

### Problem 1:

In this problem we were asked to do a bubble sort on an array of integers. First, the program asks the user for the size of the array and the items to be stored in the array in a loop. My design approach was to get a pointer at the end of the array and decrement it whenever the following loop is done once: comparing the first to elements and swapping if needed, then checking the next elements and so on until hitting the pointer of the last element which will be decremented every loop. For repeated items they stay the same and they are printed twice. The parameters were pushed on the stack to be received in the function itself.

In x86, I did the array static just to examine the logic in x86. I assumed that the user will not enter any numbers or anything just an output for it to change the program has to change. The parameters are sent to a label that pops the items from the stack to be used. The flow is generally the same as mips but the syntax is different. I do decrement the comparison loop every time I finish the loop, and all of that happens under 3 labels. Moreover, the printing of the array was clumsy at the end because there is no spacing however I made it such that it is only one digit, so there is a separation every single digit.

The screenshot displays the Mars MIPS simulator interface. The top panel shows assembly code with instructions like `lui $t0, 0x00001001`, `syscall`, `addiu $t0, $t0, 5`, `move $t0, $v0`, `la $t1, array`, `move $t9, $t1`, and `la $a0, ask`. The middle panel shows a data segment with addresses from `0x10010000` to `0x10010100` and values. The bottom panel shows the console output with the message "Enter the integers to be sorted: 12" and the array `0 1 2 5 6 12 23 71 99`. The console also shows "program is finished running".

```
zaieda@zaieda-VirtualBox: ~/Desktop$ nasm -felf64 prol.asm && ld prol.o
zaieda@zaieda-VirtualBox: ~/Desktop$ ./a.out
zaieda@zaieda-VirtualBox: ~/Desktop$ ./a.out
01479zaieda@zaieda-VirtualBox: ~/Desktop$
```

### Problem 2:

In this problem I have implemented the recursion on one stack frame in which I only update to the items in this stack frame and use it for the following call to reduce memory use. I have passed the return address of the first jal along with the right and left pointer, the size of the array and the element to be searching for. I started with the middle element in the array with some pointer fixation when dividing the end and start pointer by two. Then comparing the item with this middle element if it is equal (stopping case) then exit the loop. Check if less than or bigger than then update the stack frame and call the function again with these pointers updated. Until finding the element. If found I get the address and subtract it with the address of the first element and mod it by 4 to get its location.

In x86 I did the same logic of recursion where I only update the same stack frame such that I reduce the memory usage. I used call and retq to be able to manipulate the ra idea in mips. I made the most validations I can do with the current knowledge, assuming that if the user wants to change something, they will update the .asm file. I always update the stack using push and pop, and whilst I'm doing that I call the recursion. Moreover the labels were done normally using jmp and other conditional instructions in x86\_64, assuming that we are working with 64 bit.

```

0x00400010: 0x00000021 addu $24,$0,$8
0x00400020: 0x3c011001 lui $1,0x00010001
0x00400024: 0x34250000 ori $9,$1,0x00000000
0x00400028: 0x24020004 addiu $2,$0,0x00000004
0x0040002c: 0x3c011001 lui $1,0x00010001

```

```

20:      move $t8, $t0      #copy the same pointer into t8 as well
21:      la $t1, array      #pointer to the array
23:      li $v0, 4           #system call to print the ask num to be stored in the array
24:      la $a0, ask num

```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000003	0x00000005	0x00000008	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x746e450a	0x74207265	0x6e206568	0x65626d75	0x66612072	0x656c6520	0x746e656d
0x10010080	0x6e692073	0x65687420	0x72726120	0x203a7961	0x6e450a00	0x20726574	0x20656874	0x65746e69
0x100100a0	0x73726567	0x6f732820	0x64657472	0x00283a29	0x746e450a	0x74207265	0x6e206568	0x65626d75
0x100100c0	0x6f752072	0x72612075	0x65732065	0x68637261	0x20676e69	0x3a726f66	0x00000020	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

☒ Hexadecimal Addresses   
 ☒ Hexadecimal Values   
 ☐ ASCII

If checked, displays all memory addresses in hexadecimal. Otherwise, decimal.

Mars Messages
Run I/O

```

Enter the number of elements in the array: 4
Enter the integers (sorted): 1
3
5
8

Enter the number you are searching for: 5
2
— program is finished running —

```

0x00400028 0x24020004 addiu \$2,\$0,0x00000004 23: li \$v0,4 #system call to print the ask num to be stored in the array  
0x0040002c 0x3c011001 lui \$1,0x00001001 24: la \$a0,ask num

Address
Value (+0)
Value (+4)
Value (+8)
Value (+c)
Value (+10)
Value (+14)
Value (+18)
Value (+1c)

0x10010000	0x00000001	0x00000003	0x00000005	0x00000007	0x00000009	0x0000000b	0x0000000d	0x0000000f
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x746e450a	0x74207265	0x65620568	0x65626d75	0x66617072	0x656c6520	0x746e656d
0x10010080	0x656c92073	0x65687420	0x72726120	0x203a7961	0x6e6450a0	0x20726574	0x20656874	0x65746e69
0x100100a0	0x73726567	0x6f732820	0x64657472	0x00283a29	0x746e450a	0x74207265	0x6e206568	0x65626d75
0x100100c0	0x6f792072	0x72612075	0x65732065	0x68637261	0x20676e69	0x3a726f66	0x00000020	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

◀ ▶

0x10010000 [data]
▼

☒ Hexadecimal Addresses
 ☒ Hexadecimal Values
 ☐ ASCII

Mars Messages
Run /O

```

Enter the number of elements in the array: 4
Enter the integers (sorted): 1
3
5
7
Enter the number you are searching for: 9
-1
— program is finished running —

```

```
zaieda@zaieda-VirtualBox:~$ cd Desktop/  
zaieda@zaieda-VirtualBox:~/Desktop$ nasm -felf64 pro2.asm && ld pro2.o  
zaieda@zaieda-VirtualBox:~/Desktop$ ./a.out  
zaieda@zaieda-VirtualBox:~/Desktop$ ./a.out  
Found in index of: 3zaieda@zaieda-VirtualBox:~/Desktop$
```

```
zaieda@zaieda-VirtualBox:~/Desktop$ clear  
zaieda@zaieda-VirtualBox:~/Desktop$ nasm -felf64 pro2.asm && ld pro2.o  
zaieda@zaieda-VirtualBox:~/Desktop$ ./a.out  
Not foundzaieda@zaieda-VirtualBox:~/Desktop$ █
```