Merge Sort in C++ (Source Code)

This code demonstrates merge sort implemented in C++. Notice the use of a wrapper function to dynamically allocate the requisite scratch space.

Back to the merge sort tutorial

```
/* Helper function for finding the max of two numbers */
int max(int x, int y)
{
    if(x > y)
    {
        return x;
    }
    else
    {
        return y;
    }
}

/* left is the index of the leftmost element of the subarray; right is one
 * past the index of the rightmost element */
void merge_helper(int *input, int left, int right, int *scratch)
{
    /* base case: one element */
    if(right == left + 1)
    {
        return;
    }
    else
    {
        int i = 0;
        int length = right - left;
        int midpoint_distance = length/2;
        /* l and r are to the positions in the left and right subarrays */
        int l = left, r = left + midpoint_distance;

        /* sort each subarray */
        merge_helper(input, left, left + midpoint_distance, scratch);
        merge_helper(input, left + midpoint_distance, right, scratch);

        /* merge the arrays together using scratch for temporary storage */
        for(i = 0; i < length; i++)
        {
            /* Check to see if any elements remain in the left array; if so,
             * we check if there are any elements left in the right array; if
             * so, we compare them.  Otherwise, we know that the merge must
             * use take the element from the left array */
            if(l < left + midpoint_distance &&
                    (r == right || max(input[l], input[r]) == input[l]))
            {
                scratch[i] = input[l];
                l++;
            }
            else
            {
                scratch[i] = input[r];
                r++;
            }
        }
        /* Copy the sorted subarray back to the input */
        for(i = left; i < right; i++)
        {
            input[i] = scratch[i - left];
        }
    }
}

/* mergesort returns true on success.  Note that in C++, you could also
 * replace malloc with new and if memory allocation fails, an exception will
 * be thrown.  If we don't allocate a scratch array here, what happens?
 *
 * Elements are sorted in reverse order -- greatest to least */

int mergesort(int *input, int size)
{
    int *scratch = (int *)malloc(size * sizeof(int));
    if(scratch != NULL)
    {
        merge_helper(input, 0, size, scratch);
        free(scratch);
        return 1;
    }
    else
    {
        return 0;
    }
```

```
}
```