<?php

frch();

function frch(){

$sure\_baslangici = microtime(true);

class Node

{

private $\_i;

public function \_\_construct($key)

{

$this->\_i = $key;

}

public function getKey()

{

return $this->\_i;

}

}

class Heap

{

private $heap\_Array;

private $\_current\_Size;

public function \_\_construct()

{

$heap\_Array = array();

$this->\_current\_Size = 0;

}

// Remove item with max key

public function remove()

{

$root = $this->heap\_Array[0];

// put last element into root

$this->heap\_Array[0] = $this->heap\_Array[--$this->\_current\_Size];

$this->bubbleDown(0);

return $root;

}

// Shift process

public function bubbleDown($index)

{

$larger\_Child = null;

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

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

$leftChild = 2 \* $index + 1;

$rightChild = $leftChild + 1;

// find larger child

if ($rightChild < $this->\_current\_Size

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

{

$larger\_Child = $rightChild;

} else {

$larger\_Child = $leftChild;

}

if ($top->getKey() >= $this->heap\_Array[$larger\_Child]->getKey()) {

break;

}

// shift child up

$this->heap\_Array[$index] = $this->heap\_Array[$larger\_Child];

$index = $larger\_Child; // go down

}

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

}

public function insertAt($index, Node $newNode)

{

$this->heap\_Array[$index] = $newNode;

}

public function incrementSize()

{

$this->\_current\_Size++;

}

public function getSize()

{

return $this->\_current\_Size;

}

public function asArray()

{

$arr = array();

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

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

}

return $arr;

}

}

function heapsort(Heap $Heap)

{

$size = $Heap->getSize();

// "sift" all nodes, except lowest level as it has no children

for ($j = (int)($size/2) - 1; $j >= 0; $j--)

{

$Heap->bubbleDown($j);

}

// 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();

}

for ($i=0; $i<=20; $i++) {

$arr[$i] = $i;

}

echo "Original Array : ";

echo implode(', ',$arr);

$Heap = new Heap();

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

$Node = new Node($val);

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

$Heap->incrementSize();

}

$result = heapsort($Heap);

echo "\nSorted Array : ";

echo implode(', ',$result)."\n";

$sure\_bitimi = microtime(true);

$sure = $sure\_bitimi - $sure\_baslangici;

echo "<br>Bekleme süresi: $sure saniye.\n";

//PHP kodlarına ayrılan belleğin miktarını bayt cinsinden döndürür.

echo 'Hafıza kullanımı: ',round(memory\_get\_peak\_usage()/1048576, 2), 'MB';}

function fr(){

$sure\_baslangici = microtime(true);

class Node

{

private $\_i;

public function \_\_construct($key)

{

$this->\_i = $key;

}

public function getKey()

{

return $this->\_i;

}

}

class Heap

{

private $heap\_Array;

private $\_current\_Size;

public function \_\_construct()

{

$heap\_Array = array();

$this->\_current\_Size = 0;

}

// Remove item with max key

public function remove()

{

$root = $this->heap\_Array[0];

// put last element into root

$this->heap\_Array[0] = $this->heap\_Array[--$this->\_current\_Size];

$this->bubbleDown(0);

return $root;

}

// Shift process

public function bubbleDown($index)

{

$larger\_Child = null;

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

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

$leftChild = 2 \* $index + 1;

$rightChild = $leftChild + 1;

// find larger child

if ($rightChild < $this->\_current\_Size

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

{

$larger\_Child = $rightChild;

} else {

$larger\_Child = $leftChild;

}

if ($top->getKey() >= $this->heap\_Array[$larger\_Child]->getKey()) {

break;

}

// shift child up

$this->heap\_Array[$index] = $this->heap\_Array[$larger\_Child];

$index = $larger\_Child; // go down

}

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

}

public function insertAt($index, Node $newNode)

{

$this->heap\_Array[$index] = $newNode;

}

public function incrementSize()

{

$this->\_current\_Size++;

}

public function getSize()

{

return $this->\_current\_Size;

}

public function asArray()

{

$arr = array();

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

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

}

return $arr;

}

}

function heapsort(Heap $Heap)

{

$size = $Heap->getSize();

// "sift" all nodes, except lowest level as it has no children

for ($j = (int)($size/2) - 1; $j >= 0; $j--)

{

$Heap->bubbleDown($j);

}

// 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();

}

for ($i=0; $i<=20; $i++) {

$arr[$i] = $i;

}

echo "Original Array : ";

echo implode(', ',$arr);

$Heap = new Heap();

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

$Node = new Node($val);

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

$Heap->incrementSize();

}

$result = heapsort($Heap);

echo "\nSorted Array : ";

echo implode(', ',$result)."\n";

$sure\_bitimi = microtime(true);

$sure = $sure\_bitimi - $sure\_baslangici;

echo "<br>Bekleme süresi: $sure saniye.\n";

//PHP kodlarına ayrılan belleğin miktarını bayt cinsinden döndürür.

