

Here we demonstrate how algorithms or pseudocode can be typeset using the `algorithm` environment provided by the `algorithm2e` package.

You should not load the `algorithm`, `algpseudocode`, `algcompatible`, `algorithmic packages` if you have already loaded `algorithm2e`.

Note that the command and argument syntax provided by `algorithm2e` are very different from those provided by `algpseudocode`. It is important to know clearly which package that you are using, and then accordingly write the relevant commands with the correct syntax.

```

 $i \leftarrow 10;$ 
if  $i \geq 5$  then
|    $i \leftarrow i - 1;$ 
else
|   if  $i \leq 3$  then
|   |    $i \leftarrow i + 2;$ 
|   end
end

```

Every line in your source code **must** end with `\;` otherwise your algorithm will continue on the same line of text in the output. Only lines with a macro beginning a block should not end with `\;`.

The above algorithm example is uncaptioned. If you need a caption for your algorithm, use `\caption{...}` inside the `algorithm` environment. You can then use `\label{...}` after the `\caption` so that the algorithm number can be cross-referenced, e.g. Algorithm ?? and ??.

By default, the `plain` algorithm style is used. But if you prefer lines around the algorithm and caption, you can add the `ruled` package option when loading `algorithm2e`, or write `\RestyleAlgo{ruled}` in your document.

The `algorithm2e` package also provides a `\listofalgorithms` command that works like `\listoffigures`, but for captioned algorithms:

List of Algorithms

1	An algorithm with caption	1
2	Bubble Sort Algorithm	2
3	Binary Search Algorithm	2
4	Merge Sort Algorithm	3
5	Find the Largest Number in an Array	4
6	Quick Sort	4
7	Fibonacci Sequence	4
8	Riemann Zeta Sequence	5

Algorithm 1: An algorithm with caption

Data: $n \geq 0$

Result: $y = x^n$

$y \leftarrow 1;$

$X \leftarrow x;$

$N \leftarrow n;$

while $N \neq 0$ **do**

if N is even **then**

$X \leftarrow X \times X;$

$N \leftarrow \frac{N}{2};$

/* This is a comment */

else

if N is odd **then**

$y \leftarrow y \times X;$

$N \leftarrow N - 1;$

end

end

end

Algorithm 2: Bubble Sort Algorithm

Data: An array A of n elements

Result: The array A sorted in non-decreasing order

for $i \leftarrow 0$ **to** $n - 1$ **do**

for $j \leftarrow 0$ **to** $n - i - 1$ **do**

if $A[j] > A[j + 1]$ **then**

 swap $A[j]$ and $A[j + 1];$

end

end

end

Algorithm 3: Binary Search Algorithm

Data: An array A of n elements sorted in non-decreasing order, and a search key x

Result: The index of x in A , or -1 if x is not found

$$low \leftarrow 0;$$
$$high \leftarrow n - 1;$$
while $low \leq high$ **do**

```
mid ←  $\left\lceil \frac{low+high}{2} \right\rceil$  ;           /* Compute the midpoint */
```

```

if  $A[mid] = x$  then
    | return  $mid$  ;                               /* Found  $x$  at index  $mid$  */

```

end

```

if  $A[mid] < x$  then
    |  $low \leftarrow mid + 1$  ;           /*  $x$  must be in the right half */

```

end

```

else
     $high \leftarrow mid - 1$ ;           /*  $x$  must be in the left half */

```

end

end

```
return -1 ;                               /* x is not in the array */
```

Algorithm 4: Merge Sort Algorithm

Data: An array A of n elements

Result: The array A sorted in non-decreasing order

if $n > 1$ then

```
mid ← ⌊ $\frac{n}{2}$ ⌋;                                /* Find the middle index */
```

$L \leftarrow$ copy of the left half of A from index 0 to $mid - 1$;

$R \leftarrow$ copy of the right half of A from index mid to $n - 1$;

```
/* Recursively sort the left and right halves */
```

MergeSort(L);MergeSort(R);

```
/* Merge the sorted halves back into A */
```

$$i \leftarrow 0;$$
$$j \leftarrow 0;$$
$$k \leftarrow 0;$$

```

while  $i < \text{lengthof} L$  and  $j < \text{lengthof} R$  do

```

if $L[i] \leq R[j]$ then

$$A[k] \leftarrow L[i];$$
$$i \leftarrow i + 1;$$

end

else

$$A[k] \leftarrow R[j];$$
$$j \leftarrow j + 1;$$

end

$$k \leftarrow k + 1;$$

end

```
/* Copy any remaining elements of  $L$  and  $R$  into  $A$  */
```

```

while  $i < \text{lengthof } L$  do

```

$$A[k] \leftarrow L[i];$$
$$i \leftarrow i + 1;$$
$$k \leftarrow k + 1;$$

end

```

while  $j < lengthof R$  do

```

$$A[k] \leftarrow R[j];$$
$$j \leftarrow j + 1;$$
$$k \leftarrow k + 1;$$

end

end

Algorithm 5: Find the Largest Number in an Array

Data: An array A of n elements

Result: The largest number in the array

```
largest  $\leftarrow A[0]$  ; /* Initialize largest with the first element of the array
*/
for  $i \leftarrow 1$  to  $n - 1$  do
    if  $A[i] > largest$  then
        | largest  $\leftarrow A[i]$  ;      /* Update largest if a larger number is found */
    end
end
return largest ;                      /* The largest number in the array */
```

Algorithm 6: Quick Sort

Data: An array A of n elements

Result: The array A sorted in non-decreasing order

```
if  $n \leq 1$  then
    | return  $A$  ;          /* Base case: Array with 1 or 0 elements is already
    | sorted */
end
Select a pivot element  $p$  from  $A$  ; /* Various pivot selection strategies exist
*/
Partition  $A$  into three subarrays:  $L$  (elements less than  $p$ ),  $E$  (elements equal to  $p$ ),
and  $G$  (elements greater than  $p$ )
sortedL  $\leftarrow QuickSort(L)$  ;      /* Recursively sort the smaller elements */
sortedG  $\leftarrow QuickSort(G)$  ;      /* Recursively sort the greater elements */
return Concatenate(sortedL,  $E$ , sortedG) ; /* Concatenate the sorted
subarrays */
```

Algorithm 7: Fibonacci Sequence

Data: An integer n

Result: The n -th Fibonacci number

```
if  $n \leq 1$  then
    | return  $n$  ;          /* Base case: Fibonacci(0) = 0, Fibonacci(1) = 1 */
end
prev  $\leftarrow 0$ 
current  $\leftarrow 1$ 
for  $i \leftarrow 2$  to  $n$  do
    next  $\leftarrow prev + current$ 
    prev  $\leftarrow current$ 
    current  $\leftarrow next$  ;      /* Update values for the next iteration */
end
return current ;              /* The  $n$ -th Fibonacci number */
```

Algorithm 8: Riemann Zeta Sequence

Data: An integer n

Result: The n -th term in the Riemann Zeta sequence

```

zeta ← 0
for  $k \leftarrow 1$  to  $n$  do

```

```

    for k ← 1 to n do
        | zeta ← zeta + 1/n : /* Calculate the sum of the sequence */

```

```

end

```

```
return zeta;          /* The n-th term in the Riemann Zeta sequence */
```
