Homework 4 – Arrays and Sorting

Objectives:
This exercise aims give you some practice working with arrays and sorting algorithms.

Description:
For this exercise you need to perform the following tasks:

1. Introduce the program.
2. Get a user *request* in the range [*min* = 10 – max = 200].
3. Generate *request* random integers in the range [*lo* = 100 – *hi* = 999] and store them in an array.
4. Display the list of integers before sorting, 10 numbers per line.
5. Sort the list in descending order (i.e. largest number first).
6. Calculate and display the median and the average *integer* values.
7. Display the sorted list, 10 numbers per line.

Requirements:

1. The program should be implemented using procedures. The main procedure should only be used to call other procedures.
2. Functional decomposition flowchart should be provided.
3. The min, max, hi and lo limits should be declared as constants.
4. The parameters for each procedure should be passed by value or by reference on the system stack. Hint: pass arrays by reference and single values (min, max…) by value.
5. When the lists are printed on the console, they should be labeled as "*unsorted*" and "*sorted*".
6. The procedure to print the lists should be one and called twice.
7. All procedures should be documented, and the addressing modes used in the procedure should be identified in the documentation.
8. All the requirements for intro, outro, data validation exist.

Notes:

1. The Irvine library provides the procedures *Randomize* and *RandomRange* for generating random numbers. Check the lecture slides on how to use them.

2. The ***Selection Sort*** is probably the easiest sorting algorithm to implement. Here is a version of the descending order algorithm, where request is the number of array elements being sorted, and exchange is the code to exchange two elements of array:

```
for (k = 0; k < request-1; k++) {
        i = k;
        for (j = k + 1; j < request; j++) {
                if(array[j] > array[i])
                        i = j;
        }
        exchange(array[k], array[i]);
}
```

3. The median is calculated after the array is sorted. It is the "middle" element of the sorted list. If the number of elements is even, the median is the average of the middle two elements (may be rounded)

EXTRA CREDIT (up to 30 points):

1. Display the numbers ordered by column instead of by row. (3 points)
2. Use a recursive sorting algorithm (e.g. Merge Sort, Quick Sort, Heap Sort, etc.) (10 points)
3. Add to the program floating-point functionality. (5 points)
4. Use macro-wrappers instead of the Irvine's procedures. (5 points)
5. Generate the numbers into a file; then read the file into the array. (5 points)
6. Do something extraordinary. (2 points)

-ATTENTION-

To ensure you receive credit for any extra credit options you did, you must add one print statement to your program output PER EXTRA CREDIT which describes the extra credit you chose to work on. You will not receive extra credit points unless you do this. The statement must be formatted as follows,

--Program Intro--
EC: DESCRIPTION

--Program prompts, etc—

What to submit:

1. A functional decomposition flow-chart of the program.
2. The .asm file