```
  quicksort(a, 0, N - 1);

  printf("In sorted order: ");
  for (i = 0; i < N; i++)
    printf("%d ", a[i]);
  printf("\n");

  return 0;
}

void quicksort(int a[], int low, int high)
{
  int middle;

  if (low >= high) return;
  middle = split(a, low, high);
  quicksort(a, low, middle - 1);
  quicksort(a, middle + 1, high);
}

int split(int a[], int low, int high)
{
  int part_element = a[low];

  for (;;) {
    while (low < high && part_element <= a[high])
      high--;
    if (low >= high) break;
    a[low++] = a[high];

    while (low < high && a[low] <= part_element)
      low++;
    if (low >= high) break;
    a[high--] = a[low];
  }

  a[high] = part_element;
  return high;
}
```

Although this version of Quicksort works, it's not the best. There are numerous ways to improve the program's performance, including:

- *Improving the partitioning algorithm.* Our method isn't the most efficient. Instead of choosing the first element in the array as the partitioning element, it's better to take the median of the first element, the middle element, and the last element. The partitioning process itself can also be sped up. In particular, it's possible to avoid the `low < high` tests in the two `while` loops.
- *Using a different method to sort small arrays.* Instead of using Quicksort recursively all the way down to arrays with one element, it's better to use a simpler method for small arrays (those with fewer than, say, 25 elements).