PHP sorting algorithms: bubble sort, selection sort, counting sort, quicksort, shellsort, heapsort

Bubble sort:

function bubbleSort(array $arr)

{

$n = sizeof($arr);

for ($i = 1; $i < $n; $i++) {

for ($j = $n - 1; $j >= $i; $j--) {

if($arr[$j-1] > $arr[$j]) {

$tmp = $arr[$j - 1];

$arr[$j - 1] = $arr[$j];

$arr[$j] = $tmp;

}

}

}

return $arr;

}

// Example:

$arr = array(255,1,22,3,45,5);

$result = bubbleSort($arr);

print\_r($result);

Bubble sort improved:

function bubbleSortImproved(array $arr)

{

$n = sizeof($arr);

for ($i = 1; $i < $n; $i++) {

$flag = false;

for ($j = $n - 1; $j >= $i; $j--) {

if($arr[$j-1] > $arr[$j]) {

$tmp = $arr[$j - 1];

$arr[$j - 1] = $arr[$j];

$arr[$j] = $tmp;

$flag = true;

}

}

if (!$flag) {

break;

}

}

return $arr;

}

// Example:

$arr = array(255,1,22,3,45,5);

$result = bubbleSortImproved($arr);

print\_r($result);

Selection sort:

function selectionSort(array $arr)

{

$n = sizeof($arr);

for ($i = 0; $i < $n; $i++) {

$lowestValueIndex = $i;

$lowestValue = $arr[$i];

for ($j = $i + 1; $j < $n; $j++) {

if ($arr[$j] < $lowestValue) {

$lowestValueIndex = $j;

$lowestValue = $arr[$j];

}

}

$arr[$lowestValueIndex] = $arr[$i];

$arr[$i] = $lowestValue;

}

return $arr;

}

// Example:

$arr = array(255,1,22,3,45,5);

$result = selectionSort($arr);

print\_r($result);

Counting sort:

function countingSort(array $arr)

{

$n = sizeof($arr);

$p = array();

$sorted = array();

for ($i = 0; $i < $n; $i++) {

$p[$i] = 0;

}

for ($i = 0; $i < $n; $i++) {

for ($j = $i + 1; $j < $n; $j++) {

if ($arr[$i] > $arr[$j]) {

$p[$i]++;

} else {

$p[$j]++;

}

}

}

for ($i = 0; $i < $n; $i++) {

$sorted[$p[$i]] = $arr[$i];

}

return $sorted;

}

// Example:

$arr = array(255,1,22,3,45,5);

$result = countingSort($arr);

print\_r($result);

Quicksort:

function quicksort(array $arr, $left, $right)

{

$i = $left;

$j = $right;

$separator = $arr[floor(($left + $right) / 2)];

while ($i <= $j) {

while ($arr[$i] < $separator) {

$i++;

}

while($arr[$j] > $separator) {

$j--;

}

if ($i <= $j) {

$tmp = $arr[$i];

$arr[$i] = $arr[$j];

$arr[$j] = $tmp;

$i++;

$j--;

}

}

if ($left < $j) {

$arr = quicksort($arr, $left, $j);

}

if ($right > $i) {

$arr = quicksort($arr, $i, $right);

}

return $arr;

}

// Example:

$arr = array(20,12,4,13,5);

$result = quicksort($arr, 0, (sizeof($arr)-1));

print\_r($result);

Shellsort:

function shellsort(array $arr)

{

$n = sizeof($arr);

$t = ceil(log($n, 2));

$d[1] = 1;

for ($i = 2; $i <= $t; $i++) {

$d[$i] = 2 \* $d[$i - 1] + 1;

}

$d = array\_reverse($d);

foreach ($d as $curIncrement) {

for ($i = $curIncrement; $i < $n; $i++) {

$x = $arr[$i];

$j = $i - $curIncrement;

while ($j >= 0 && $x < $arr[$j]) {

$arr[$j + $curIncrement] = $arr[$j];

$j = $j - $curIncrement;

}

$arr[$j + $curIncrement] = $x;

}

}

return $arr;

}

// Example:

$arr = array(20,12,67,34,4,19,40,75,55,82,5,41,13,25,71);

$result = shellsort($arr);

print\_r($result);

Heapsort (OOP version):

class Node

{

private $\_iData; // data item (key)

public function \_\_construct($key)

{

$this->\_iData = $key;

}

public function getKey()

{

return $this->\_iData;

}

}

class Heap

{

private $\_heapArray;

private $\_currentSize;

public function \_\_construct()

{

$\_heapArray = array();

$this->\_currentSize = 0;

}

/\*\*

\* Delete item with max key (assumes non-empty list)

\*/

public function remove()

{

$root = $this->\_heapArray[0];

// put last element into root

$this->\_heapArray[0] = $this->\_heapArray[--$this->\_currentSize];

// "sift" the root

$this->bubbleDown(0);

return $root; // return reference to removed root

}

/\*\*

\* The "sift" process

\* (heap formation from our array of nodes)

\*

\* @param type $index

\*/

public function bubbleDown($index)

{

$largerChild = null;

$top = $this->\_heapArray[$index]; // save root

while ($index < (int)($this->\_currentSize/2)) { // not on bottom row

$leftChild = 2 \* $index + 1;

$rightChild = $leftChild + 1;

// find larger child

if ($rightChild < $this->\_currentSize

&& $this->\_heapArray[$leftChild] < $this->\_heapArray[$rightChild]) // right child exists?

{

$largerChild = $rightChild;

} else {

$largerChild = $leftChild;

}

// top >= largerChild?

if ($top->getKey() >= $this->\_heapArray[$largerChild]->getKey()) {

break;

}

// shift child up

$this->\_heapArray[$index] = $this->\_heapArray[$largerChild];

$index = $largerChild; // go down

}

$this->\_heapArray[$index] = $top; // root to index

}

public function insertAt($index, Node $newNode)

{

$this->\_heapArray[$index] = $newNode;

}

public function incrementSize()

{

$this->\_currentSize++;

}

public function getSize()

{

return $this->\_currentSize;

}

public function asArray()

{

$arr = array();

for ($j = 0; $j < sizeof($this->\_heapArray); $j++) {

$arr[] = $this->\_heapArray[$j]->getKey();

}

return $arr;

}

}

function heapsort(Heap $Heap)

{

$size = $Heap->getSize();

// "sift" all nodes,

// except lowest level (it means only for half of nodes array)

// we skip lowest level, because lowest level don't have children

for ($j = (int)($size/2) - 1; $j >= 0; $j--) { // make array into heap

$Heap->bubbleDown($j);

}

// display heap

// $arr = $Heap->asArray();

// echo "Heap : ";

// foreach ($arr as $val) {

// echo $val . " ";

// }

// sort the heap

for ($j = $size-1; $j >= 0; $j--) {

$BiggestNode = $Heap->remove();

// use same nodes array

// for sorted elements

$Heap->insertAt($j, $BiggestNode);

}

return $Heap->asArray(); // get sorted array

}

// Example:

$arr = array(81,6,23,38,95,71,72,39,34,53);

$Heap = new Heap();

foreach ($arr as $key => $val) {

$Node = new Node($val);

$Heap->insertAt($key, $Node);

$Heap->incrementSize();

}

$result = heapsort($Heap);

print\_r($result);