



SOICT

FINAL PROJECT REPORT

PROBLEM 1: MOVING A BALL IN THE BITMAP DISPLAY PROBLEM 9: TESTING SORTING ALGORITHMS

MEMBER LIST

Trương Linh Duyên	20225968	duyen.tl225968@sis.hust.edu.vn
Đinh Nguyễn Sơn	20225997	son.dn225997@sis.hust.edu.vn

Teacher: Lê Bá Vui

CONTENTS

1	Introduction	4
2	Problem 1: MOVING A BALL IN THE BITMAP DISPLAY	4
2.1	Problem Statement	4
2.2	Algorithm Idea	4
2.3	Implementation Pipeline	6
2.4	Source Code	11
2.5	Display Results	15
3	Problem 9: TESTING SORTING ALGORITHMS	18
3.1	Problem Statement	18
3.2	Algorithm Idea	18
3.3	Implementation Pipeline	20
3.4	Source Code	27
3.5	Display Result	43

LIST	OF	TABLES
1	Sorting Algorithm Performance Comparison	47

LIST	OF	FIGURES
1	Start of the bitmap display	16
2	The ball moved up when press W or w	16
3	The ball moved right when press D or d	16
4	The ball moved down when press S or s	17
5	The ball moved left when press A or a	17
6	Bubble Sort Result	43
7	Instruction Count Analysis for Bubble Sort	44
8	Insertion Sort Result	44
9	Instruction Count Analysis for Insertion Sort	45
10	Selection Sort Result	45

11	Instruction Count Analysis for Selection Sort	46
12	Quick Sort Result	46
13	Instruction Count Analysis for Quick Sort	47
14	Output Demonstration	48

1 INTRODUCTION

In this report, we explore solutions to two problems using the RISC-V assembly language, a modern open-source instruction set architecture (ISA) that emphasizes simplicity and modularity. RISC-V's clean and flexible design makes it a suitable choice for low-level programming tasks that require efficiency and control. Our team presents solutions to Problem 1: Moving a Ball in the Bitmap Display and Problem 9: Testing Sorting Algorithms, both implemented in assembly language on the RISC-V architecture.

For Problem 1, we design an interactive program where a ball moves within a bitmap display, responding to keyboard inputs to adjust its direction and speed. The ball's movement is constrained within the display boundaries, reversing direction when it hits the screen edges. This solution demonstrates the application of basic input-output handling, boundary checking, and real-time user interaction, making use of fundamental assembly constructs.

For Problem 9, we implement various sorting algorithms to test their performance and functionality. The algorithms are implemented from scratch, providing insights into their efficiency and behavior at the assembly level. Through this problem, we highlight the importance of understanding sorting mechanics and optimizing code for performance in low-level programming.

Both problems showcase the power and flexibility of assembly programming, especially with RISC-V, allowing us to leverage the hardware directly while optimizing performance. Our solutions illustrate key concepts of system-level programming, hardware control, and algorithmic implementation.

2 PROBLEM 1: MOVING A BALL IN THE BITMAP DISPLAY

2.1 Problem Statement

Create a program that displays a movable round ball on the bitmap screen. If the ball touches the edge of the screen, it will move in the opposite direction.

Requirement:

- Set display width and height to 512 pixels, unit width and height to 1 pixel.
- The direction of movement depends on the key pressed from the keyboard. (W moves up, S moves down, A moves left, D moves right, Z speeds up, X slows down).
- The default position is the center of the screen.

2.2 Algorithm Idea

In this section, we mainly focus on our general idea to solve the problem, read the following section for more details.

Initialization

Initially, the program sets up the following parameters:

- The ball's starting position is set to the center of the screen, with coordinates ($x = 256, y = 256$).
- The ball has a radius of 20 pixels, which is used to calculate its boundary when moving.
- The default movement distance per step is set to 10 pixels.
- The ball's color is set to yellow.
- The program defines a speed variable that controls the delay between updates, thus affecting the movement speed of the ball.

Keyboard Input Handling

The program continuously monitors the keyboard for key presses. Depending on the key pressed, the ball's movement and speed are adjusted:

- **Movement Keys:**
 - W: Move the ball up.
 - S: Move the ball down.
 - A: Move the ball left.
 - D: Move the ball right.
- **Speed Adjustment Keys:**
 - Z: Increase the speed (decrease delay).
 - X: Decrease the speed (increase delay).
- Enter: Exit the program.

Boundary Check

Before each move, the program checks whether the ball will go beyond the screen boundaries. The ball's position is compared with the screen dimensions (512x512 pixels), considering its radius. The following checks are performed:

- If the ball exceeds the right edge, the horizontal direction (dx) is reversed.
- If the ball exceeds the left edge, the horizontal direction (dx) is reversed.
- If the ball exceeds the top edge, the vertical direction (dy) is reversed.
- If the ball exceeds the bottom edge, the vertical direction (dy) is reversed.

These checks ensure that the ball bounces back when hitting the edges.

Ball Movement and Redrawing

After the boundary check, the ball's position is updated:

- The ball's previous position is erased by setting the pixel values in the bitmap to the background color.

- The new position of the ball is computed by adding the respective movement distances (dx , dy) to the current coordinates (x , y).
- The ball is redrawn at the new position using the Bresenham Circle Algorithm.

Bresenham Circle Algorithm

To render the ball as a circle on the screen, the program uses the Bresenham Circle Algorithm, which efficiently computes the points of a circle and plots them on the bitmap. The algorithm calculates the points symmetrically around the center of the circle, ensuring that all eight octants are plotted correctly. The key steps involve:

- Starting with initial values of $(x = 0, y = r)$ and an initial decision parameter $d = 3 - 2r$, where r is the radius.
- Iterating through the circle's points and plotting the corresponding pixel locations in all eight symmetrical positions.
- Adjusting the decision parameter d after each point is plotted, depending on the relative positions of the calculated points.

Speed Control

The ball's speed is adjusted by changing the delay between updates:

- Pressing Z decreases the delay, causing the ball to move faster.
- Pressing X increases the delay, slowing the ball down.

