Sreyas Janamanchi(IMT2022554)

Pradyumna G(IMT2022555)

We have chosen bubble sort as our sorting algorithm.

MIPS code's sorting algorithm explanation (IMT2022554_IMT2022555_sorter.asm):-

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.

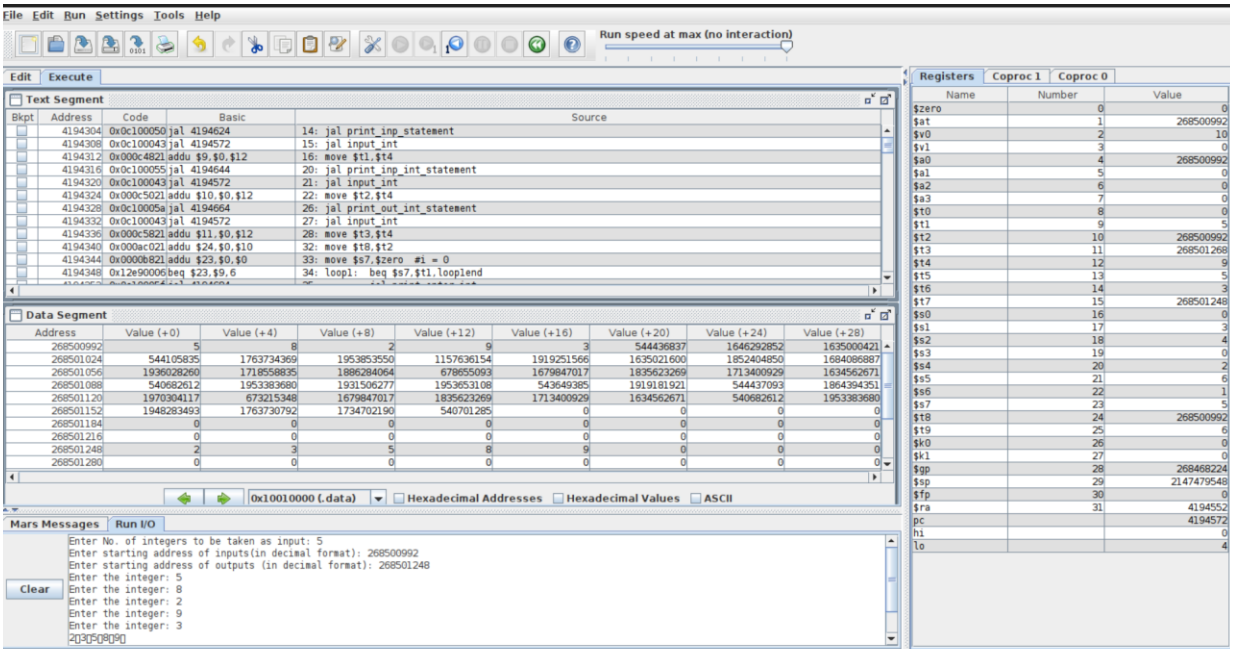
IMT2022554_IMT2022555_assembler.py:-

1. We created 3 dictionaries: instructions (list of instructions against their opcodes), pseudcodes (list of pseudocodes against their MIPS operations), 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_input.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.

6. To find the immediate for beq instruction and j instruction we use beqHelper and jHelper functions respectively.
7. To convert decimal number to the desired binary length we created “decimal_to_binary” function.
8. Each element in “data” list is printed in: “IMT2022554_IMT2022555_output.txt”.

SCREENSHOTS

1) Execution of MIPS code



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_inp_int_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_int_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 $s7,$zero #i = 0
□	4194348	0x12e90006	beq $23,$9,6	34: loop1: beq $s7,$t1,looplend

