Lab 5: Merge Sort

Due date: By the end of Thursday, October 10th, 2019.

Important Note:

- **For Lab5, you need to work as a group of 2 or 3 maximum. You better work in a group with your classmates in the same session duration so can work on the assignment together during the lab.**
- **This Lab worth 20 points!**
- **The Labs on Wednesday, October 2nd and October 9th will be on Lab5.**
- **Send your TA an email that contains the names of students of your group by Wednesday, October 2nd.**
- **Only one of your group needs to submit the assignment. Don't forget to write all the student names of your group at the beginning of your assignment as a comment.**

Objectives

You have learned all the moves. Now it is time to practice. In this lab, you will implement a merge sort function to sort 32-bit (signed) integers in an array. You will need to use many concepts you have learned, which include loops, arrays, recursive functions, and local storage on the stack.

Description

Merge sort is a comparison-based sorting algorithm. Its complexity is O($n$lg(n)). Taking divide and conquer approaches, the algorithm divides an array to be sorted into two halves, sorts each half **recursively**, and then merges two sorted arrays. One of the main ideas behind merge sort is that it is easier to merge two sorted lists. The Wikipedia page (https://en.wikipedia.org/wiki/Merge_sort) has a nice description and some animations. The algorithm is also discussed in many algorithm books.

The C-like pseudocode is listed on the next page. It is sufficient for you to complete the lab.

The interface of the function is:

```
void  merge_sort(int p[], int n);
```

The function takes two arguments. The first argument p is the starting address of a word array. The second argument is the number of elements in the array. The function sorts the array into ascending order and places the sorted words back in the array `p`. The function does not return a value.

The main function in the skeleton code initializes a word array with random values, and then calls the merge sort function. You can change the number of words in the array by loading a different value in `$s1`. Remember to change it back to 1024 before submitting your code. The function `check_array` checks if array elements are in ascending order and if any data are corrupted. It is called twice in the main function: once before and once after the sorting. The program should output "Not sorted" and then "Sorted". Your implementation must be **recursive**.

You only need to work on the `merge_sort` function. You may add other helper functions (e.g., copy array). Do not change the main function and other functions provided. To avoid name conflicts, all the labels you add in your code should start with 'L_", e.g., "L_exit" or "l_exit".

Add brief comments in your code.

Deliverables

- Submit revised lab5.s, which has your code and comments, in HuskyCT.
- Write the names of all the students who worked in this assignment as a group at the beginning of the lab5.s as a comment.
- To receive full credits, your code should use proper MIPS instructions/pseudoinstructions for the tasks and follow the MIPS calling conventions.

Appendix: Pseudocode of the merge sort algorithm

```
void   merge_sort(int    p[], int    n) {
        // Allocate space on stack and save registers
        // Let p3 be the starting address of a local array of n words
        if (n < 2)   goto Exit;          // already sorted
        // divide the array into two halves and sort each half recursively
        // p1 and n1 are the starting address and number of words in the first half
        // p2 and n2 are the starting address and number of words in the second half
        n1 = n >> 1;                // n1 = n / 2
        n2 = n - n1;                // the rest is in the second half
        p1 = p;      p2 = p + n1;        // p2 is the address of p[n1].
        merge_sort(p1, n1);              // sort the first half
        merge_sort(p2, n2);              // sort the second half
        // Merge two sorted arrays. Note p3 is a local array of n words
        i1 = i2 = i3 = 0;                // Use array notation. You can try pointers
        while (i1 < n1 and i2 < n2)      {
                // Compare the first element in each half and add the smaller one to p3
                if (p1[i1] < p2[i2]) {
                        p3[i3] = p1[i1]     // add p1[i1] to p3
                        increment i1 and i3
                } else {
                        p3[i3] = p2[i2]    // add p1[i1] to p3
                        increment i2 and i3
                }
        }
```

```
       while (i1 < n1) add p1[i1] to p3 and increment indexes
       while (i2 < n2) add p2[i2] to p3 and increment indexes
       Copy n words from p3 to p
Exit: Restore saved registers and stack
       Return
}
```