The speed change is managed by adjusting the delay variable, which controls how quickly the game loop proceeds.

Game Loop

The program operates in a continuous game loop:

- The program waits for keyboard input and processes it.
- Based on the input, the direction of movement or speed is updated.
- After updating the position, the ball is erased and redrawn at the new location.
- A delay is introduced, and the loop repeats.

Program Exit

The program runs in a loop until the Enter key is pressed, at which point it terminates gracefully by invoking the exit system call.

2.3 Implementation Pipeline

The implementation of the game is organized into multiple stages, each focusing on a specific task, from initialization to handling user input, checking boundaries, moving the ball, drawing it on the screen, and adjusting speed. This section outlines the key steps involved in the implementation, breaking down each section of the code for better understanding.

Initialization of Game Variables

The first step in the pipeline is the initialization of game variables. This step sets up the initial state of the ball, the screen parameters, movement variables, and the color of the ball.

```

1 # Initialization
2 li s0, 256      # x = 256 (initial x-coordinate)
3 li s1, 256      # y = 256 (initial y-coordinate)
4 li s2, 20       # R = 20 (radius)
5 li s3, 512      # Screen width (512 pixels)
6 li s4, 512      # Screen height (512 pixels)
7 li s5, YELLOW   # Ball color (yellow)
8 li s6, MOVE_DISTANCE # Move distance per step (10 pixels)
9 li s7, 0        # dx = 0 (horizontal movement change)
10 li s8, 0        # dy = 0 (vertical movement change)
11 li s9, 70       # Initial speed (delay)

```

****Explanation:****

- The variables 's0' and 's1' represent the initial coordinates of the ball (set to the center of the screen at '(256, 256)').
- 's2' defines the ball's radius, which is '20' pixels.
- 's3' and 's4' define the screen's width and height, which are both '512' pixels.
- 's5' stores the color of the ball, which is set to yellow.
- 's6' defines the movement distance per step (10 pixels).
- 's7' and 's8' are initialized to zero and will store the change in position ('dx' and 'dy') for horizontal and vertical movement respectively.
- 's9' is the delay used to control the speed of the game, initially set to '70'.

Keyboard Input Handling

The program continuously reads the keyboard input to detect key presses that control the ball's movement. Each key has a specific function.

```

1 # Read Keyboard Input
2 li t0, KEYBOARD_CONTROL # Load the control register address
3 lw t1, 0(t0)            # Read the status of the keyboard
4 beqz t1, check_bounds   # If no key pressed, check bounds
5
6 # Read the key data
7 li t0, KEYBOARD_DATA
8 lw t2, 0(t0)            # Read the key code
9
10 # Handle movement keys
11 li t3, KEY_A
12 beq t2, t3, move_left   # Move left (A)
13 li t3, KEY_D
14 beq t2, t3, move_right  # Move right (D)
15 li t3, KEY_W
16 beq t2, t3, move_up     # Move up (W)
17 li t3, KEY_S
18 beq t2, t3, move_down   # Move down (S)
19
20 # Handle speed keys
21 li t3, KEY_Z
22 beq t2, t3, speed_up    # Speed up (Z)
23 li t3, KEY_X

```

```

24 beq t2, t3, speed_down      # Speed down (X)
25
26 # Exit program
27 li t3, KEY_ENTER
28 beq t2, t3, exit_program    # Exit (Enter)
29
30 j check_bounds

```

****Explanation:****

- The first step is to check if any key is pressed by reading the keyboard control register. If no key is pressed, the program jumps to the boundary checking phase ('check_bounds').
- If a key is pressed, the key code is read from the 'KEYBOARD_DATA' register.
- The program then checks if the key corresponds to one of the predefined movement or speed control keys ('W', 'A', 'S', 'D', 'Z', 'X', 'Enter').
 - 'A' moves the ball left ('move_left'), 'D' moves it right ('move_right'), 'W' moves it up ('move_up'), and 'S' moves it down ('move_down').
 - 'Z' and 'X' are used to increase or decrease the movement speed ('speed_up' and 'speed_down').
 - 'Enter' triggers the exit of the program ('exit_program').

Boundary Checking

The ball's movement is restricted within the screen boundaries. If the ball touches the edge of the screen, it will reverse direction along the respective axis.

```

1  check_bounds:
2      # Check right boundary (x + R + dx >= screen_width)
3      add t0, s0, s2          # x + R
4      add t0, t0, s7          # x + R + dx
5      bge t0, s3, reverse_x   # If x + R + dx >= screen_width, reverse direction
6
7      # Check left boundary (x - R + dx < 0)
8      sub t0, s0, s2          # x - R
9      add t0, t0, s7          # x - R + dx
10     bltz t0, reverse_x      # If x - R + dx < 0, reverse direction
11
12     # Check top boundary (y - R + dy < 0)
13     sub t0, s1, s2          # y - R
14     add t0, t0, s8          # y - R + dy
15     bltz t0, reverse_y      # If y - R + dy < 0, reverse direction
16
17     # Check bottom boundary (y + R + dy >= screen_height)
18     add t0, s1, s2          # y + R
19     add t0, t0, s8          # y + R + dy
20     bge t0, s4, reverse_y    # If y + R + dy >= screen_height, reverse
                                # direction
21
22     j update_position        # If within bounds, update position

```

****Explanation:****

- The boundaries of the screen are checked along both the x and y axes:

- Right boundary: If the ball's rightmost edge ($x + R + dx$) exceeds or equals the screen width, the horizontal direction is reversed ('reverse_x').
- Left boundary: If the ball's leftmost edge ($x - R + dx$) is less than zero, the horizontal direction is reversed ('reverse_x').
- Top boundary: If the ball's topmost edge ($y - R + dy$) is less than zero, the vertical direction is reversed ('reverse_y').
- Bottom boundary: If the ball's bottommost edge ($y + R + dy$) exceeds or equals the screen height, the vertical direction is reversed ('reverse_y').

