**Array Sorting Algorithms In PHP**

There are many ways to sort an array in PHP, the easiest being to use the sort() function built into PHP. This sort function is quick but has its limitations, especially when sorting things like dates as PHP usually guesses which value is higher than the other and can produce odd results. However, there are plenty of sorting algorithms available than can allow you to sort an array in any way you want.

The simplest of these is called the bubble sort. Here is a function that will sort an array of values using the bubble sort algorithm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Function** bubbleSort($array)  {   if(!$length=[count](http://www.php.net/count)($array)){    return$array;   }   for($outer=0;$outer<$length;$outer++){    for($inner=0;$inner<$length;$inner++){     if($array[$outer]<$array[$inner]){      $tmp=$array[$outer];      $array[$outer]=$array[$inner];      $array[$inner]=$tmp;     }    }   }  } |

This algorithm works by running through the array and swapping a value for the next value along if that value is less than the current value. After the first run throughthe highest value in the array will be at the correct end. It therefore must run through the array once for every item in the array, so it has a low efficiency.

An improvement on this is the bidirectional bubble sort, in which the items are run through twice at the same time, one going from top to bottom and one going from bottom to top. The following code is an example of a bidirectional bubble sort with an added level of efficiency. This function assumes that after one iteration through the array the first and last elements are in the correct place. It therefore looks at the array minus these two values.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **function** bidirectionalBubbleSort($array){   if(!$length=[count](http://www.php.net/count)($array)){    return$array;   }   $start=-1;   while($start<  $length){    ++$start;    --$length;    for($i=$start;$i<  $length;++$i){     if($array[$i]>$array[$i+1]){      $temp=$array[$i];      $array[$i]=$array[$i+1];      $array[$i+1]=$temp;     }    }    for($i=$length;--$i>=$start;){     if($array[$i]>$array[$i+1]){      $temp=$array[$i];      $array[$i]=$array[$i+1];      $array[$i+1]=$temp;     }    }   }  } |

However, this still isn't that efficient. To get another level of efficiency you would need to use a shell short. Shell shortworks on a "divide and conquer" technique where groups of the array are looked at and sorted individually. Here is an example function.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **Function** shellSort($array)  {   if(!$length=[count](http://www.php.net/count)($array)){    return$array;   }   $k=0;   $gap[0]=(int)($length/2);   while($gap[$k]>1){    $k++;    $gap[$k]=(int)($gap[$k-1]/2);   }     for($i=0;$i<=$k;$i++){    $step=$gap[$i];    for($j=$step;$j<$length;$j++){     $temp=$array[$j];     $p=$j-$step;     while($p>=0&&$temp<$array[$p]){      $array[$p+$step]=$array[$p];      $p=$p-$step;     }     $array[$p+$step]=$temp;    }   }   return$array;  } |

Although this doesn't look like it is efficient it depends on the data you are sorting. In a best case scenario the data will be randomly placed throughout the array and the algorithm will therefore have a good efficiency. Worst case is when all of the data is sorted in the wrong direction before you try to sort it. However, it is worth using this function unless you know for sure that the worst case will happen all of the time. Tests with random number arrays show that this algorithm is a good choice over the bubble sorts.

I should mention another algorithm here called a quick sort. The function works by splitting the array into smaller and smaller pieces eventually merging the array back together again at the end.It does this by first finding a middle point and then spitting the array depending on if the current value is higher or lower than the middle value. It then recursively calls itself in order to do the same to each section of the array. Here is an example of a quick sort:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **Function** quickSort($array)  {   if(!$length=[count](http://www.php.net/count)($array)){    return$array;   }     $k=$array[0];   $x=$y=[array](http://www.php.net/array)();     for($i=1;$i<$length;$i++){    if($array[$i]<=$k){     $x[]=$array[$i];    }else{     $y[]=$array[$i];    }   }   return[array\_merge](http://www.php.net/array_merge)(quickSort($x),[array](http://www.php.net/array)($k),quickSort($y));  } |

As the name suggests it is a fast way of sorting, and it would be if we were not using PHP. Although this function is very much quicker in Java or C, it is a very slow function in PHP. This is because PHP doesn't handle recursion all that well. In fact when trying to test this function it either timed out the script, or simply gave the following memory error.

Fatal error: Allowed memory size of 134217728 bytes exhausted (tried to allocate 35 bytes) in C:\htdocs\test.php on line 130

This occurred when testing an array length of just 50 and happens because every time PHP recursively calls a function it adds that information to the call stack. This error happens because that stack is full. So although quick sort is fast in other languages you should probably stick to a shell sort or something similar. At the very least you will need to use a function that doesn't use recursion.

If you fancy writing your own sorting functions then you can use the following function to check your work.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **Function** checkSort($array)  {   if(!$length=[count](http://www.php.net/count)($array)){    return**true**;   }   for($i=0;$i<$length;$i++){    if([isset](http://www.php.net/isset)($array[$i+1])){     if($array[$i]>$array[$i+1]){      return**false**;     }    }   }   return**true**;  } |