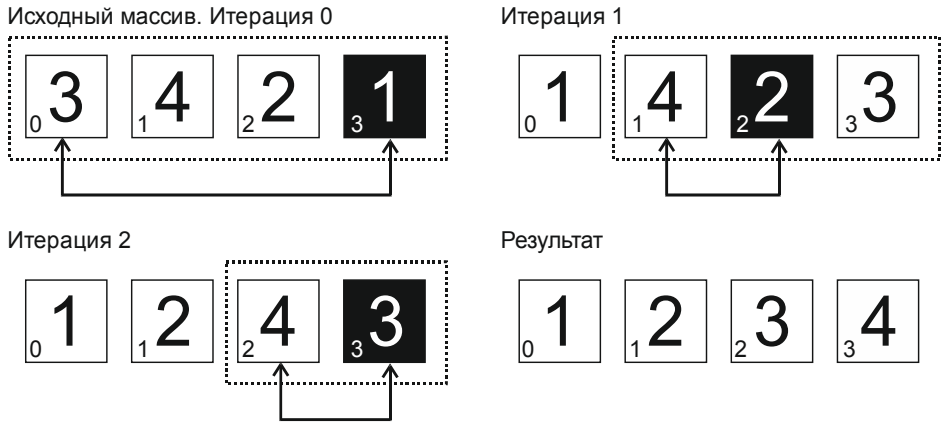
Запишем этот алгоритм с помощью PHP в виде отдельной функции `sorting_by_choice()`.

Листинг А-7. 4

```
function sorting_by_choice( $arr ) // функция сортировки выбором
{
	for($i=0; $i<count($arr)-1; $i++) // цикл для всех элементов списка
	{
		// неотсортированной частью массива считаем элементы начиная от текущего
	    $min=$i;

	    for($j=$i+1; $j<count($arr); $j++) // ищем минимальный элемент
	              if( $arr[$j]<$arr[$i] ) $min=$j; // в неотсортированной части

	    if( $min > $i+1 )     // если минимальный элемент не первый в
	    {                 // неотсортированной части массива
	         $element = $arr[$i];       // меняем его с первым
	         $arr[$i]=$arr[$min];
	         $arr[$min] = $element;
	    }
	}
	return $arr;
}
```

В качестве аргумента функции передается неотсортированный массив. В теле функции запускается цикл по всем элементам массива, кроме последнего. Если массив пустой или содержит только один элемент – ничего делать не надо, массив уже отсортирован. В противном случае на каждой итерации цикла определяется индекс минимального элемента массива начиная с текущего. Для этого в начале установим текущий элемент как минимальный, т.е. элемент с индексом `$i`. Затем в цикле переберём все последующие за ним элементы – если встретился меньший, то уже он становится минимальным, а его индекс сохраняется в переменной `$min`.

Если в неотсортированной части массива минимальный элемент уже стоит на первом месте – ничего делать не надо. Если же нет, т.е. если найденный индекс больше текущего индекса – два элемента меняются местами. Рассмотрим работу этого алгоритма для небольшого массива.

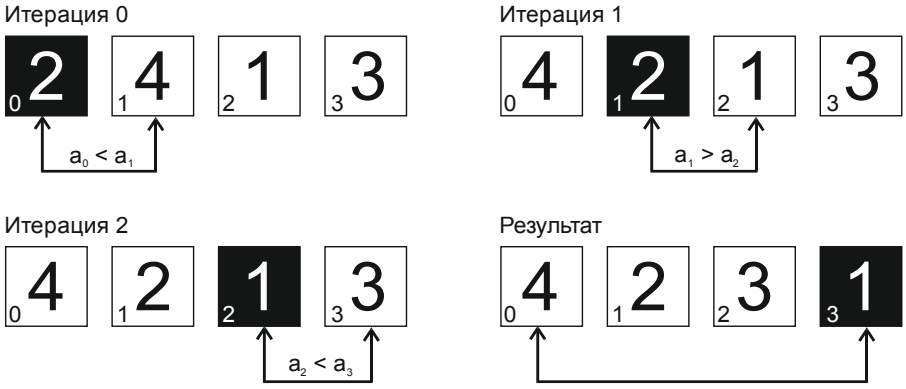


На рисунке пунктирной рамкой обведена неотсортированная часть массива. В начале первой итерации весь массив не отсортирован. Ищется минимальный элемент среди неотсортированной части – в данном случае это элемент с индексом 3 и значением 1. Т.к. индекс 3 больше текущего элемента 0 – они меняются местами. На второй итерации неотсортированная часть массива уже меньше, минимальный элемент расположен по ключ у 2. Меняем его с элементом с индексом 1. Аналогично проводим третью и последнюю итерацию и на выходе получаем полностью отсортированный массив.