Updating the Position and Redrawing the Ball

Once the boundaries have been checked and movement is determined, the ball's position is updated, and it is redrawn at its new location.

```

1 | update_position:
2 |     # Erase previous ball position (set background color)
3 |     li s5, BACKGROUND      # Set color to background (black)
4 |     jal draw_circle        # Draw ball at previous position
5 |
6 |     # Update position (x += dx, y += dy)
7 |     add s0, s0, s7          # x += dx
8 |     add s1, s1, s8          # y += dy
9 |
10 |    # Draw the ball at the new position
11 |    li s5, YELLOW           # Set color to yellow
12 |    jal draw_circle        # Draw ball at new position
13 |
14 |    # Delay
15 |    mv a0, s9
16 |    li a7, 32
17 |    ecall                  # System call for delay
18 |
19 |    j game_loop             # Return to game loop

```

****Explanation:****

- The previous position of the ball is erased by drawing it in the background color ('BACKGROUND').
- The new position is computed by adding 'dx' and 'dy' to the current position ('s0' and 's1').
- The ball is then drawn at its updated position using the 'draw_circle' subroutine, and the ball is colored yellow.
- A delay is introduced to control the speed of the game, based on the value in 's9'.
- The game loop then repeats.

Bresenham Circle Drawing Algorithm

To draw the ball (represented as a circle), the 'draw_circle' subroutine uses Bresenham's circle algorithm, which efficiently computes the points of a circle.

```

1 | draw_circle:
2 |     # Initialize circle drawing variables
3 |     li t0, 0               # x = 0
4 |     mv t1, s2              # y = r (radius)

```

```

5      li t2, 3                # d = 3 - 2r (initial decision parameter)
6      sub t3, t2, s2          # d = 3 - 2r - r (adjusted decision parameter)
7      sub t3, t3, s2
8
9  draw_loop:
10     bgt t0, t1, draw_end    # Loop while x <= y
11
12     # Plot the 8 symmetric points of the circle
13     add a0, s0, t0          # Plot point (x0 + x, y0 + y)
14     add a1, s1, t1
15     jal plot_pixel_safe     # Plot point on screen
16
17     # Repeat for other symmetrical points...
18
19     # Update decision parameter d and other variables
20     bgez t3, adjust_d_positive
21
22     # Adjust decision parameter
23 adjust_d_positive:
24     add t3, t3, t4
25     addi t1, t1, -1
26     continue_draw:
27     addi t0, t0, 1          # x++
28     j draw_loop
29
30 draw_end:
31     ret

```

****Explanation:****

- The 'draw_circle' subroutine uses Bresenham's algorithm to plot a circle by calculating points along the perimeter of the circle.
- The algorithm uses a decision parameter ('d') that determines whether the next point should be plotted horizontally or diagonally.
- For each step, the algorithm plots the 8 symmetric points of the circle using the 'plot_pixel_safe' subroutine.

Speed Control

Speed control is handled by adjusting the delay between game loop iterations. The delay is decreased to speed up the game or increased to slow it down.

```

1  speed_up:
2      addi s9, s9, -10    # Decrease delay, increase speed
3      j check_bounds
4
5  speed_down:
6      addi s9, s9, 10     # Increase delay, decrease speed
7      j check_bounds

```

****Explanation:****

- Pressing 'Z' reduces the delay by 10 units, making the ball move faster.
- Pressing 'X' increases the delay by 10 units, making the ball move slower.

Exiting the Program

The program can be exited by pressing the 'Enter' key. The exit procedure is handled using the 'ecall' system call.

```
1 | exit_program:
2 |     li a7, 10      # Exit system call
3 |     ecall          # Exit the program
```

****Explanation:****

- When the 'Enter' key is pressed, the program triggers an exit system call ('ecall' with argument '10') to terminate the program.

Game Loop

The game loop is the core of the program, repeatedly checking for input, updating the position of the ball, drawing it on the screen, and controlling the game speed.

```
1 | game_loop:
2 |     # Process keyboard input
3 |     # Move ball or adjust speed based on key press
4 |     # Update position and redraw ball
5 |     # Add delay and repeat
6 |     j game_loop
```

****Explanation:****

- The 'game_loop' is a continuous loop that checks the keyboard input, updates the ball's position, redraws the ball at the new position, and then introduces a delay before repeating the process.

2.4 Source Code

```
1 | # Cac hang so cho man hinh va mau sac
2 | .eqv DISPLAY_ADDRESS 0x10010000 # Dia chi bat dau cua Bitmap Display
3 | .eqv YELLOW 0x00FFFF66
4 | .eqv BACKGROUND 0x00000000
5 |
6 | # Cac hang so cho phim
7 | .eqv KEYBOARD_CONTROL 0xFFFF0000
8 | .eqv KEYBOARD_DATA 0xFFFF0004
9 | .eqv KEY_A 0x61 # sang trai
10 | .eqv KEY_D 0x64 # sang phai
11 | .eqv KEY_S 0x73 # xuong duoi
12 | .eqv KEY_W 0x77 # len tren
13 | .eqv KEY_Z 0x7a # giam toc
14 | .eqv KEY_X 0x78 # tang toc
15 | .eqv KEY_ENTER 0x0a # thoat
16 |
17 | # Hang so khac
18 | .eqv MOVE_DISTANCE 10 # Khoang cach di chuyen
19 | .eqv CIRCLE_RADIUS 20 # Ban kinh hinh tron
20 |
21 | .data
```

```

22     coords: .space 512 # Mang luu toa do cac diem
23
24 .text
25 .globl main
26
27 main:
28     # Khoi tao cac gia tri
29     li s0, 256      # x = 256 (toa do x ban dau)
30     li s1, 256      # y = 256 (toa do y ban dau)
31     li s2, 20       # R = 20 (ban kinh)
32     li s3, 512      # Chieu rong man hinh
33     li s4, 512      # Chieu cao man hinh
34     li s5, YELLOW   # Mau hinh tron
35     li s6, MOVE_DISTANCE # Khoang cach di chuyen
36     li s7, 0        # dx (thay doi theo x)
37     li s8, 0        # dy (thay doi theo y)
38     li s9, 70       # Toc do delay
39
40     # Ve hinh tron ban dau
41     jal draw_circle
42
43 game_loop:
44     # Doc phim
45     li t0, KEYBOARD_CONTROL
46     lw t1, 0(t0)
47     beqz t1, check_bounds
48
49     # Doc ma phim
50     li t0, KEYBOARD_DATA
51     lw t2, 0(t0)
52
53     # Xu ly phim
54     li t3, KEY_A
55     beq t2, t3, move_left
56     li t3, KEY_D
57     beq t2, t3, move_right
58     li t3, KEY_W
59     beq t2, t3, move_up
60     li t3, KEY_S
61     beq t2, t3, move_down
62     li t3, KEY_Z
63     beq t2, t3, speed_up
64     li t3, KEY_X
65     beq t2, t3, speed_down
66     li t3, KEY_ENTER
67     beq t2, t3, exit_program
68     j check_bounds
69
70 move_left:
71     neg s7, s6      # dx = -MOVE_DISTANCE
72     li s8, 0        # dy = 0
73     j check_bounds
74
75 move_right:
76     mv s7, s6       # dx = MOVE_DISTANCE
77     li s8, 0        # dy = 0
78     j check_bounds
79
80 move_up:
81     li s7, 0        # dx = 0

