

How to Quick Sort an Array in C++

Sorting is a very useful tool in programming. It is often necessary to arrange the members of a list in ascending or descending order. A sorted list allows a user to search and find information very quickly. Sorting a list requires the...

Part 1 of 2:

Creating the quickSort Function

```
// Sorts elements using the middle index as the pivot
void quickSort(int *arr, int left, int right) {
} //end quickSort
```

1.

wikiHow

Create the quickSort function. This is a recursive `void` function. It requires three parameters:

1. The `array` (an `int array`)
2. The `left` bound (an `int` variable)
3. The `right` bound (an `int` variable; the size of the `array` subtracted by 1)

```
// Sorts elements using the middle index as the pivot
void quickSort(int *arr, int left, int right) {
    int i = left;
    int j = right;
    int temp;
    int pivot = arr[(left + right) / 2];
} //end quickSort
```

2.

Create the variables. These variables will be used to go through the list and to swap the values. Four variables are needed:

1. An `int i` (the left bound)
2. An `int j` (the right bound)
3. An `int temp` (a temporary variable used for swapping without losing any data)
4. An `int pivot` (the value of the middle point that splits the list to make it easier to sort)

```
// Sorts elements using the middle index as the pivot
void quickSort(int *arr, int left, int right) {
    int i = left;
    int j = right;
    int temp;
    int pivot = arr[(left + right) / 2];
    // Begin the loop for the quicksort
    while (i <= j) {
    } //end while
} //end quickSort
```

3.

Create a `while` loop to begin sorting. A loop `while i ? j` is used to go through the indexes of the list. These values will be changed as the sublists that are being sorted change.

```
int i = left;
int j = right;
int temp;
int pivot = arr[(left + right) / 2];

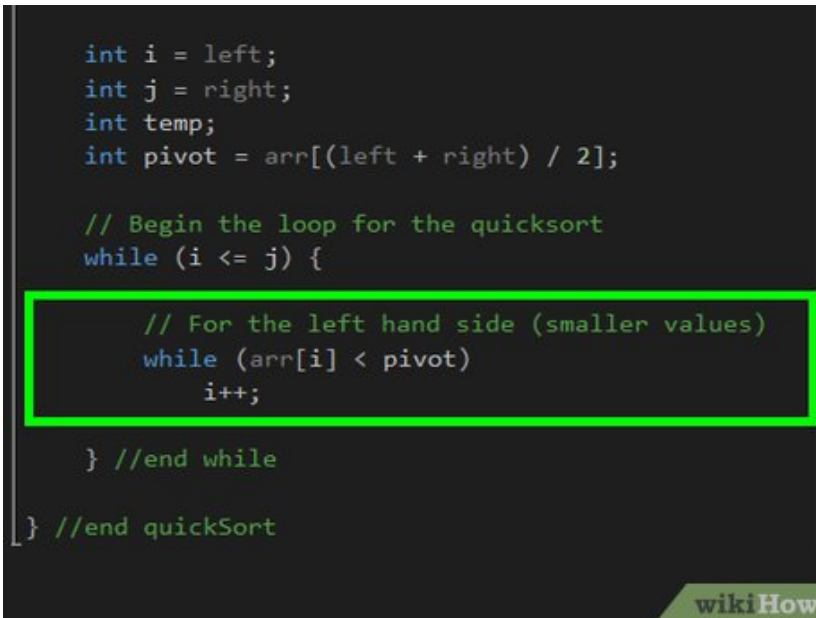
// Begin the loop for the quicksort
while (i <= j) {

    // For the left hand side (smaller values)
    while (arr[i] < pivot)
        i++;

} //end while

} //end quickSort
```

4.