Рассмотренный алгоритм очень хорошо, а порой и более эффективно чем его более сложные и продвинутые аналоги, работает для небольших массивов. Но для больших массивов алгоритм всегда проигрывает им.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно.

Самый известный алгоритм, изучаемый обычно еще в школе. Его суть заключается в том, что более "легкие" элементы "всплывают" наверх – отсюда и название по аналогии с пузырями в воде. Для этого все элементы последовательно в цикле сравниваются с соседним: если элемент меньше следующего, то они меняются местами.



Проведя эту операцию в цикле для всех элементов массива окажется, что самый "лёгкий" элемент находится на последнем месте. Например, нам необходимо отсортировать массив (2, 4, 1, 3). Рассмотрим работу алгоритма. На нулевой итерации берется элемент с индексом 0 и сравнивается со следующим, т.е. с индексом 1. Т.к. два меньше четырех, то элементы меняются местами – двойка "всплывает" вверх, а четверка "оседает" вниз.

На следующей итерации рассматриваем уже индексы 1 и 2. Обратите внимание: раньше элемент с индексом 1 имел значение 4, но после первой итерации оно поменялось на 2. Поэтому сравнивается элементы 2 и 1: единица "легче" (меньше) двойки, поэтому ничего не происходит. На второй итерации сравниваются значения 1 и 3, после нее элемент со значением 1 "всплывает" на самый верх массива: именно этого мы и добивались.

Таким образом на каждой итерации цикла рассматривается элемент с индексом номера итерации и следующий за ним, т.е. номер итерации плюс один. А для полного прохода по всему массиву необходимо *N*-1 итераций, где *N* – длинна массива. Поэтому для пустого массива или массива с одним элементом сортировка не производится – он уже априори отсортирован. Опишем приведенный алгоритм с помощью естественного языка следующим образом.

В цикле со счетчиком *i* от 0 до *N*-2 (где *N* – длина массива) делаем:

- если элемент *a*[*i*] меньше *a*[*i*+1] – меняем местами их значения.

Переведем описание алгоритма с русского языка на PHP.

Листинг А-7. 5

```
for($i=0; $i<=count($arr)-2; $i++)    // для элементов от первого до предпоследнего
     if( $arr[$i]<$a[$i+1] )          // если текущий элемент меньше следующего
     {
          $temp = $arr[ $i ];         // меняем их местами элементы i и i+1
          $arr[ $i ] = $arr[ $i+1 ];
          $arr[ $i+1 ] = $temp;
      }
}
```

В программе организован цикл со счетчиком от нуля до размера массива минус два. Т.к. нумерация элементов ведется от нуля, то последний рассматриваемый элемент будет иметь индекс *N*-2, т.е. будет предпоследним элементом массива. Сравниваться он будет с элементом (*N*-2)+1 = *N*-1, т.е. с последним элементом. Далее цикл прекращается, т.к. сравнивать последний элемент уже не с чем.

Обратите также внимание на условие в цикле: обычно используется сравнение "меньше", а не "меньше или равно". Действительно, сейчас условие было записано таким образом только для

того, чтобы дословно перевести с русского на PHP. После оптимизации программу можно записать так.

Листинг А-7. 6

```
for($i=0; $i<count($arr)-1; $i++)     // для элементов от первого до предпоследнего
      if( $arr[$i]<$a[$i+1] )         // если текущий элемент меньше следующего
      {
            $temp = $arr[ $i ];       // меняем их местами
            $arr[ $i ] = $arr[ $i+1 ];
            $arr[ $i+1 ] = $temp;
       }
}
```

Перестановка элементов осуществляется если элемент с индексом *i* меньше элемента с индексом *i*+1. Для этого вводится дополнительная переменная `$temp`, в которой сохраняется значение элемента *i*. Если этого не сделать, а сразу присвоить *i*-ому элементу значение *i*+1, то информация о его значении пропадет: оба элемента будут иметь одинаковое значение. А так для окончания перестановки элементу *i*+1 присваивается сохраненное в `$temp` значение *i*-ого.