Data Segment

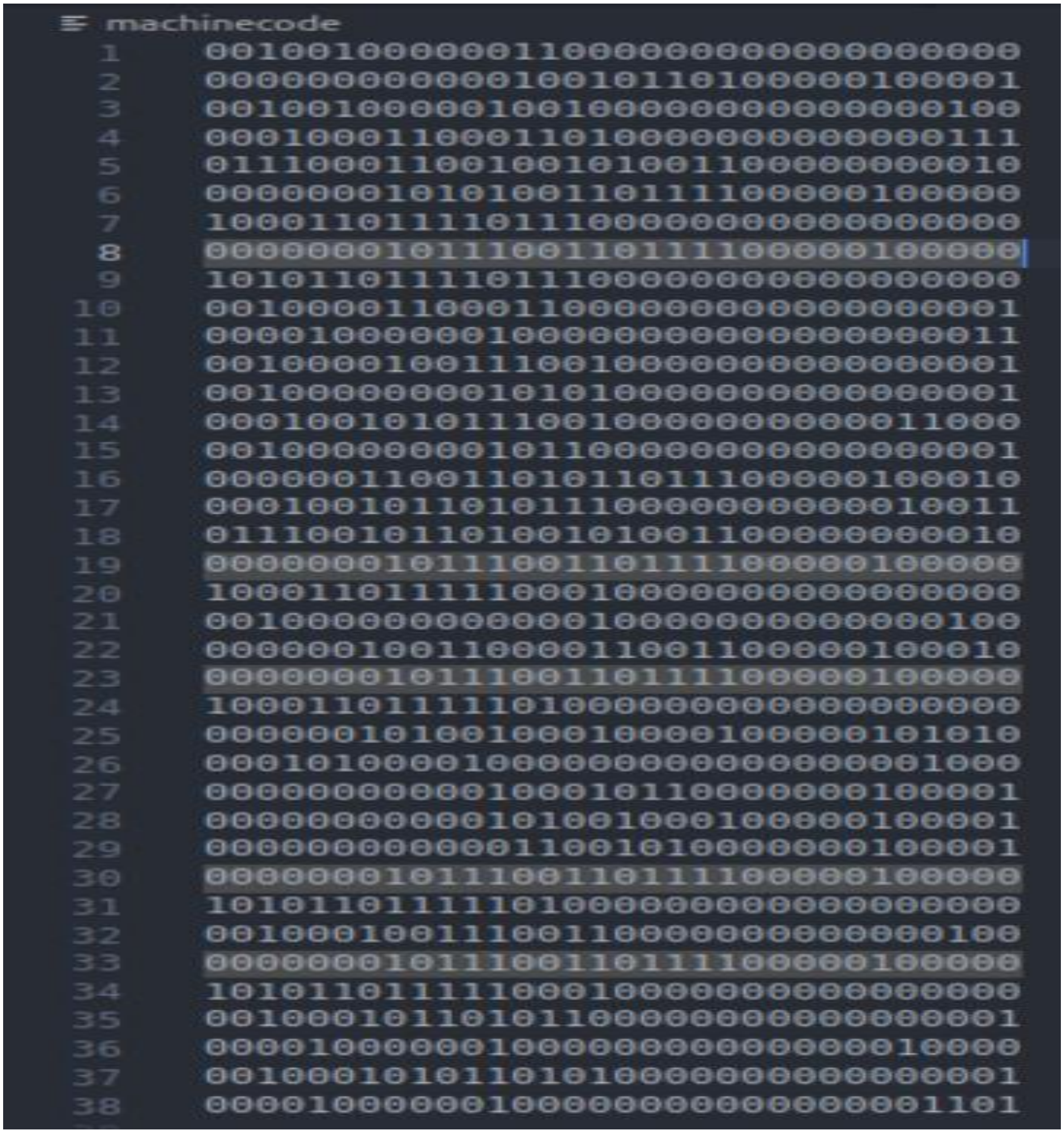
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	5	8	2	9	3	544436837	1646292852	1635000421
268501024	544105835	1763734369	1953853550	1157636154	1919251566	1635021600	1852404850	1684086887
268501056	1936028260	1718558835	1886284064	678655093	1679847017	1835623269	1713400929	1634562671
268501088	540682612	1953383680	1931506277	1953653108	543649385	1919181921	544437093	1864394351
268501120	1970304117	673215348	1679847017	1835623269	1713400929	1634562671	540682612	1953383680
268501152	1948283493	1763730792	1734702190	540701285	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	2	3	5	8	9	0	0	0
268501280	0	0	0	0	0	0	0	0

0x10010000 (.data) □ Hexadecimal Addresses □ Hexadecimal Values □ ASCII

```
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
2▯3▯5▯8▯9▯
```