```

```

82     neg s8, s6      # dy = -MOVE_DISTANCE
83     j check_bounds
84
85 move_down:
86     li s7, 0        # dx = 0
87     mv s8, s6       # dy = MOVE_DISTANCE
88     j check_bounds
89
90 speed_up:
91     addi s9, s9, -10 # Giam delay = tang toc
92     j check_bounds
93
94 speed_down:
95     addi s9, s9, 10  # Tang delay = giam toc
96     j check_bounds
97
98 check_bounds:
99     # Kiem tra bien phai
100    add t0, s0, s2    # x + R
101    add t0, t0, s7    # + dx
102    bge t0, s3, reverse_x
103
104    # Kiem tra bien trai
105    sub t0, s0, s2    # x - R
106    add t0, t0, s7    # + dx
107    bltz t0, reverse_x
108
109    # Kiem tra bien tren
110    sub t0, s1, s2    # y - R
111    add t0, t0, s8    # + dy
112    bltz t0, reverse_y
113
114    # Kiem tra bien duoi
115    add t0, s1, s2    # y + R
116    add t0, t0, s8    # + dy
117    bge t0, s4, reverse_y
118
119    j update_position
120
121 reverse_x:
122     neg s7, s7      # Doi chieu dx
123     j update_position
124
125 reverse_y:
126     neg s8, s8      # Doi chieu dy
127     j update_position
128
129 update_position:
130     # Xoa hinh tron cu
131     li s5, BACKGROUND
132     jal draw_circle
133
134     # Cap nhat vi tri
135     add s0, s0, s7   # x += dx
136     add s1, s1, s8   # y += dy
137
138     # Ve hinh tron moi
139     li s5, YELLOW
140     jal draw_circle
141

```

```

142     # Delay
143     mv a0, s9
144     li a7, 32
145     ecall
146
147     j game_loop
148
149 draw_circle:
150     # Luu ra
151     addi sp, sp, -4
152     sw ra, 0(sp)
153
154     # Bresenham circle algorithm
155     li t0, 0          # x = 0
156     mv t1, s2          # y = r
157     li t2, 3          # d = 3
158     sub t3, t2, s2     # d = 3 - 2r
159     sub t3, t3, s2
160
161 draw_loop:
162     bgt t0, t1, draw_end # while x <= y
163
164     # Plot 8 points
165     # Tren phai
166     add a0, s0, t0      # x0 + x
167     add a1, s1, t1      # y0 + y
168     jal plot_pixel_safe
169
170     add a0, s0, t1      # x0 + y
171     add a1, s1, t0      # y0 + x
172     jal plot_pixel_safe
173
174     # Tren trai
175     sub a0, s0, t0      # x0 - x
176     add a1, s1, t1      # y0 + y
177     jal plot_pixel_safe
178
179     sub a0, s0, t1      # x0 - y
180     add a1, s1, t0      # y0 + x
181     jal plot_pixel_safe
182
183     # Duoi phai
184     add a0, s0, t0      # x0 + x
185     sub a1, s1, t1      # y0 - y
186     jal plot_pixel_safe
187
188     add a0, s0, t1      # x0 + y
189     sub a1, s1, t0      # y0 - x
190     jal plot_pixel_safe
191
192     # Duoi trai
193     sub a0, s0, t0      # x0 - x
194     sub a1, s1, t1      # y0 - y
195     jal plot_pixel_safe
196
197     sub a0, s0, t1      # x0 - y
198     sub a1, s1, t0      # y0 - x
199     jal plot_pixel_safe
200
201     # Cap nhat cac bien

```

```

202     bgez t3, adjust_d_positive
203
204     # d < 0
205     slli t4, t0, 2      # 4x
206     addi t4, t4, 6      # 4x + 6
207     add t3, t3, t4      # d += 4x + 6
208     j continue_draw
209
210 adjust_d_positive:
211     # d >= 0
212     slli t4, t0, 2      # 4x
213     sub t4, t4, t1      # 4x - y
214     sub t4, t4, t1      # 4x - 2y
215     sub t4, t4, t1      # 4x - 3y
216     sub t4, t4, t1      # 4x - 4y
217     addi t4, t4, 10     # 4x - 4y + 10
218     add t3, t3, t4      # d += 4x - 4y + 10
219     addi t1, t1, -1     # y--
220
221 continue_draw:
222     addi t0, t0, 1      # x++
223     j draw_loop
224
225 draw_end:
226     lw ra, 0(sp)
227     addi sp, sp, 4
228     ret
229
230 plot_pixel_safe:
231     # Kiem tra bien
232     bltz a0, plot_end   # x < 0
233     bge a0, s3, plot_end # x >= width
234     bltz a1, plot_end   # y < 0
235     bge a1, s4, plot_end # y >= height
236
237     # Tinh offset
238     mul t6, a1, s3      # y * width
239     add t6, t6, a0      # + x
240     slli t6, t6, 2      # * 4 bytes/pixel
241     li t4, DISPLAY_ADDRESS
242     add t6, t6, t4
243
244     # Ve pixel
245     sw s5, 0(t6)
246
247 plot_end:
248     ret
249
250 exit_program:
251     li a7, 10
252     ecalls