wikiHow

Iterate through the left side. Another `while` loop checking if the element is less than `pivot` iterates through the list. If it is less than `pivot` value, increase `i` by 1. This checks if the left side of the sublist needs to be sorted.

```
int j = right;
int temp;
int pivot = arr[(left + right) / 2];

// Begin the loop for the quicksort
while (i <= j) {

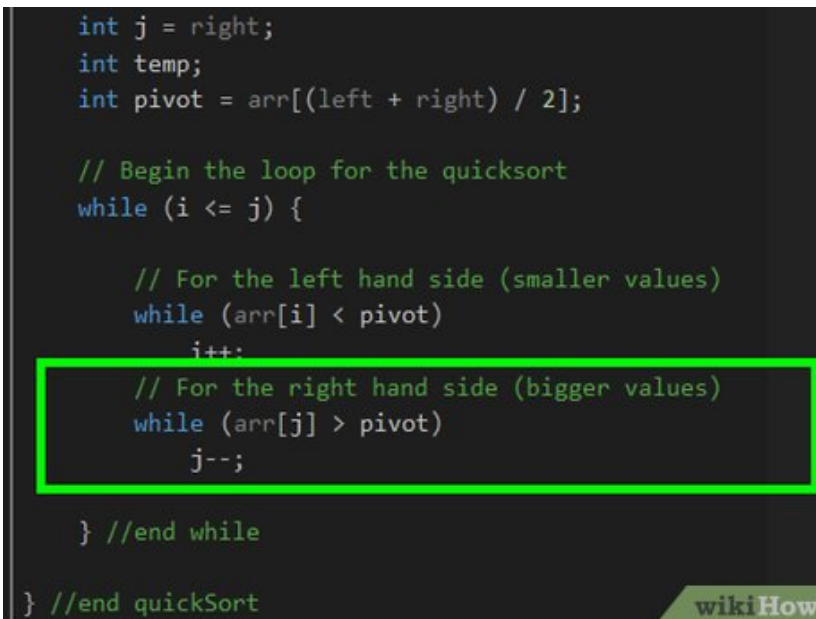
    // For the left hand side (smaller values)
    while (arr[i] < pivot)
        i++;

    // For the right hand side (bigger values)
    while (arr[j] > pivot)
        j--;

} //end while

} //end quickSort
```

5.



wikiHow

Iterate through the right side. Another `while` loop checking if the element is greater than `pivot` iterates through the list. If it is greater than `pivot`, decrease `j` by 1. This checks if the right side of the sublist needs to be sorted.

```

// Begin the loop for the quicksort
while (i <= j) {

    // For the left hand side (smaller values)
    while (arr[i] < pivot)
        i++;
    // For the right hand side (bigger values)
    while (arr[j] > pivot)
        j--;

    // Begins the swap
    if (i <= j) {
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        i++;
        j--;
    } //end if

} //end while
} //end quickSort

```

6.

Begin swapping the values if `i > j`. Swapping the values of the list puts the values in ascending order. Assigning one value to another without a temporary variable will result in a loss of data. To avoid this, this procedure is used:

1. Assign the value of the list at index `i` to `temp`.
2. Assign the value of the list at index `j` to the list at index `i`.
3. Assign `temp` to the list at index `j`.
4. Add 1 to `i`.
5. Subtract 1 from `j`.

```

// Begins the swap
if (i <= j) {
    temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++;
    j--;
} //end if

} //end while

// Check if we need to keep sorting the left side
if (left < j)
    quickSort(arr, left, j);

// Check if we need to keep sorting the right side
if (i < right)
    quickSort(arr, i, right);

} //end quickSort

```

7.

Check if each half of the list is sorted. This is done by two recursive calls. The first function call sorts the left sublist created by changing the bounds. When the left side is completely sorted, the next recursive call sorts the right sublist by changing its bounds.

1. If `left < j`, call the function with `left` and `i` as the bounds.

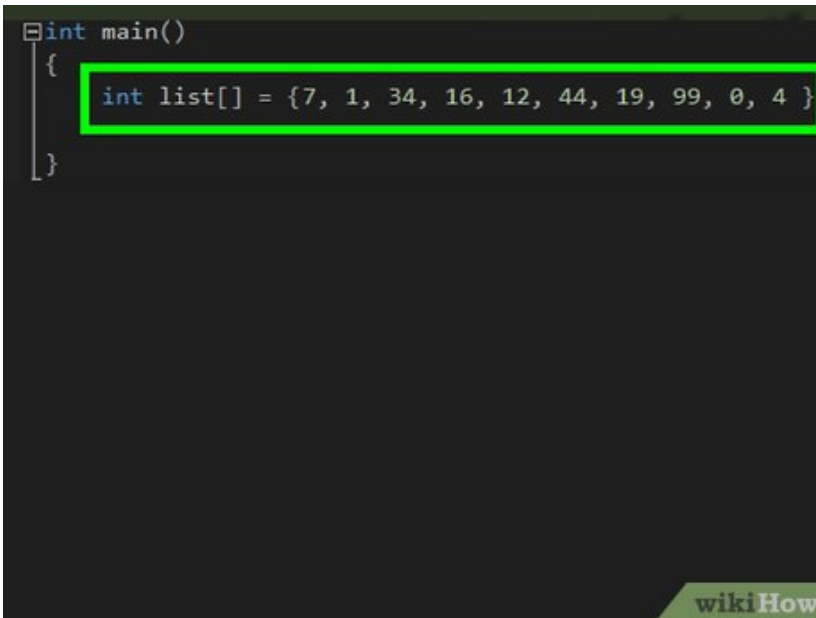
2. If `right i`, call the function with `i` and `right` as the bounds.