Вернемся к массиву (2, 4, 1, 3): в результате выполнения разработанной программы он превратится в массив (4, 2, 3, 1) который все еще не до конца отсортирован. Но обратите внимание: после одной итерации один последний элемент массива находится на своем месте. Рассмотрим его и остальные элементы отдельно: (4, 2, 3) и (1). Левый массив содержит более "тяжелые" элементы, чем правый и не отсортирован. Логично предположить, что, если мы его отсортируем и соединим левую и правую часть обратно – то итоговый массив будет полностью отсортирован.

Поэтому применим к левой части уже описанный алгоритм – в результате элемент 2 "всплывет" наверх. Его можно присоединить к массиву (1), т.к. элемент 2 – самый легкий из всех элементов, которые "тяжелее" 1 и его место в начале массива: (2, 1). Остальные два элемента (4, 3) тоже надо провести через алгоритм – этот массив тоже надо упорядочить несмотря на то, что нам кажется очевидным что он упорядочен. На самом деле Вы только что выполнили этот алгоритм самостоятельно в уме: сравнили первый и следующий за ним элементы, убедились, что первый меньше второго и вышли из цикла.

В итоге за три вызова алгоритма, рассматривая при каждом вызове лишь часть массива на *j* меньшую его длины (где *j* – номер вызова), мы получаем полностью отсортированный массив. Функцию, которая выполняет эту операцию, можно представить следующим образом.

Листинг А-7. 7

```
function BubbleSort($arr)
{
      // число повторов: длина массива - 1
      for($j=0; $j<count($arr)-1; $j++)
      {
            // для всех элементов рассматриваемой части массива
               // от нулевого до предпоследнего
               for($i=0; $i<count($arr)-1-$j; $i++)
            if( $arr[$i]<$a[$i+1] )    // если текущий элемент меньше следующего
            {
                  $temp = $arr[ $i ];        // меняем их местами
                  $arr[ $i ] = $arr[ $i+1 ];
                  $arr[ $i+1 ] = $temp;
               }
       }
       return $arr;
}
```

Обычно при реализации для счетчиков циклов используется сначала переменная `$i`, а уже затем `$j` – с точки зрения PHP это не имеет значение. В примере сначала используется `$j`, а потом `$i` только исходя из очередности подачи материала для изучения.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно.

Алгоритм шейкерной сортировки массива

Шейкерная сортировка – немного модифицированный вариант сортировки пузырьком. Если в нем перебор массива осуществляется только в одну сторону, то здесь последовательно идут обход с начала неотсортированной части массива с выбором максимального элемента и обход с конца неотсортированной части с выбором н элемента. Поэтому такой алгоритм называется также "двунаправленная пузырьковая сортировка.

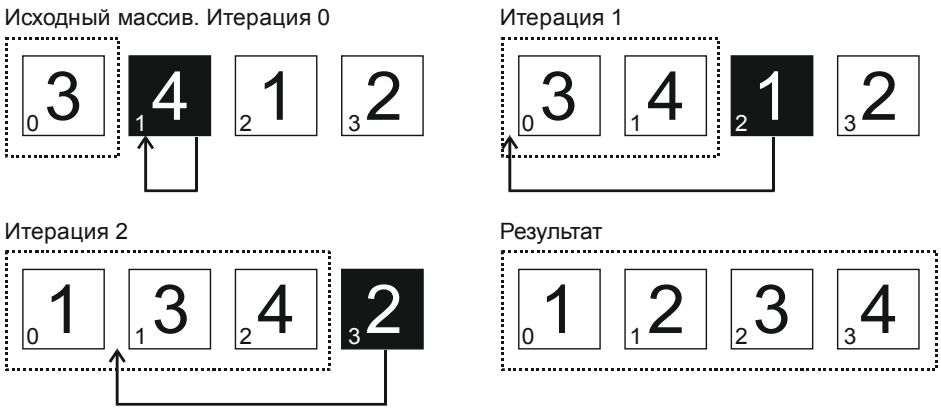
Листинг А-7. 8

```
function ShakerSort($arr)
{
      $left = 1;                  // слева-направо с первого элемента массива
      $right = count($arr)-1;          // справа-налево с последнего
      while( $left <= $right )         // пока границы не сойдутся
      {
            // обход аналогично пузырьковому алгоритму с правой границы до левой
            for($i=$right; $i>=$left; $i--)
                  if( $arr[$i-1]>$arr[$i] )  // если предыдущий элемент больше
                  {
                        $temp=$arr[$i-1];     // меняем его с текущим местами
                        $arr[$i-1]=$arr[$i];
                        $arr[$i]=$temp;
                  }
            $left++;   // сдвигаем левую границу вправо
            // обход аналогично пузырьковому алгоритму с левой границы до правой
            for($i=$left; $i<=$right; $i++)
                  if( $arr[$i-1]>$arr[$i] )  // если предыдущий элемент больше
                  {
                        $temp=$arr[$i-1];     // меняем его с текущим местами
                        $arr[$i-1]=$arr[$i];
                        $arr[$i]=$temp;
                  }
            $right--;  // сдвигаем правую границу влево
       }
       return $arr;
}
```

