MIPS MARS Simulator

-Chinthan Chandra
-Tejas Sharma

We have implemented both bubble and selection sort algorithm to sort N numbers.

Input Format:

⇒ First input is the value of N and it is stored in $t1 i.e. the number of numbers to be sorted.
⇒ Second input is the starting memory location of where the numbers are stored, this value is in $t2.
⇒ Third input is the starting memory location of where the numbers are to be copied for output, this value is in $t3.
⇒ The next N inputs are the numbers which are stored in memory starting from the address in $t2.

Running the program can be in either terminal or the GUI.

- In terminal we have to run the following command

```
java -jar Mars4_5.jar nc IMT2020109_IMT2020548_mips.asm
```

- In the GUI we can open the file, assemble and run it.

C programs which are implemented:

1. Bubble Sort:

```
int i, j;
 for (i = 0; i < n-1; i++)
    for (j = 0; j < n-i-1; j++)
        if (a[j] > a[j+1])
            swap(a[j], a[j+1]);
```

2. Selection Sort:

```
int i, j, min_idx;
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
```

```
            if (a[j] < a[min_idx])
                min_idx = j;
            // Swap the found minimum element with the first element
            swap(a[min_idx],a[i]);
        }
```

The copying function is same for both the sorting algorithms.

.asm Code explanation:

The copying is same for both the codes.
Copying the sorted numbers to output address.

- Starting address of output is copied from register $t3 to $t0.
- Starting address of sorted array is copied from register $t2 to $t4.
- A loop variable is stored in $s0 which runs for $s0<$t0 (n).
- Load word from address $t4 and store it in address $t0.
- Increment address registers by 4, and $so by 1.

Variables i and j are stored in registers $s1 and $s2.
The value n-1 required for loop condition checking is stored in $s0.

Bubble Sort code:

Outer loop runs until $s1<$s0.

- Set j=0 at every iteration.($s2=$zero)
- Copy address of start of numbers to $t4($t4=$t2).
- Set $s3=$s0-$s1 (checker for inner loop) i.e. n-i-1.

Inner loop runs until $s2<$s3 i.e j<n-i-1:

- Load words from a[$t4] and a[$t4+4] into $t5 and $t6.
- Compare $t6<$t5 using slt.
- Increase j($s2) by 1.
- Increase memory address $t4 by 4.
- If the comparison is false jump to start of loop else swap the values at memory by storing $t5 at Mem[$t4] and $t6 at Mem[$t4-4]. (because we increased $t4 by 4 irrespective of the comparison.)
- After swapping jump to inner loop starting.

- Increment $s1 by 1(i+=1).
- Jump to outer loop starting.

We basically compare the adjacent memory addresses and swap them. We do this until all the greater numbers are put to the right.

Selection Sort code:

Store address of start of numbers at $s3 ($s3=$t2)
Outer loop runs until $s1<$s0.

- Set j=0 at every iteration.($s2=$zero)
- Copy next address of $s3+4 to $t4($t4=$s3+4).
- Assuming first number to be minimum i.e. Mem[$s3]=min_address.
- Store this $s3 in $s7 for changing in loop.(address of minimum value)
- Set $s5=$s0-$s1 (checker for inner loop) i.e. n-i-1.

  Inner loop runs until $s2<$s5 i.e j<n-i-1:
  - Load words from a[$s7] and a[$t4] into $t5 and $t6.($t5=a[j] and $t6
  - Compare $t6<$t5 using slt.
  - Increase j($s2) by 1.
  - Increase memory address $t4 by 4.
  - If the comparison is false jump to start of loop else change min_address($s7) to address which stores $t6 value.
  - jump to inner loop starting.
- Increment $s1 by 1(i+=1).
- Load word of first address i.e Mem[$s3] and the word at min_address.
- Swap these both.
- Increase the $s3 by 4(now (i+1)th number in array starts as minimum).
- Jump to outer loop starting.

We keep the first element at the array as minimum element and then compare this with the rest of the array. We update minimum element if we find an elment smaller than this we change the minimum address to this element's address. After going through the loop we finally swap the minimum address with the first element.
We do this n times to sort the array.

This is a step by step explanation, all of these are commented in the program too.
We increase registers which store address of words by 4 because MIPS is byte addressable and each word is 32 bits and the address of memory is 8 bit wide.

But we increment loop variables by 1 as they are not addresses of memory.
We have used registers to help us store and access elements of array and compare them. We change them according to the requirements of the code.