Part 2 of 2:

Testing the quickSort Function

1.

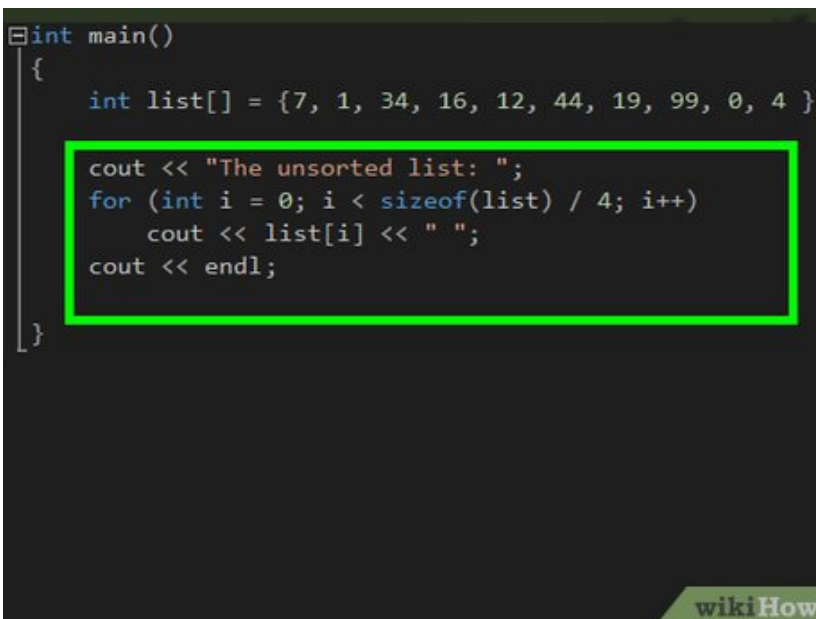
```
int main()
{
    int list[] = {7, 1, 34, 16, 12, 44, 19, 99, 0, 4 };
}
```



Create the `list` in the `main` function. The array can be any size and can be initialized both explicitly and through other methods.

2.

```
int main()
{
    int list[] = {7, 1, 34, 16, 12, 44, 19, 99, 0, 4 };
    cout << "The unsorted list: ";
    for (int i = 0; i < sizeof(list) / 4; i++)
        cout << list[i] << " ";
    cout << endl;
}
```



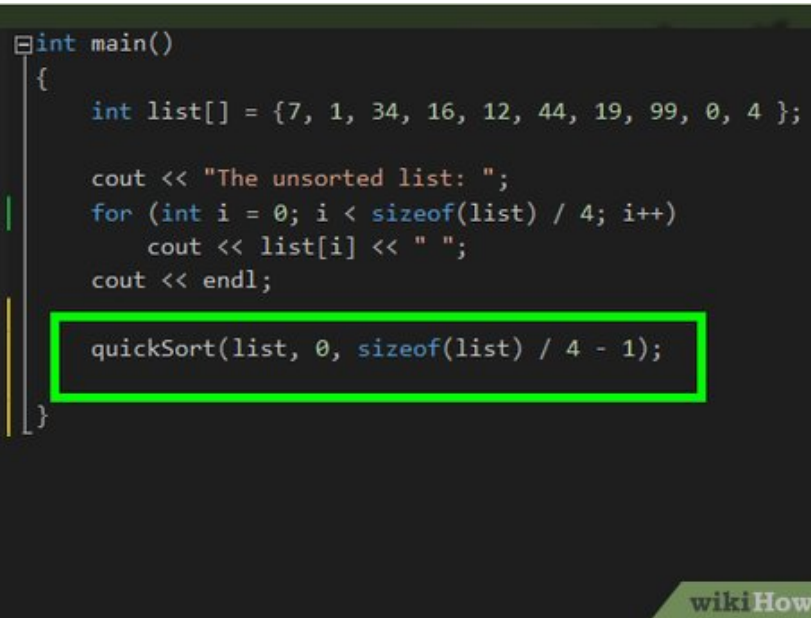
Output the unsorted `list` using a `for-loop`. The bounds of the loop go from 0 to the `sizeof(list) / 4`. This piece of code gives the number of elements in `list`.

```
int main()
{
    int list[] = {7, 1, 34, 16, 12, 44, 19, 99, 0, 4 };

    cout << "The unsorted list: ";
    for (int i = 0; i < sizeof(list) / 4; i++)
        cout << list[i] << " ";
    cout << endl;

    quickSort(list, 0, sizeof(list) / 4 - 1);
}
```

3.