В начале алгоритма устанавливаются две границы с уже отсортированными частями массива: левая и правая. До тех пор, пока эти границы не сойдутся массив нуждается в сортировке: именно это условие используется в цикле с предусловием.

Первый вложенный цикл начинает обход от правой границы неотсортированной части массива и до его левой границы. В первый раз срабатывания этого цикла самый "легкий" элемент будет перемещен в самое начало массива. Во второй раз – самый легкий из оставшихся на второе место, т.е. левая отсортированная часть массива увеличивается. Поэтому левая граница неотсортированной части массива после окончания цикла сдвигается вправо. Аналогично работает и следующий цикл, проходящий массив слева на право.

Алгоритм сортировки массива вставками



Смысл алгоритма заключается во вставке каждого элемента массива на свое место в уже отсортированной части массива. На первой итерации такой частью является только первый элемент начального массива; текущим элементом – второй. Он сравнивается с первым и либо остается на своем месте, либо переставляется в начало массива. На втором шаге берется третий элемент и сравнивается со всеми предыдущими до тех пор, пока не подобран первый из них, который меньше его: тогда он вставляется сразу за ним.

Листинг А-7. 9

```
function InsertSort($arr)
{
      for($i=1; $i<count($arr); $i++) // для всех элементов начиная с первого
      {
            $val = $arr[$i];           // сохраняем значение текущего элемента
            $j = $i-1;            // начинаем перебор с предыдущего элемента

               // пока не найден элемент меньше текущего
            while( $j>=0 && $arr[$j]>$val )
            {
                  $arr[$j+1] = $arr[$j];     // сдвигаем элементы массива вправо
                  $j--;
            }
            $arr[$j+1] = $val;         // вставляем текущий элемент на свое место
      }
      return $arr;
}
```

В основном цикле осуществляется перебор всех элементов массива, начиная со второго. Если в массиве нет элементов или он только один – то цикл не выполняет ни одной итерации, т.е. такой массив уже отсортирован. Если же элементов два и более – то значение текущего запоминается в переменной `$val`. Далее в теле цикла осуществляется перебор всех предыдущих элементов, т.е. перебор элементов отсортированной части массива, до тех пор, пока не встретился элемент меньший текущего. Для этого удобно использовать цикл с предусловием; дополнительным условием цикла является существование в массиве элемента с рассматриваемым индексом, т.е. `$j>=0` – это важно, т.к. иначе существует риск бесконечного цикла.

На каждой итерации этого вложенного цикла элементы массива сдвигаются вправо, освобождая место для вставки текущего элемента из переменной `$val`. Рассмотрим пример: пусть текущим состоянием массива после двух итераций будет (1, 3, 4, 2), т.е. первые три элемента уже отсортированы. Тогда, в переменной `$val` сохраняется значение 2; перебор во вложенном цикле начинается с элемента со значением 4. Массив, во время работы этого цикла будет изменяться следующим образом: (1, 3, 4, 2) -> (1, 3, 4, 4) ->(1, 3, 3, 4), после чего цикл закончится на элементе со значением 1, т.к. оно меньше значения 2. Сразу после цикла после этого элемента будет вставлен текущий и массив будет полностью упорядочен: (1, 2, 3, 4).

