

Sreyas Janamanchi(IMT2022554)

Pradyumna G(IMT2022555)

We have chosen bubble sort as our sorting algorithm.

MIPS code:-

1. We copied all the elements of the input array into the output array.
2. We created a loop in a loop and initialized counter variables.
3. To implement loop we made a branch condition, that branches when value of the loop counter reaches the maximum value. Just before the branch label a jump statement is used to jump back to the branch condition of the loop.
4. loopa: for(int i=1;i<N;i++)  
    loopb: for(int j=1;j<N-i;j++)
5. In loopb we compare output[j] and output[j-1]. If output[j]>output[j-1] we branch ahead of a swap functionality snippet.

Assembler.py:-

1. We created 3 dictionaries: instructions (list of instructions against their opcodes), pseudocodes (list of pseudocodes against their MIPS instruction names), variables (list of registers against their address)
2. We initialized an empty list(data) that stores the machine code which is later printed out in another text file known as:  
    "IMT2022554\_IMT2022555\_output.txt".
3. The MIPS code is read from "IMT2022554\_IMT2022555\_sorting.asm".
4. The code is split line by line and each word of a line is looped through.
5. We try matching each word with the keys present in "instructions" dictionary. And store the machine code of the instruction in the "data" list accordingly.

- ## SCREENSHOTS

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	4194304	0x0c100050	jal 4194624	14: jal print_inp_statement
	4194308	0x0c100043	jal 4194572	15: jal input_int
	4194312	0x000c4821	addu \$9,\$0,\$12	16: move \$t1,\$t4
	4194316	0x0c100055	jal 4194644	20: jal print_int_inp_statement
	4194320	0x0c100043	jal 4194572	21: jal input_int
	4194324	0x000c5021	addu \$10,\$0,\$12	22: move \$t2,\$t4
	4194328	0x0c10005a	jal 4194664	26: jal print_out_inp_statement
	4194332	0x0c100043	jal 4194572	27: jal input_int
	4194336	0x000c5821	addu \$11,\$0,\$12	28: move \$t3,\$t4
	4194340	0x000ac021	addu \$24,\$0,\$10	32: move \$t8,\$t2
	4194344	0x0000b821	addu \$23,\$0,\$0	33: move \$9,\$zero #1 = 0
	4194348	0x12e90006	bq \$23,\$9,6	34: loopi: beq \$9,\$t1,loopiend

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	5	8	2	9	3	544436837	1646292852	1635000421
268501024	544105835	1763734369	1953853550	1157636154	1919251566	1635021600	18524004850	1684069887
268501056	19626028260	1718558835	1886284064	1678955093	1679847017	1835623269	1713400929	1634562671
268501088	540692612	1953383680	1931506277	1953653108	543649385	1919181921	544437093	1864394351
268501120	1970304117	673215348	1679847017	1835623269	1713400929	1634562671	540692612	1953383680
268501152	1948283493	1763730792	1734702190	540701285				0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	2	3	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0

☐ Hexadecimal Addresses
 ☐ Hexadecimal Values
 ☐ ASCII

Mars Messages Run I/O

Enter No. of integers to be taken as input: 5

Enter starting address of inputs(in decimal format): 268500992

Enter starting address of outputs (in decimal format): 268501248

Enter the integer: 5

Enter the integer: 8

Enter the integer: 2

Enter the integer: 9

Enter the integer: 3

2130508190

Clear

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	
\$at	1	268500992
\$v0	2	1
\$v1	3	
\$a0	4	268500992
\$a1	5	
\$a2	6	
\$a3	7	
\$t0	8	
\$t1	9	
\$t2	10	268500992
\$t3	11	268501248
\$t4	12	
\$t5	13	
\$t6	14	
\$t7	15	268501248
\$t8	16	
\$t9	17	
\$s2	18	
\$s3	19	
\$s4	20	
\$s5	21	
\$s6	22	
\$s7	23	
\$t10	24	268500992
\$t19	25	
\$k0	26	
\$k1	27	
\$gp	28	268486224
\$fp	29	214747954
\$fp	30	
\$ra	31	419455
PC		419457
hi		
lo		

## 2) Machine code of MIPS sorting algorithm generated by MIPS

```

≡ machinecode
 1  0010010000000110000000000000000000000000
 2  000000000000001001011010000001000001
 3  00100100000010010000000000000000000100
 4  0001000011000110100000000000000000111
 5  01110000110010010100110000000000010
 6  0000000010101001101111000000100000
 7  10001101111011100000000000000000000000
 8  0000000010111001101111000000100000
 9  10101101111011100000000000000000000000
10  00100000110001100000000000000000000001
11  00001000000010000000000000000000000011
12  0010000010011100100000000000000000001
13  0010000000001010100000000000000000001
14  00010010101110010000000000000011000
15  0010000000001011000000000000000000001
16  0000000110011010110111000000100010
17  00010010110101110000000000000010011
18  0111001011010010100110000000000010
19  0000000010111001101111000000100000
20  10001101111100010000000000000000000000
21  0010000000000000001000000000000000100
22  0000000100110000110011000000100010
23  0000000010111001101111000000100000
24  10001101111101000000000000000000000000
25  0000000101001000100001000000101010
26  00010100000100000000000000000000001000
27  000000000000001000101100000000100001
28  000000000000001010010001000000100001
29  000000000000000110010100000000100001
30  0000000010111001101111000000100000
31  10101101111101000000000000000000000000
32  0010001001110011000000000000000000100
33  0000000010111001101111000000100000
34  10101101111100010000000000000000000000
35  0010001011010110000000000000000000001
36  0000100000001000000000000000000010000
37  001000101011010100000000000000000001
38  00001000000010000000000000000000001101

```

3)Machine code output from assembler.py

IMT2022554\_IMT2022555\_output.txt

```
1 0010010000001100000000000000000000000000
2 0000000000000100101101000001000001
3 0010010000010010000000000000000100
4 0001000110001101000000000000000111
5 011100011001001010011000000000010
6 00000001010100110111100000100000
7 1000110111101110000000000000000000
8 00000001011100110111100000100000
9 1010110111101110000000000000000000
10 0010000110001100000000000000000001
11 0000100000010000000000000000000011
12 001000010011100100000000000000001
13 001000000001010100000000000000001
14 000100101011100100000000000011000
15 001000000001011000000000000000001
16 00000011001101011011100000100010
17 000100101101011100000000000010011
18 01110010110100101001100000000010
19 00000001011100110111100000100000
20 1000110111110001000000000000000000
21 0010000000000000100000000000000100
22 00000010011000011001100000100010
23 00000001011100110111100000100000
24 1000110111110100000000000000000000
25 00000010100100010000100000101010
26 0001010000100000000000000000001000
27 0000000000001000101100000000100001
28 000000000000101001000100000100001
29 0000000000000110010100000000100001
30 00000001011100110111100000100000
31 1010110111110100000000000000000000
32 0010001001110011000000000000000100
33 00000001011100110111100000100000
34 1010110111110001000000000000000000
35 001000101101011000000000000000001
36 0000100000010000000000000000010000
37 001000101011010100000000000000001
38 000010000001000000000000000001101
```