Results

ORIGINAL IMPLEMENTATION
Total run time: 0.60732s
Merge time: 0.45207s

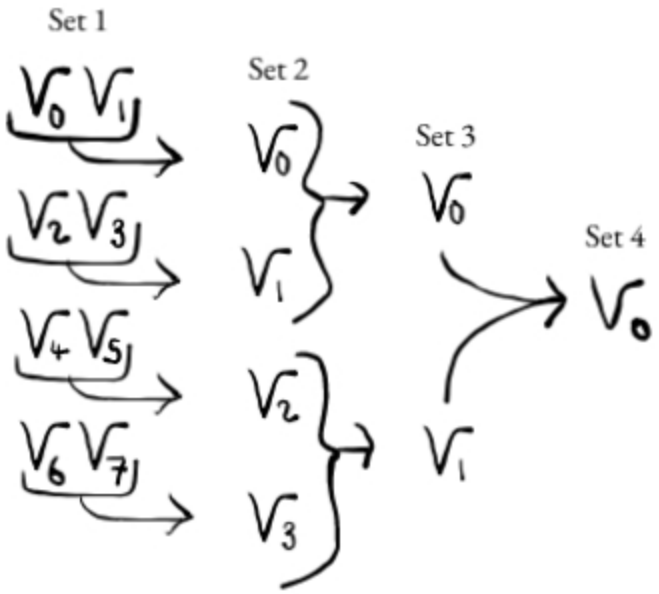
MODIFIED IMPLEMENTATION
Total run time: 6.84429s
Merge time: 6.72097s

Clearly, by the results, using multiple processes in regards to the sorting of subvectors is the optimal approach in comparison to only utilising a single process, despite whether or not those multiple processes have been utilised to presort the initial subvectors (as they have been in our code).

This result is fairly intuitive in that multiple processes may simultaneously merge individual subvectors, yet a single process must handle one merging operation at a time. This is explained in further detail using diagrams.

Each of these arrows represents an iteration through the merge loops.
Using a single process, each arrows must be performed linearly, first completing those in set 1, then set 2, etc.
Using multiple processes, each arrow in a set is handled by several individual processes. In this case, 4 processes, then 2, then 1.
Obviously, using more than one process to simultaneously sort several subvectors is faster than using one, but in the end it is always one process that must merge the final two presorted subvectors. However, by presorting the subvectors this final procedure is much simpler, and hence explains (beyond simply having more processes to put to work) why this procedure is faster on multiple processes compared to a single one.

To explain, take a list of 10 numbers. In order to sort this numerically, the program would look through the list and compare each number with the first element, if it was smaller then it would push this number to the first element and shift the other elements in the list over. It would then check the next number and so on until finished (Assuming an insertion method of sorting). Instead by taking the list and splitting it in two, it takes half the time to order these smaller lists, then when the pre-ordered lists are put together again it is simply a case of removing the smallest number of the first elements of the two lists until no numbers are left in either onet and therefore the remaining elements in the other list must be larger and in order.
A diagram has been provided to help illustrate this



Insertion Sorting Method:

Order of Operations

{9,4,5,8,8,7,4,2,0,1}
{4,9,5,8,8,7,4,2,0,1}
{4,5,9,8,8,7,4,2,0,1}
{4,5,8,9,8,7,4,2,0,1}
{4,5,8,8,9,7,4,2,0,1}
{4,5,7,8,8,9,4,2,0,1}
{4,4,5,7,8,8,9,2,0,1}
{2,4,4,5,7,8,8,9,0,1}
{0,2,4,4,5,7,8,8,9,1}
{0,1,2,4,4,5,7,8,8,9}

Order of Operations

Ordered Subvectors		Resulting Vector
{4,5,8,8,9}	{0,1,2,4,7}	{}
{4,5,8,8,9}	{ 1,2,4,7}	{0}
{4,5,8,8,9}	{2,4,7}	{0,1}
{4,5,8,8,9}	{4,7}	{0,1,2}
{5,8,8,9}	{4,7}	{0,1,2,4}
{5,8,8,9}	{7}	{0,1,2,4,4}
{8,8,9}	{7}	{0,1,2,4,4,5}
{8,8,9}		{0,1,2,4,4,5,7}
		{0,1,2,4,4,5,7,8,8,9}