Алгоритм садового гнома

По мнению голландского ученого Дика Груна обычные садовые гномы очень любят поддерживать порядок в их саду. Поэтому, если вечером расставить горшки с цветами, то утром они обязательно будут выстроены в идеальном порядке. При этом гномы, как очень умные и трудолюбивые существа, выработали свой собственный алгоритм сортировки: гном смотрит на предыдущий и текущий горшки. Если они в правильном порядке, то гном шагает вперед к следующему горшку, если нет – он меняет их местами и делает шаг назад. Начинает работу гном всегда со второго горшка; если предыдущего горшка нет – гном шагает вперед; если следующего горшка нет – гном заканчивает свою работу и таинственно исчезает.

Алгоритм очень прост в реализации и не содержит встроенных циклов, но вместе с тем очень перегружен. Левая, уже отсортированная часть массива, в определенный момент времени может оказаться неотсортированной и сравнения с перестановками придется повторять заново.

Листинг А-7. 10

```
function gnomeSort($arr)
{
      $i=1;                 // начинаем со второго элемента массива
      while( $i<count($arr) )    // пока не достигнут последний элемент - цикл
      {
            // если первый элемент массива (предыдущего нет)
               // или текущий элемент больше предыдущего
            if( !$i || $arr[$i-1]<=$arr[$i] )
                  $i++;                     // шагаем вперед
            else       // иначе
            {
                  $temp = $arr[$i];          // меняем элементы местами
                  $arr[$i] = $arr[$i-1];
                  $arr[$i-1] = $temp;
                  $i--;                // шагаем назад
            }
      }
      return $arr;
}
```

Приведённый алгоритм может быть оптимизирована за счет возврата к движению вперед с того места, откуда началось движение назад.

Листинг А-7. 11

```
function gnomeSort($arr)
{
      $i=1;                 // начинаем со второго элемента массива
      $j=2;
      while( $i<count($arr) )    // пока не достигнут последний элемент - цикл
      {
            // если первый элемент массива (предыдущего нет)
               // или текущий элемент больше предыдущего
            if( !$i || $arr[$i-1]<=$arr[$i] )
            {
                  $i=$j;                          // возвращаемся к месту
                  $j++;                     // до которого уже дошли
            }
            else       // иначе
            {
                  $temp = $arr[$i];          // меняем элементы местами
                  $arr[$i] = $arr[$i-1];
                  $arr[$i-1] = $temp;
                  $i--;                // шагаем назад
            }
      }
      return $arr;
}
```

Для этого вводится переменная `$j`, которая сохраняет индекс массива до начал движения назад. Когда вновь начинается движение вперед, то уже рассмотренные элементы массива не сравниваются: сравнение начинается с нерассмотренной части массива.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно.

Алгоритм Шелла сортировки массива

Улучшенный вариант сортировки массива вставками, предложенный Дональдом Шеллом. Его идея заключается в том, что сравниваются не только соседние элементы, но элементы на некотором расстоянии: предварительно осуществляется грубый проход алгоритма, который постепенно становится все точнее и точнее. Для этого выбирается последовательность расстояний, которая заканчивается 1 (например, 5, 3, 1). Для каждого из них выполняется сортировка вставками с сравнением элементов с текущим расстоянием. Последняя итерация содержит классический алгоритм сортировки вставками. Существует различные способы подбора последовательности расстояний, рассмотрим один из самых простых: $D_1 = N/2$; $D_i = D_{i-1}/2$ … 1 – первой расстояние берется как половина длины массива; каждое последующее – как половина предыдущего. Последовательность заканчивается расстоянием 1.

Листинг А-7. 12

```
function ShellsSort($arr)
{
       // вычисляем в цикле последовательность расстояний
      for( $k=ceil(count($arr)/2); $k>=1; $k= ceil($k/2) )
      {
            for($i=$k; $i<count($arr); $i++)     // для всех элементов начиная с первого
            {
                  $val = $arr[$i];      // сохраняем значение текущего элемента
                  $j = $i-$k;     // начинаем перебор с элемента с k меньшим индексом
                      // пока не найден элемент меньше текущего
                  while( $j>=0 && $arr[$j]>$val )
                  {
                        $arr[$j+$k] = $arr[$j];    // сдвигаем элементы вправо
                        $j-=$k;
                  }
                  $arr[$j+$k] = $val;  // вставляем текущий элемент на свое место
            }
      }
      return $arr;
}
```

Первый цикл вычисляет последовательность расстояний `$k` между сравниваемыми элементами: первое расстояние берется как округленное в большую сторону половина длины массива; каждое последующее – как половина предыдущего. Цикл завершается алгоритмом сортировкой вставками, который и реализуется внутренними циклами при расстоянии равном 1.

