

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	2
2	Bubble Sort Algorithm	2
3	Binary Search Algorithm	3
4	Merge Sort Algorithm	4
5	Bresenham Line Drawing Algorithm	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 ← mid + 1 ;           /* x must be in the right half */
```

end

else

```
high ← mid - 1;           /* x must be in the left half */
```

end

end

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

```
/* Compute the midpoint */
```

```
/* Found  $x$  at index  $mid$  */
```

```
/* x must be in the right half */
```

```
/* x must be in the left half */
```

```
/* 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
     $mid \leftarrow \lfloor \frac{n}{2} \rfloor$ ;                                /* 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  $j < \text{lengthof } R$  do
         $A[k] \leftarrow R[j]$ ;
         $j \leftarrow j + 1$ ;
         $k \leftarrow k + 1$ ;
    end
end
end
```

Algorithm 5: Bresenham Line Drawing Algorithm

Data: Coordinates of two endpoints: (x_1, y_1) and (x_2, y_2)

Result: List of points on the line between (x_1, y_1) and (x_2, y_2)

$\Delta x \leftarrow x_2 - x_1$

$\Delta y \leftarrow y_2 - y_1$

$dx2 \leftarrow 2 \cdot \Delta x$

$dy2 \leftarrow 2 \cdot \Delta y$

$slopeError \leftarrow 0$

$y \leftarrow y_1$

foreach x in range x_1 to x_2 **do**

 Plot point (x, y) ;

 /* Plot the current point */

$slopeError \leftarrow slopeError + dy2$

if $slopeError \geq \Delta x$ **then**

$y \leftarrow y + 1$;

 /* Increment y for diagonal movement */

$slopeError \leftarrow slopeError - dx2$

end

end
