# PHP - Recursive Functions

A recursive function is such a function that calls itself until a certain condition is satisfied. In PHP, it is possible to defines a recursive function.

- Recursion is used when a certain problem is defined in terms of itself.
- Sometimes, it can be tedious to solve a problem using iterative approach. Recursive approach provides a very concise solution to seemingly complex problems.
- Recursion in PHP is very similar to the one in C and C++.
- Recursive functions are particularly used in traversing nested data structures, and searching or sorting algorithms.
- Binary tree traversal, heap sort and finding shortest route are some of the cases where recursion is used.

## Calculation of Factorial using Recursion

The most popular example of recursion is calculation of factorial. Mathematically factorial is defined as −

```
n! = n × (n-1)!
```

It can be seen that we use factorial itself to define factorial. Hence, this is a fit case to write a recursive function.

Let us expand the above definition for calculation of factorial value of 5

```
5! = 5 × 4!
    5 × 4 × 3!
    5 × 4 × 3 × 2!
    5 × 4 × 3 ×  2 × 1!
    5 × 4 × 3 ×  2 × 1
  = 120
```

While we can perform this calculation using a loop, its recursive function involves successively calling it by decrementing the number till it reaches 1.

## Example

Following is the recursive function to calculate factorial.

```php
<?php
   function factorial ($n) {
      if ($n == 1) {
         echo $n . PHP_EOL;
         return 1;
      } else {
         echo "$n * ";
         return $n*factorial($n-1);
      }
   }
   echo "Factorial of 5 = " . factorial(5);
?>
```

Open Compiler

It will produce the following **output** –

```
5 * 4 * 3 * 2 * 1
Factorial of 5 = 120
```

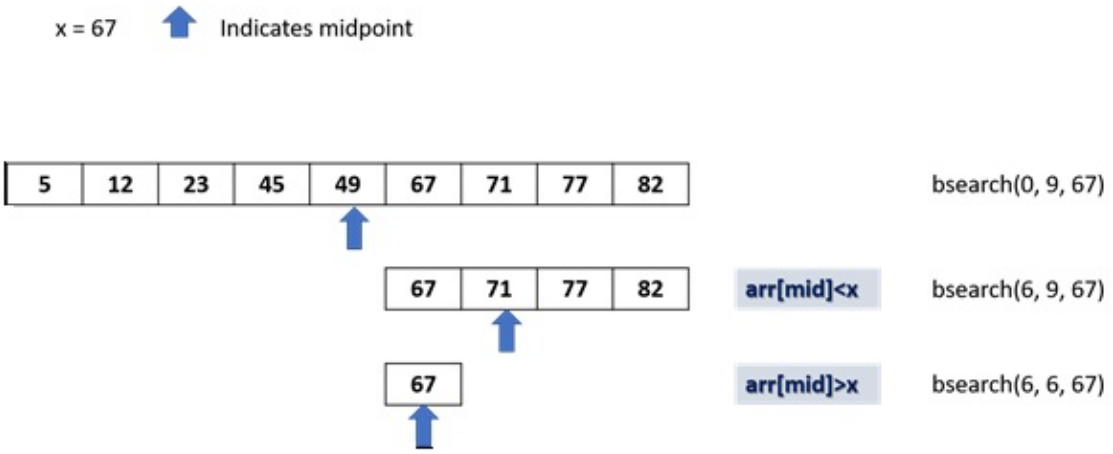
## Binary Search using Recursion

Let us have a look at another example to understand how recursion works. The problem at hand is to check whether a given number is present in a list.

While we can perform a sequential search for a certain number in the list using a **for** loop and comparing each number, the sequential search is not efficient especially if the list is too large. Here, we can use the binary search algorithm that checks if the index 'high' is greater than index 'low. Based on the value present at 'mid' variable, the function is called again to search for the element.

We have a list of numbers, arranged in ascending order. Then, we find the midpoint of the list and restrict the checking to either left or right of midpoint depending on whether the desired number is less than or greater than the number at midpoint.

The following diagram shows how binary search works –




## Example

The following code implements the recursive binary searching technique –

</> Open Compiler

```
<?php
   function bsearch($my_list, $low, $high, $elem) {
      if ($high >= $low) {
         $mid = intval(($high + $low)/2);

         if ($my_list[$mid] == $elem)
         return $mid;

         elseif ($my_list[$mid] > $elem)
         return bsearch($my_list, $low, $mid - 1, $elem);

         else
         return bsearch($my_list, $mid + 1, $high, $elem);
      }
      else
      return -1;
   }

   $list = [5,12,23, 45, 49, 67, 71, 77, 82];
   $num = 67;
   $result = bsearch($list,0,count($list)-1, $num);
   if ($result != -1)
   echo " Number $num found at index " . $result;
   else
```

```
   echo "Element not found!";
?>
```

It will produce the following **output** –

```
Number 67 found at index 5
```

You can check the output for different numbers present in the given list, as well as those which are not present in the list.