Внутри цикла размещается уже рассмотренный ранее алгоритм сортировки вставками, отличие которого только в том, что сравниваются и сдвигаются не все элементы подряд, а элементы на расстоянии `$k`. Например, при `$k=3` будут сравниваться элементы (0, 3, 6, 9 …); (1, 4, 7, 10 …) и (2, 5, 8, 11 …). Такой подход позволяет в среднем уменьшить количество перестановок и сравнений по сравнению с обычным алгоритмом вставками.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно.

Алгоритм быстрой сортировки массива

Один из самых эффективных алгоритмов сортировки массивов. Парадоксально, но факт в том, что это модернизация одного из самых медленных алгоритмов – пузырьковой сортировки.

Небольшие изменения позволили достичь поразительных результатов. Разработана английским ученым Чарльзом Хоаром в 1960 году.

Суть алгоритма заключается в разбиении массива на две части с помощью некоторого элемента, называемого "опорная точка". Все элементы меньше опорной точки перекидываются в левую часть массива, все элементы большие или равные опорной точке – в правую (или наоборот для другого направления сортировки). Для двух получившихся частей массивов описанные действия повторяются вновь: т.е. две части рассматриваются как отдельные массивы, которые сортируются тем же способом, с выбором своих опорных точек. Такое разбиение происходит до тех пор, пока оно возможно. В конце все массивы склеиваются в один.

Очень важно правильно выбрать опорную точку: в идеале это должна быть медиана, но для ускорения вычислений можно выбирать среднее арифметическое всех элементов массива. Также можно выбирать значение случайного элемента массива; последний элемент в массиве; значение "центрального" элемента в массиве и т.д.

Листинг А-7. 13

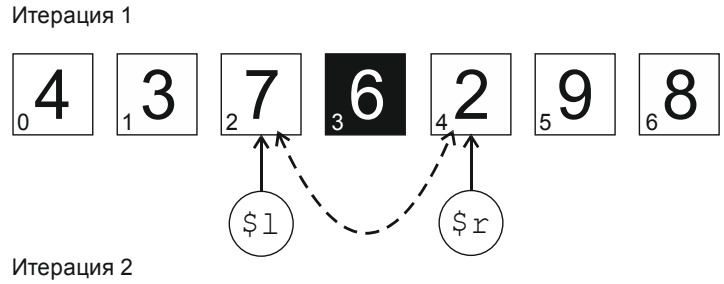
```
// входными параметрами функции являются массив,
// левая и правая граница сортируемой части
function quickSort(&$arr, $left, $right)
{
      $l=$left;       // копируем переменные для манипуляции
      $r=$right;

      $point = $arr[ floor(($left+$right)/2) ]; // вычисляем опорную точку

      do    // цикл сортировки массива
      {
            // сдвигаем левую границу вправо до тех пор,
               // пока не найден элемент массива равный опорному
            while( $arr[$l]<$point ) $l++;

               // сдвигаем правую границу влево до тех пор,
               // пока не найден элемент массива равный опорному
            while( $arr[$r]>$point ) $r++;

            if( $l <= $r ) // если левая и правая граница не пересекаются
            {
                  // меняем текущие элементы местами
                  $temp=$arr[$l]; $arr[$l]=$arr[$r]; $arr[$r]=$temp;
                      $l++;       // сдвигаем границы далее
                  $r--;
            }
      } while( $l<=$r )     // продолжаем цикл пока границы не пересекутся

      if( $r > $left )      // если присутствует левая част массива
            quickSort($arr, $left, $r);     // сортируем ее

      if( $l > $right )     // если присутствует правая часть массива
            quickSort($arr, $l, $right);    // сортируем ее
}

quickSort($arr, 0, count($arr)-1);   // вызов функции быстрой сортировки
```