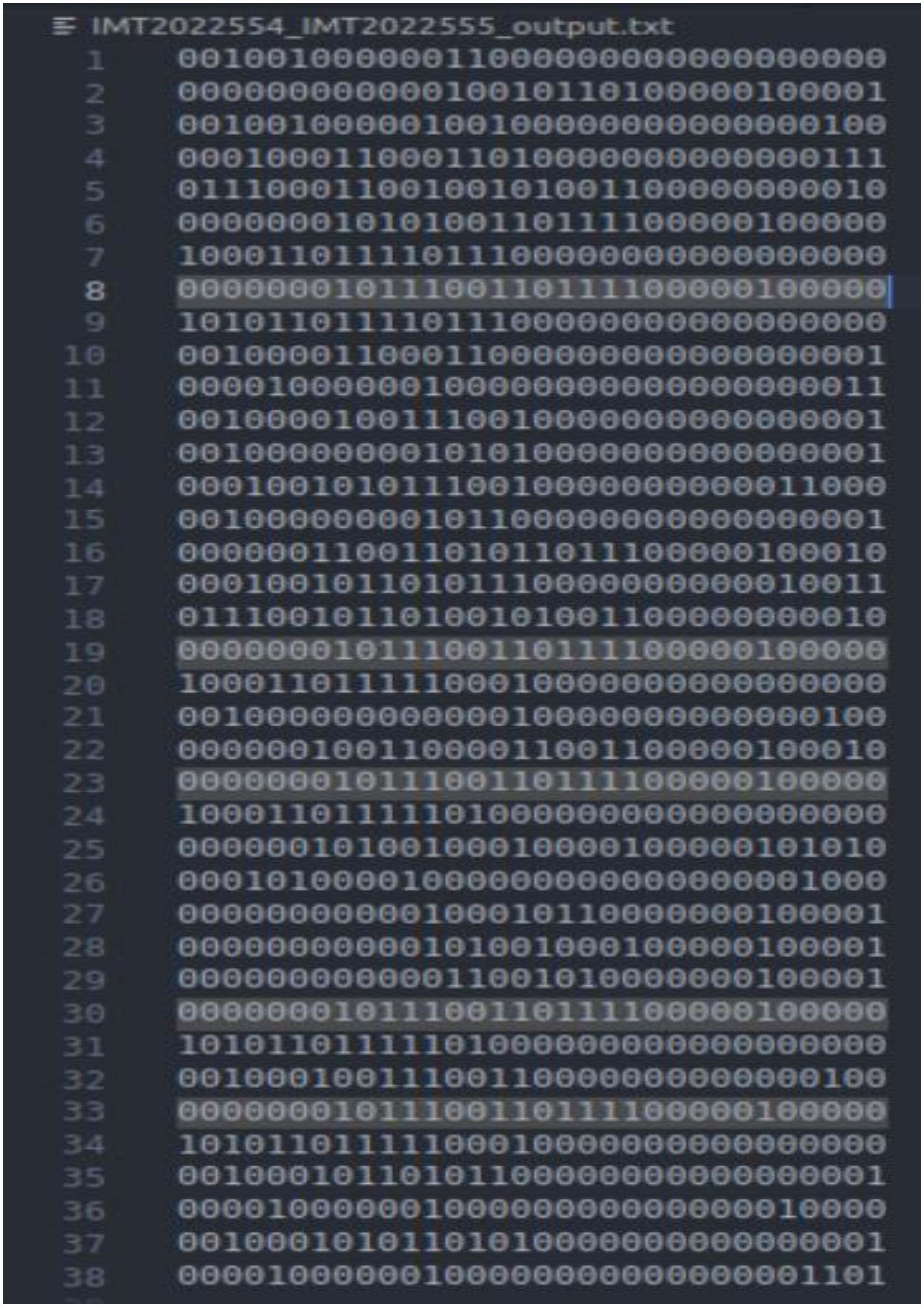
Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	0
$a0	4	268500992
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	5
$t2	10	268500992
$t3	11	268501268
$t4	12	9
$t5	13	5
$t6	14	3
$t7	15	268501248
$s0	16	0
$s1	17	3
$s2	18	4
$s3	19	0
$s4	20	2
$s5	21	6
$s6	22	1
$s7	23	5
$t8	24	268500992
$t9	25	6
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	4194552
pc		4194572
hi		0
lo		4

2)Machine code of MIPS sorting algorithm generated by MIPS



```
00100100000001100000000000000000
00000000000010010110100000100001
00100100000010010000000000000100
00010001100011010000000000000111
01110001100100101001100000000010
00000001010100110111100000100000
10001101111101110000000000000000
00000001011100110111100000100000
10101101111101110000000000000000
00100000110001100000000000000001
00001000000100000000000000000011
00100001001110010000000000000001
00100000000010100000000000000001
00010010101110010000000000011000
00100000000010110000000000000001
00000011001101011011100000100010
00010010110101110000000000010011
01110010110100101001100000000010
00000001011100110111100000100000
10001101111110000000000000000000
00100000000000010000000000000100
00000010011000011001100000100010
00000001011100110111100000100000
10001101111110100000000000000000
00000010100100010000100000101010
00010100001000000000000000001000
00000000000100010110000000100001
00000000000101001000100000100001
00000000000011001010000000100001
00000001011100110111100000100000
10101101111110100000000000000000
00100010011100110000000000000100
00000001011100110111100000100000
10101101111110000000000000000000
00100010110101100000000000000001
00001000000100000000000000010000
00100010101101010000000000000001
00001000000100000000000000001101
```

3)Machine code output from IMT2022554_IMT2022555_assembler.py



```
00100100000011000000000000000000
00000000000010010110100000100001
00100100000100100000000000000100
00010001100011010000000000000111
01110001100100101001100000000010
00000001010100110111100000100000
10001101111101110000000000000000
00000001011100110111100000100000
10101101111101110000000000000000
00100001100011000000000000000001
00001000000100000000000000000011
00100001001110010000000000000001
00100000000101010000000000000001
00010010101110010000000000011000
00100000000101100000000000000001
00000011001101011011100000100010
00010010110101110000000000010011
01110010110100101001100000000010
00000001011100110111100000100000
10001101111100010000000000000000
00100000000000010000000000000100
00000010011000011001100000100010
00000001011100110111100000100000
10001101111101000000000000000000
00000010100100010000100000101010
00010100000100000000000000001000
00000000000100010110000000100001
00000000000101001000100000100001
00000000000011001010000000100001
00000001011100110111100000100000
10101101111101000000000000000000
00100010011100110000000000000100
00000001011100110111100000100000
10101101111100010000000000000000
00100010110101100000000000000001
00001000000100000000000000010000
00100010101101010000000000000001
00001000000100000000000000001101
```