Screenshots:

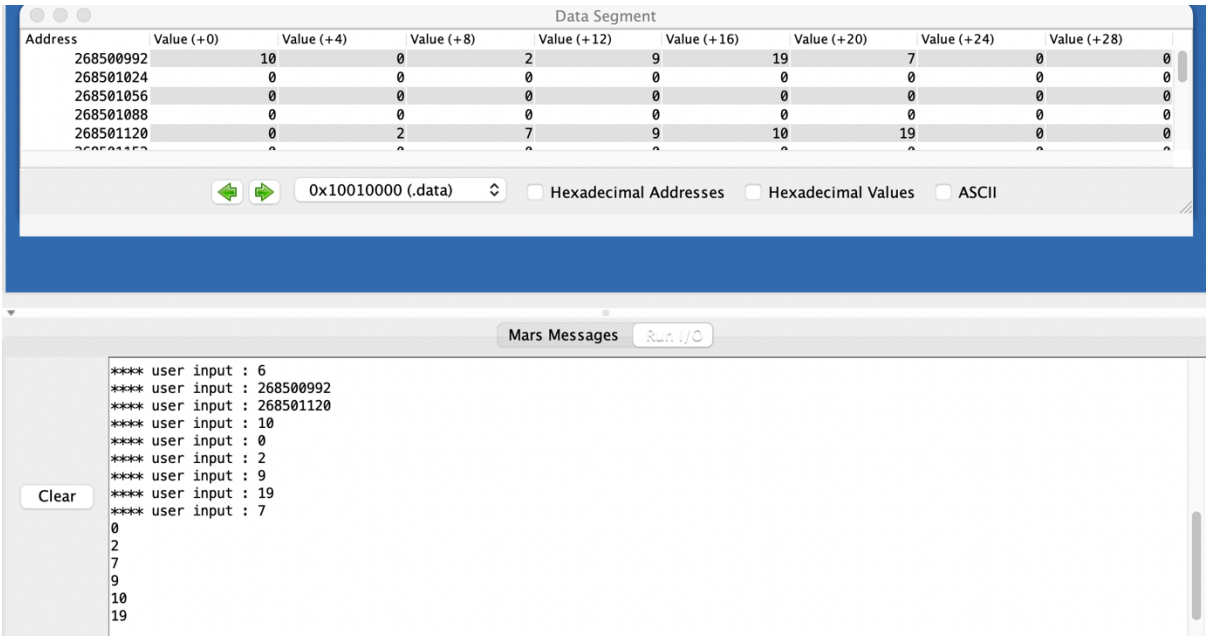
Test 1:

```
chinthanchandra@Chinthans-MacBook-Pro ~ % java -jar Mars4_5.jar nc IMT2020109_IMT2020548_mips.asm
6
268500992
268501120
10
0
2
9
19
7
0
2
7
9
10
19
```

INPUT

OUTPUT

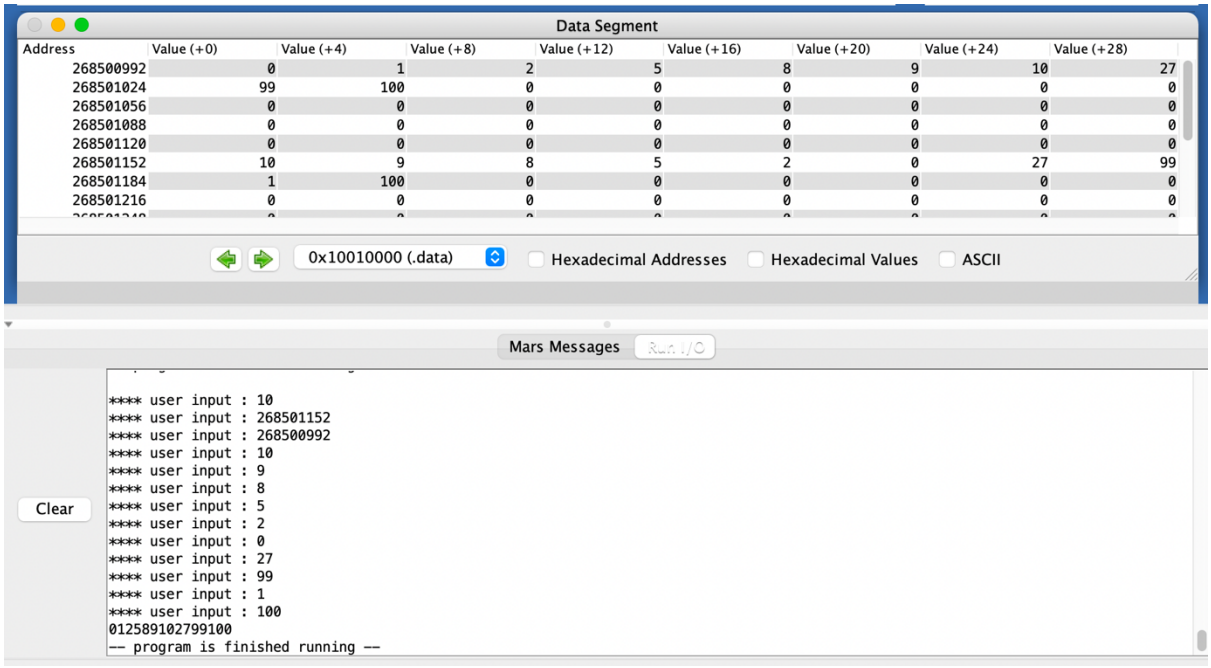
In GUI:



Test 2:

```
chinthanchandra@Chinthans-MacBook-Pro ~ % java -jar Mars4_5.jar nc IMT2020109_IMT2020548_mips.asm
10
268501152
268500992
10
9
8
5
2
0
27
99
1
100
012589102799100
```

In GUI:



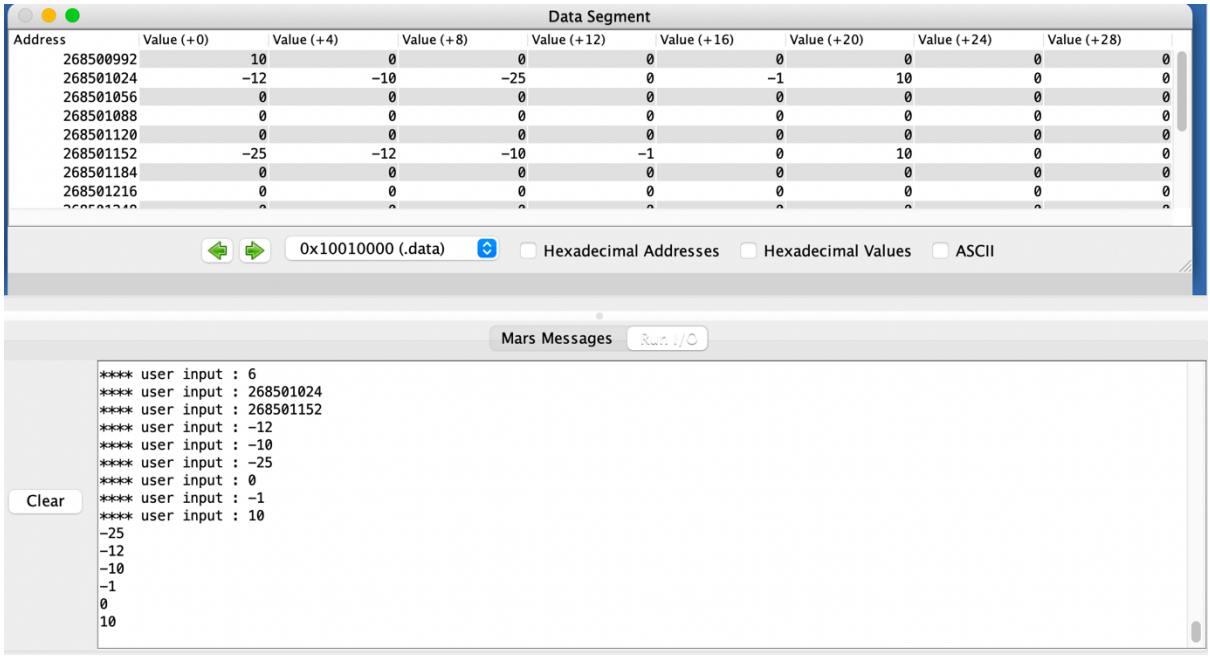
Test 3:

```
chinthanchandra@Chinthans-MacBook-Pro ~ % java -jar Mars4_5.jar nc IMT2020109_IMT2020548_mips.asm
6
268501024
268501152
-12
-10
-25
0
-1
10
-25
-12
-10
-1
0
10
```

INPUT

OUTPUT

In GUI:


Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+12) Value (+16) Value (+20) Value (+24) Value (+28)
268500992 10 0 0 0 0 0 0 0
268501024 -12 -10 -25 0 -1 10 0 0
268501056 0 0 0 0 0 0 0 0
268501088 0 0 0 0 0 0 0 0
268501120 0 0 0 0 0 0 0 0
268501152 -25 -12 -10 -1 0 10 0 0
268501184 0 0 0 0 0 0 0 0
268501216 0 0 0 0 0 0 0 0
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII

Mars Messages
Run I/O
Clear
**** user input : 6
**** user input : 268501024
**** user input : 268501152
**** user input : -12
**** user input : -10
**** user input : -25
**** user input : 0
**** user input : -1
**** user input : 10
-25
-12
-10
-1
0
10