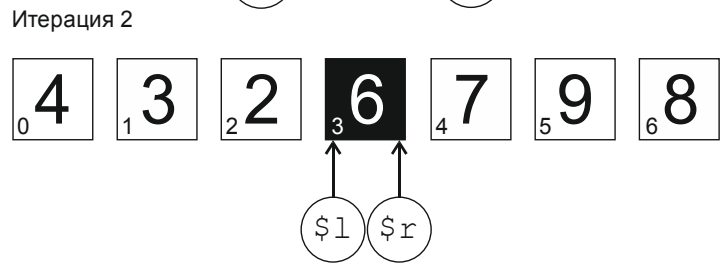
Первое на что следует обратить внимание – функция не возвращает значение. Это возможно т.к. массив передается по ссылке (`&$arr`), в этом случае при его изменении внутри функции он изменяется и в остальной частим программы. Также входными параметрами являются первый и последний индексы массива для которых необходима сортировка. При первом вызове это `0` и `count($arr)-1`, т.е. индексы первого и последнего элемента массива.

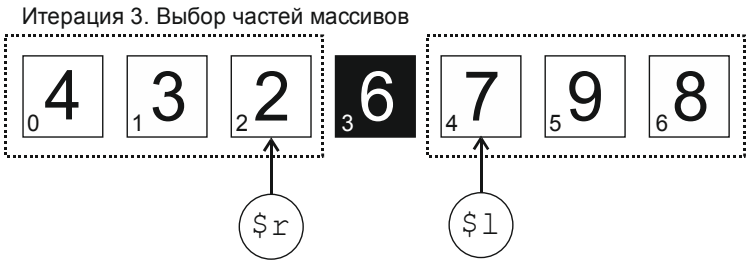
В начале работы функция сохраняет границы сортируемой части массива в переменных `$l` и `$r` – теперь их можно изменять, не опасаясь потерять начальные значения. Опорная точка вычисляется как индекс посередине массива и сохраняется в переменной `$point`. Далее в двух циклах границы смещаются к центру массива до тех пор, пока не достигнут значения опорной точки. Так для

массива (4, 9, 7, 6, 2, 3, 8) опорной точкой будет служить значение 6, а левой и правой границей – 9 и 3.

Если текущая левая граница `$l` не превышает правую границу `$r` – значит наступила ситуация, когда в левой части массива найден элемент которому место в правой, а в правой – элемент которому место в левой. А значит необходимо поменять эти элементы местами и анализировать уже следующие элементы. Далее еще раз проверяется рассмотренное условие о непересекаемости границ и, если оно выполняется, действия по сдвигу границ продолжаются с помощью цикла.







В конце концов правая граница будет меньше левой: тогда все элементы он первого до правой границы будут меньше опорной точки, а элементы от левой границы до конца массива – больше. Тогда, если эти части содержат более одного элемента – они сортируются той же самой функцией, как и весь массив: в этом случает в качестве параметров передаются не границы массива, а границы этих частей.

В задании лабораторной работы необходимо вывести промежуточные состояния массива на каждой итерации. Для этого доработайте приведенную функцию самостоятельно. Также сформируйте еще одну дополнительную функцию, которая бы сортировала массив без передачи в нее нуля и длины массива (эта функция будет вызывать уже сформированную). Для сортировки массива используйте ее.

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы, а также на контрольные вопросы всех предыдущих лабораторных работ.

1. Какие изменения необходимо внести в программу, чтобы она корректно работала без использования информации из скрытого поля *arrLength*?
2. Как при нажатии кнопки "Добавить еще один элемент" добавлять не один, а несколько элементов сразу?
3. Как будет выглядеть обратная к *arg_is_not_Num*() функция (возвращает *true*, если аргумент число)?
4. Детально разберите и расскажите: как работают алгоритмы шейкерной сортировки массива, сортировки слиянием и вставками?
5. В каком случае на листинге 7.9 без условия $j>=0 будет бесконечный цикл?
6. Как изменится работа программы листинга 7.9, если в цикле с предусловием будет $j>0?
7. Какой метод сортировки использует стандартная функция PHP?
8. Как досрочно прекратить выполнения PHP-программы?
9. Если была вызвана функция exit() – будет ли в браузер передана статическая верстка, размещенная после последнего фрагмента кода PHP?
10. Как получить системное время с помощью PHP?
11. Какие параметру присутствуют у функции microtime() и для чего они используются?