echo 'Hafıza kullanımı: ',round(memory\_get\_peak\_usage()/1048576, 2), 'MB';}

?>

<?php

$sure\_baslangici = microtime(true);

class Node

{

private $\_i;

public function \_\_construct($key)

{

$this->\_i = $key;

}

public function getKey()

{

return $this->\_i;

}

}

class Heap

{

private $heap\_Array;

private $\_current\_Size;

public function \_\_construct()

{

$heap\_Array = array();

$this->\_current\_Size = 0;

}

// Remove item with max key

public function remove()

{

$root = $this->heap\_Array[0];

// put last element into root

$this->heap\_Array[0] = $this->heap\_Array[--$this->\_current\_Size];

$this->bubbleDown(0);

return $root;

}

// Shift process

public function bubbleDown($index)

{

$larger\_Child = null;

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

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

$leftChild = 2 \* $index + 1;

$rightChild = $leftChild + 1;

// find larger child

if ($rightChild < $this->\_current\_Size

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

{

$larger\_Child = $rightChild;

} else {

$larger\_Child = $leftChild;

}

if ($top->getKey() >= $this->heap\_Array[$larger\_Child]->getKey()) {

break;

}

// shift child up

$this->heap\_Array[$index] = $this->heap\_Array[$larger\_Child];

$index = $larger\_Child; // go down

}

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

}

public function insertAt($index, Node $newNode)

{

$this->heap\_Array[$index] = $newNode;

}

public function incrementSize()

{

$this->\_current\_Size++;

}

public function getSize()

{

return $this->\_current\_Size;

}

public function asArray()

{

$arr = array();

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

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

}

return $arr;

}

}

function heapsort(Heap $Heap)

{

$size = $Heap->getSize();

// "sift" all nodes, except lowest level as it has no children

for ($j = (int)($size/2) - 1; $j >= 0; $j--)

{

$Heap->bubbleDown($j);

}

// 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();

}

for ($i=0; $i<=20; $i++) {

$arr[$i] = $i;

}

echo "Original Array : ";

echo implode(', ',$arr);

$Heap = new Heap();

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

$Node = new Node($val);

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

$Heap->incrementSize();

}

$result = heapsort($Heap);

echo "\nSorted Array : ";

echo implode(', ',$result)."\n";

$sure\_bitimi = microtime(true);

$sure = $sure\_bitimi - $sure\_baslangici;

echo "<br>Bekleme süresi: $sure saniye.\n";

//PHP kodlarına ayrılan belleğin miktarını bayt cinsinden döndürür.

echo 'Hafıza kullanımı: ',round(memory\_get\_peak\_usage()/1048576, 2), 'MB';

?>

<?php

$sure\_baslangici = microtime(true);

class Node

{

private $\_i;

public function \_\_construct($key)

{

$this->\_i = $key;

}

public function getKey()

{

return $this->\_i;

}

}

class Heap

{

private $heap\_Array;

private $\_current\_Size;

public function \_\_construct()

{

$heap\_Array = array();

$this->\_current\_Size = 0;

}

// Remove item with max key

public function remove()

{

$root = $this->heap\_Array[0];

// put last element into root

$this->heap\_Array[0] = $this->heap\_Array[--$this->\_current\_Size];

$this->bubbleDown(0);

return $root;

}

// Shift process

public function bubbleDown($index)

{

$larger\_Child = null;

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

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

$leftChild = 2 \* $index + 1;

$rightChild = $leftChild + 1;

// find larger child

if ($rightChild < $this->\_current\_Size

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

{

$larger\_Child = $rightChild;

} else {

$larger\_Child = $leftChild;

}

if ($top->getKey() >= $this->heap\_Array[$larger\_Child]->getKey()) {

break;

}

// shift child up

$this->heap\_Array[$index] = $this->heap\_Array[$larger\_Child];

$index = $larger\_Child; // go down

}

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

}

public function insertAt($index, Node $newNode)

{

$this->heap\_Array[$index] = $newNode;

}

public function incrementSize()

{

$this->\_current\_Size++;

}

public function getSize()

{

return $this->\_current\_Size;

}

public function asArray()

{

$arr = array();

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

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

}

return $arr;

}

}

function heapsort(Heap $Heap)

{

$size = $Heap->getSize();

// "sift" all nodes, except lowest level as it has no children

for ($j = (int)($size/2) - 1; $j >= 0; $j--)

{

$Heap->bubbleDown($j);

}

// 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();

}

for ($i=0; $i<=20; $i++) {

$arr[$i] = $i;

}

echo "Original Array : ";

echo implode(', ',$arr);

$Heap = new Heap();

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

$Node = new Node($val);

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

$Heap->incrementSize();

}

$result = heapsort($Heap);

echo "\nSorted Array : ";

echo implode(', ',$result)."\n";

$sure\_bitimi = microtime(true);

$sure = $sure\_bitimi - $sure\_baslangici;

echo "<br>Bekleme süresi: $sure saniye.\n";

//PHP kodlarına ayrılan belleğin miktarını bayt cinsinden döndürür.

echo 'Hafıza kullanımı: ',round(memory\_get\_peak\_usage()/1048576, 2), 'MB';

?>