```

2.5 Display Results

The following images represent our results:

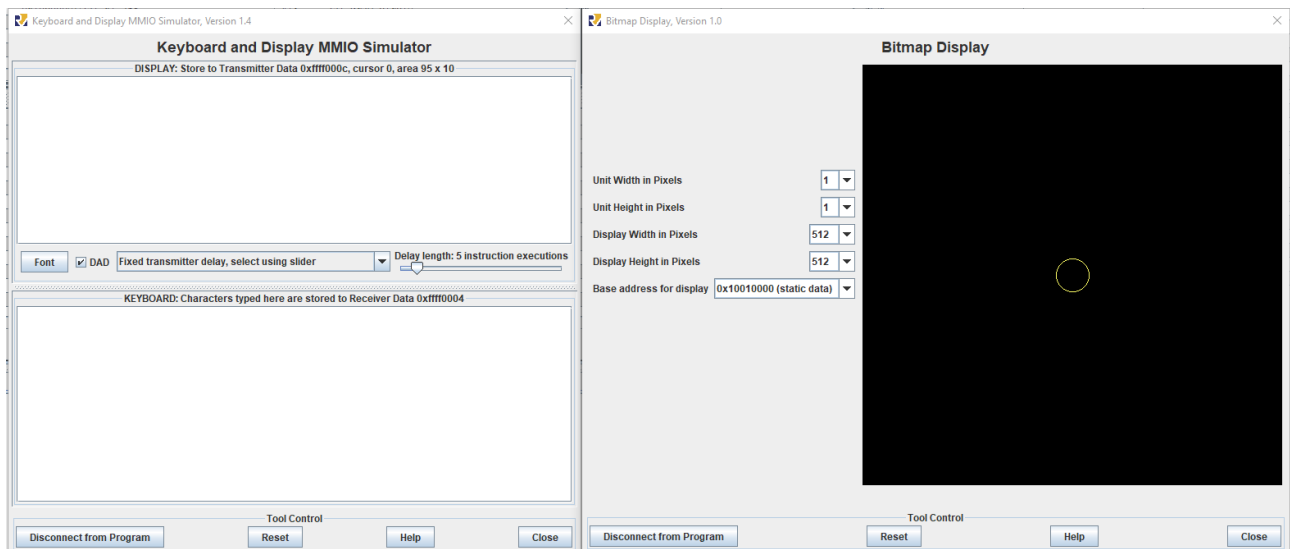


Figure 1: Start of the bitmap display

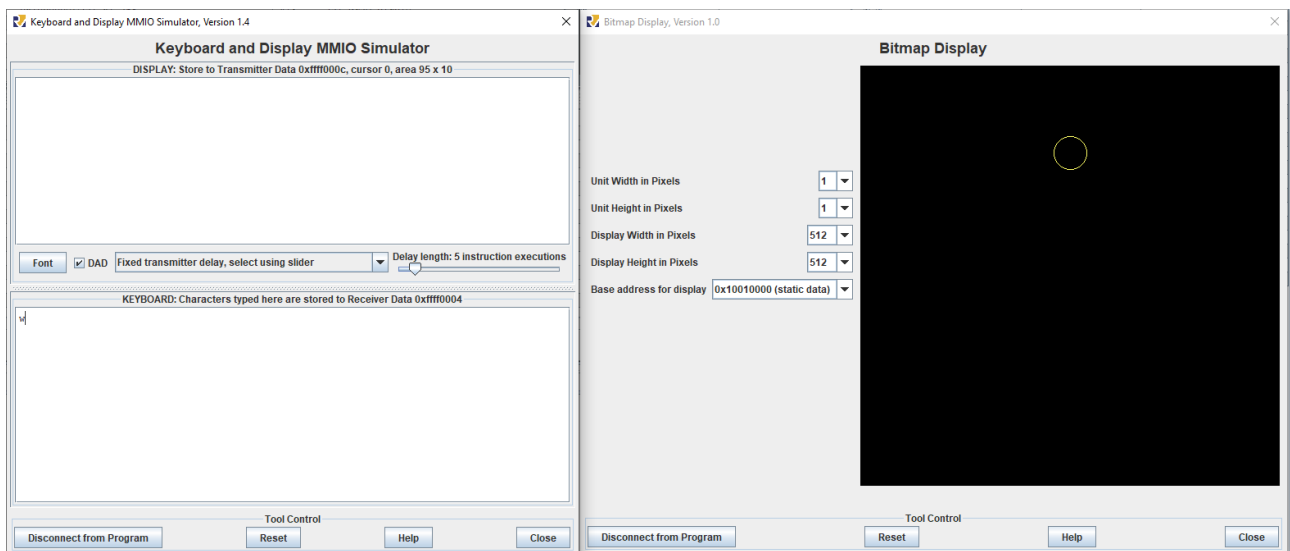


Figure 2: The ball moved up when press W or w

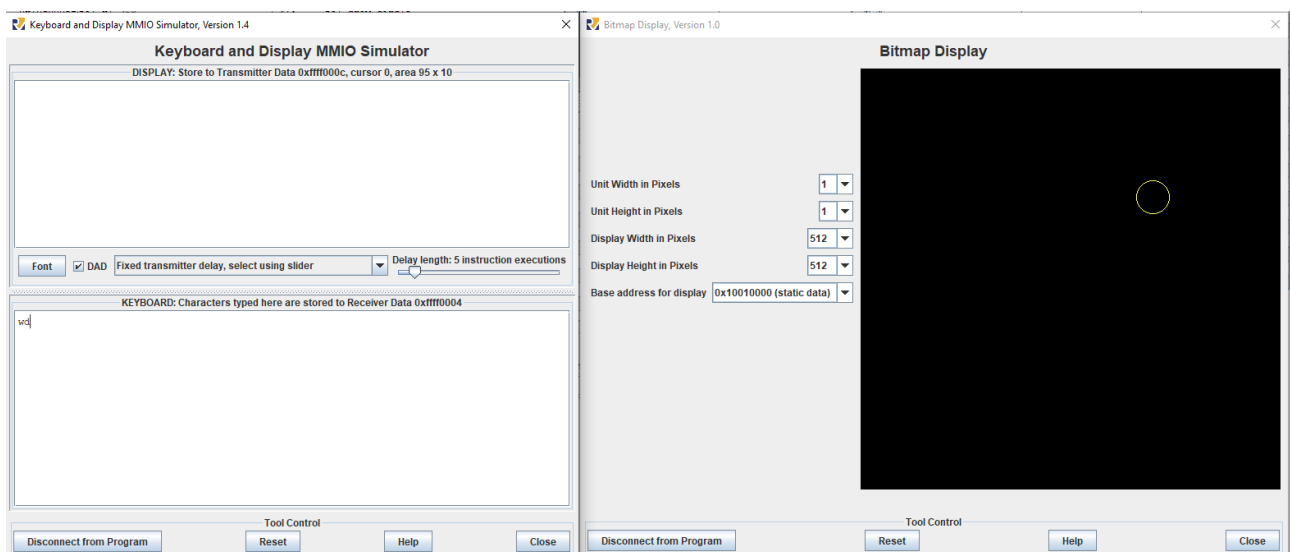


Figure 3: The ball moved right when press D or d

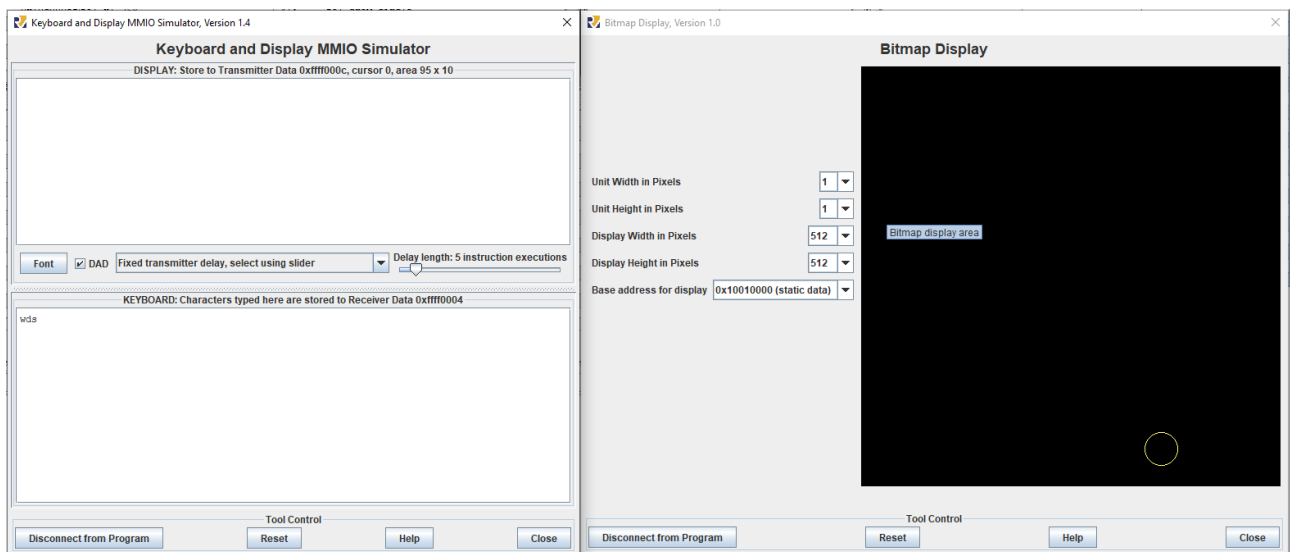


Figure 4: The ball moved down when press S or s

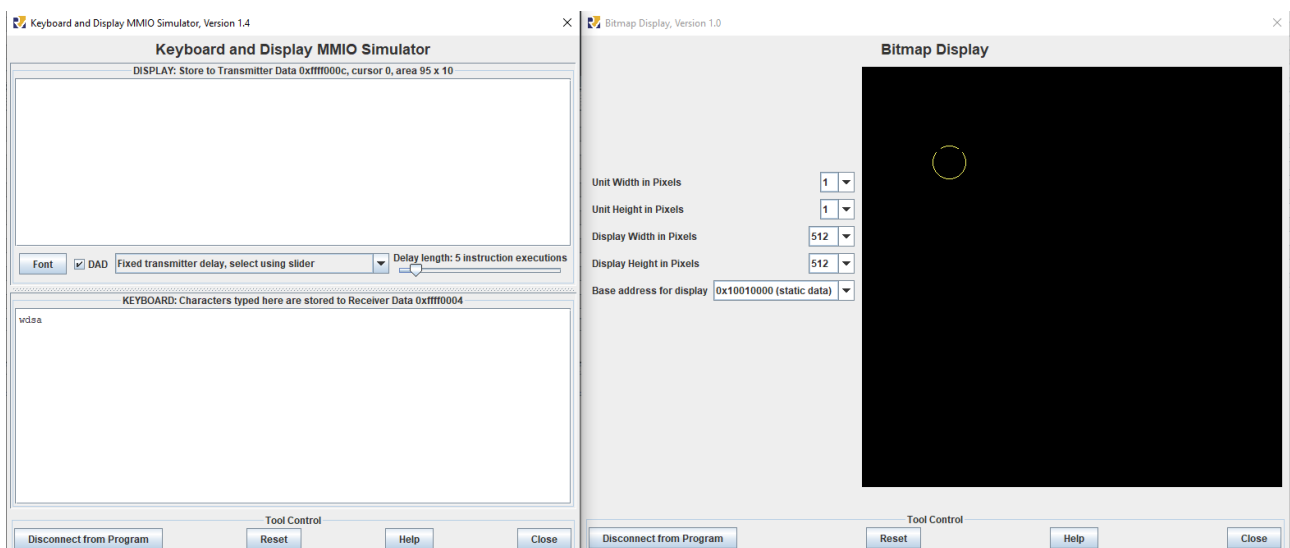


Figure 5: The ball moved left when press A or a

3 PROBLEM 9: TESTING SORTING ALGORITHMS

graphicx

3.1 Problem Statement

Create a program to read data from a text file with up to 10,000 random integers separated by spaces. The user will type in the file name, and the application will read the numbers into memory. The user may then choose one of the four sorting algorithms: Bubble Sort, Insertion Sort, Selection Sort and Quick Sort. The program will run the chosen algorithm, record the execution time, count the total number of instructions performed, and save the sorted results to an output file.