Call the quickSort function. The three needed parameters are:

1. The list
2. The left bound (0)
3. The right bound (the size of the array subtracted by 1)

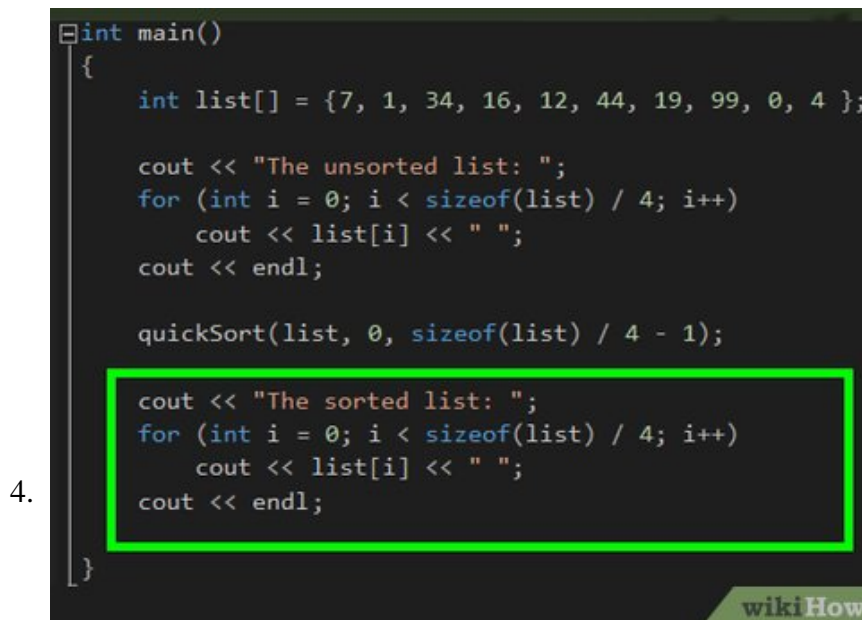
```
int main()
{
    int list[] = {7, 1, 34, 16, 12, 44, 19, 99, 0, 4 };

    cout << "The unsorted list: ";
    for (int i = 0; i < sizeof(list) / 4; i++)
        cout << list[i] << " ";
    cout << endl;

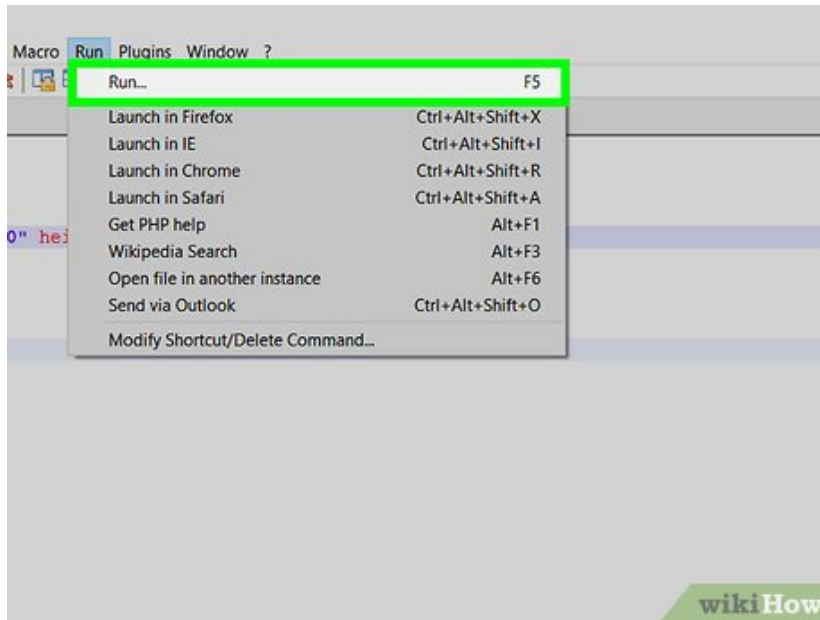
    quickSort(list, 0, sizeof(list) / 4 - 1);

    cout << "The sorted list: ";
    for (int i = 0; i < sizeof(list) / 4; i++)
        cout << list[i] << " ";
    cout << endl;
}
```

4.



Output the new list using a for-loop. Again, the bounds of the loop go from 0 to the sizeof(list)/4. This is because the sorted list contains the same amount of elements as the unsorted list (no data was lost).



5.

Run the program to see the sorted list. The number of items in `list` should be the same in both lists.

You finished reading the article "**How to Quick Sort an Array in C++**" edited by the [TipsMake](#) team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.