Write a MIPS assembly language program to \textit{merge sort} an array of integers. The general idea of merge sort is as follows. Assume \( n \) items to sort.

\begin{enumerate}
\item Split the unsorted part in half to get two smaller sorting problems of about equal size \( n/2 \)
\item Solve both smaller problems recursively using merge sort
\item "Merge" the solutions to the smaller problems together to solve the original sorting problem of size \( n \)
\end{enumerate}

You “just” need to translate the following C++ merge sort and merge code to MIPS assembly language. \textbf{HOWEVER, YOU NEED TO FOLLOW THE MIPS REGISTER CONVENTIONS.}

```c++
void mergeSort(int myArray[], int size) {
    int index;
    if (size > 1) { // For size of 0 or 1, nothing to do
        // Determine size of each half
        int leftSize = size / 2;
        int rightSize = size - leftSize;

        // Dynamically allocate arrays big enough to hold each half
        int * leftArray = new int [leftSize]; // NOTE: Use the sbrk syscall
        int * rightArray = new int [rightSize]; // NOTE: Use the sbrk syscall

        // Copy each half to their arrays
        for (index = 0; index < leftSize; index++) {
            leftArray[index] = myArray[index];
        }
        for (index = 0; index < rightSize; index++) {
            rightArray[index] = myArray[leftSize+index];
        }

        // Recursively sort both halves
        mergeSort(leftArray, leftSize);
        mergeSort(rightArray, rightSize);

        // Merge sorted halves back together to complete the sort
        merge(myArray, size, leftArray, leftSize, rightArray, rightSize);

        // Deallocate ("Free up") memory from both array halves
        delete [] leftArray; // NOTE: no good syscall so we waste some memory
        delete [] rightArray;
    }
}
```
void merge(int sortedArray[], int size, int leftArray[], int leftSize, 
        int rightArray[], int rightSize) {
    int left = 0;
    int right = 0;
    int next = 0;

    // Repeatedly copy the next smaller element from either the leftArray or rightArray
    // to the end of the combined sortedArray until one of the smaller arrays runs out of
    // elements.
    while (left < leftSize && right < rightSize) { // NOTE: "&&" means Boolean "AND"
        if (leftArray[left] < rightArray[right]) {
            sortedArray[next] = leftArray[left];
            left++;
        } else {
            sortedArray[next] = rightArray[right];
            right++;
        } // end if
        next++;
    } // end while

    // Copy the remain elements from the leftArray if the rightArray ran out first
    while (left < leftSize) {
        sortedArray[next] = leftArray[left];
        left++;
        next++;
    } // end while

    // Copy the remain elements from the rightArray if the leftArray ran out first
    while (right < rightSize) {
        sortedArray[next] = rightArray[right];
        right++;
        next++;
    } // end while

} // end merge

Run your program on the following array values: 60, 35, 10, 50, 45, 20, 25, 40, 30. Your program should be in a single file organized like:

.data
array:  .word 60,35,10,50,45,20,25,40,30
n:  .word 9

.text
globl main
main:
    # call to mergeSort here
    li $v0, 10  # system call to exit the program
    syscall
merge:
    # merge subprogram here
    jr $ra
mergeSort:
    # mergeSort subprogram here
    jr $ra

You should turn in:
- a print-out of the MIPS assembly language program, e.g., hw7.s from any text-editor (e.g., WordPad)
- a window capture of the PCSpim simulator showing the sorted array, i.e., after running your assembly language program