Requirement:

- The application must be able to open and read a file containing random integers up to 10,000. The numbers must be read and saved to memory as a list or array.
- The user can select one of four sorting algorithms: Bubble Sort, Insertion Sort, Selection Sort and Quick Sort.
- The application must measure and report the sorting algorithm's execution duration, as well as determine the total number of instructions performed throughout the sorting process.
- The user interface should be straightforward, allowing for file entry, algorithm selection, and easy-to-read result presentation.
- The application should handle problems when the file does not exist or is in the improper format, as well as ensuring that the numbers in the file are valid integers.

3.2 Algorithm Idea

In this problem, we implement several sorting algorithms to process an array of integers, read from a file, and sort as required. Additionally, the algorithm marks negative numbers in the array using a bitmask array, helping to quickly identify negative numbers during processing.

Initialization

- Initialise the numbers array, which will be used to hold the integers read from the file. Initially, the array is empty.
- Variable count: This variable keeps track of the number of elements in the array. Initially, count is set to zero.
- Time Variables: Variables such as `start_time` and `end_time` are initialised to track the sorting algorithm's execution time.
- Read Data from File: The data will be read into the numbers array after the file containing the integers has been opened. The application will alert the user and request a proper file name if the file is not found or if there is a reading problem.
- Initialise Sorting Variables: Sorting algorithms like Bubble Sort and Quick Sort employ variables (such array indices) that have been initialised.

Handling Keyboard Input

- **Enter File Name:** The application requests that the user provide the name of the file containing the numbers. The application will ask the user to enter the file name again if it doesn't exist until a legitimate file is located.
- **Verify File Name Validity:** The application verifies if the file exists after the user inputs the file name. If not, it will prompt the user to provide the file name once again until a legitimate file is supplied.
- **Read Data from File:** The integers in the file will be read and saved in the numbers array once a valid file name has been supplied. Until all of the data in the file has been processed, each integer will be added to the array.

Checking Boundaries

- **Verify for Negative Values:** The numbers are examined for negative values while they are being read from the file. The bitmask array will be used to flag negative values for treatment at a later time. Only legitimate (non-negative) numbers are added to the numbers array, thanks to the application.
- **Verify Array Indexes During Operations:** When the sorting algorithm is being executed, comparisons and swaps are always carried out inside the acceptable boundaries of the array of integers. The program will halt and raise an error if any operation goes above the array's limits.
- **Array Index Boundaries:** When using sorting algorithms (such as Bubble Sort and Quick Sort), the software makes sure that array indices don't go beyond the array's size to avoid out-of-bounds access.

Sorting Algorithm

- **Bubble Sort:** A simple algorithm that alternates neighbouring items in a list if they are out of order after periodically comparing them. This method is carried out again until the array is sorted and no swaps are required. The largest unsorted element "bubbles" up to its proper location after each pass. In the worst scenario, Bubble Sort's temporal complexity is $O(n^2)$.
- **Selection Sort:** From the unsorted section, the algorithm continually chooses the smallest (or biggest) element and replaces it with the first unsorted element. With a temporal complexity of $O(n^2)$ in any scenario, this operation keeps going until the full array is sorted.
- **Quick Sort:** This divide-and-conquer method divides the array into two sub-arrays after choosing a pivot element. Until the array is sorted, the same process is used recursively to the left and right sub-arrays. The temporal complexity of Quick Sort is $O(n \log n)$ on average, but it can be $O(n^2)$ in the worst situation (with a faulty pivot).
- **Insertion Sort:** One element at a time, this algorithm creates the sorted array. With a time complexity of $O(n^2)$ in the worst case and $O(n)$ in the best, it selects the subsequent element from the unsorted section and places it in the appropriate location in the sorted section.

Results of the Output

- **Print Sorted Array:** To allow the user to confirm the outcome, the software will print the sorted array after the sorting procedure is complete.
- **Execution Time:** To notify the user, the sorting algorithm's execution time will be computed and shown.
- **Save Results:** The sorted results can be stored for later use in a new file if necessary.

Exception Handling

- **File Handling Errors:** The application will alert the user and ask for a suitable file name if an error occurs when accessing the file or reading data from it (such as file not found).
- **Input Validation Errors:** The application will prompt the user to re-enter the data until valid data is entered if the user enters invalid data.

3.3 Implementation Pipeline

Input File Handling:

- The program prompts the user to enter a filename using the message "Enter filename: "
- A 256-byte buffer (input_filename) is allocated to store the input filename
- A newline character removal process is implemented to clean the filename input
- File opening is performed using the RISC-V syscall 1024 (open file)
- Error handling is integrated with a specific error message if file opening fails

```

1  # Prompt for filename
2  li a7, 4
3  la a0, msg_prompt_input
4  ecall
5
6  # Read filename
7  li a7, 8
8  la a0, input_filename
9  li a1, 256
10 ecall
11
12 # Remove newline from filename
13 remove_newline_from_filename:
14     lb t1, 0(t0)
15     beqz t1, open_input_file
16     li t2, 10
17     beq t1, t2, replace_null
18     addi t0, t0, 1
19     j remove_newline_from_filename

```

Number Reading Strategy

- The (read_numbers) function implements a sophisticated number parsing algorithm

- Supports reading both positive and negative integers
- Handles different input scenarios:
 - Recognizes numbers separated by spaces or newline characters
 - Detects negative numbers by identifying the '-' sign
 - Converts ASCII characters to integer values
 - Builds numbers digit by digit
- Stores parsed numbers in a pre-allocated numbers array
- Maintains a count variable to track the total number of integers read

```

1 read_loop:
2     # Read one character from file
3     li a7, 63          # ReadFile syscall
4     lw a0, fd          # File descriptor
5     la a1, file_read_buffer # Buffer address
6     li a2, 1           # Read one character
7     ecall
8
9     # Check for negative number
10    li t3, 45          # ASCII for '-'
11    bne t2, t3, not_char_minus
12    beqz t1, set_negative # Set negative only at number start
13
14    # Convert ASCII to number
15    addi t2, t2, -48    # Convert ASCII to digit
16    li t3, 10
17    mul t0, t0, t3      # Current number * 10
18    add t0, t0, t2      # Add new digit

```

Sorting Algorithms Implementation

The program provides four distinct sorting algorithms, each implemented with a similar structural approach:

- **Bubble Sort:**
 - Implements a classic bubble sort algorithm
 - Outer and inner loop structure for comparing and swapping adjacent elements
 - Time complexity: $O(n^2)$
 - Iterates through the array multiple times, progressively "bubbling up" larger elements
- **Insertion Sort:**
 - Uses an in-place insertion sort strategy
 - Builds the sorted portion of the array incrementally
 - Time complexity: $O(n^2)$
 - Efficiently handles partially sorted arrays
 - Shifts elements to create space for inserting the current element
- **Selection Sort:**
 - Implements selection sort with a two-loop mechanism
 - Finds the minimum element in each iteration and places it in its correct position
 - Time complexity: $O(n^2)$
 - Consistently divides the array into sorted and unsorted regions

- **Quick Sort:**

- Utilizes the Quicksort algorithm with a recursive approach
- Employs a partitioning strategy to divide the array
- Chooses the last element as the pivot
- Recursively sorts sub-arrays on both sides of the pivot
- Average time complexity: $O(n \log n)$
- More efficient for larger datasets compared to other implemented algorithms

Performance Measurement

- **Execution Time Tracking**

- Uses syscall 30 to capture start and end timestamps
- Calculates execution time in milliseconds
- Prints execution time after each sorting operation

```

1 get_time:
2     li a7, 30          # GetTime syscall
3     ecall
4     ret
5
6 print_time:
7     # Calculate execution time
8     la t0, start_time
9     lw t1, 0(t0)
10    la t0, end_time
11    lw t2, 0(t0)
12    sub t3, t2, t1      # Execution time
13
14    # Print execution time
15    li a7, 4
16    la a0, msg_execution_time
17    ecall
18
19    li a7, 1
20    mv a0, t3
21    ecall

```

Negative Number Handling:

- **Bit Masking Technique:** A unique feature of the implementation is the negative number tracking
 - Implements a custom bit masking approach for tracking negative numbers
 - Allocates a separate neg_bitmask to mark negative number positions
 - Efficiently uses bitwise operations to flag negative integers

```

1 flag_negative_numbers:
2     mv s0, a0          # Array address
3     mv s1, a1          # Array size
4     li s2, 0           # Counter
5
6 flag_loop:
7     bge s2, s1, flag_done

```

```

8
9     # Get current number
10    slli t0, s2, 2
11    add t0, s0, t0
12    lw t1, 0(t0)      # Get number
13
14    # Skip if positive
15    bgez t1, skip_flag
16
17    # Calculate bit position in bitmask
18    mv t0, s2          # Copy index
19    srai t1, t0, 3     # Byte offset = index / 8
20    andi t2, t0, 0x7   # Bit position = index % 8
21    li t3, 1
22    sll t3, t3, t2     # Shift 1 to exact bit position
23
24    # Set bit in bitmask
25    la t4, neg_bitmask
26    add t4, t4, t1     # Add byte offset
27    lb t5, 0(t4)       # Get current byte
28    or t5, t5, t3      # Set bit
29    sb t5, 0(t4)       # Save updated byte
30
31 skip_flag:
32     addi s2, s2, 1
33     j flag_loop

```

Output Processing

• File Writing Mechanism

- Creates an output file at a predefined path (/Users//Documents/Prj/Final/output.txt)
- Writes sorted numbers to the file
- Includes error handling for file writing operations
- Converts integers back to string representation for writing
- Handles spacing between numbers in the output file

```

1 write_results:
2     # Open output file
3     li a7, 1024      # Open file syscall
4     la a0, output_filename
5     li a1, 1         # Write mode
6     li a2, 0x1ff     # File permissions
7     ecall
8
9     # Write numbers to file
10    li s0, 0         # Counter
11    lw s1, count      # Total numbers
12    la s2, numbers    # Array address
13
14 write_loop:
15     bge s0, s1, write_done
16
17     # Get current number
18     slli t0, s0, 2
19     add t1, s2, t0
20     lw t2, 0(t1)     # Get number

```

```

21
22     # Convert number to string
23     la a0, buffer_number
24     mv a1, t2
25     jal number_to_string
26
27     # Write number to file
28     li a7, 64           # WriteFile syscall
29     lw a0, out_fd
30     la a1, buffer_number
31     mv a2, t3           # Exact length
32     ecall

```

Auxiliary Functions

Several utility functions support the main implementation:

- `number_to_string`: Converts integers to string representation
- `str_reverse`: Reverses strings in-place
- `flag_negative_numbers`: Marks negative number positions
- Error handling and input validation functions

```

1  number_to_string:
2      # a0 = buffer address
3      # a1 = number to convert
4      mv s0, a0           # Save buffer address
5      mv s1, a1           # Save number to convert
6      li s2, 0            # Length counter
7      li s3, 0            # Negative flag
8
9      # Handle zero case
10     bnez s1, check_sign
11     li t0, 48            # ASCII '0'
12     sb t0, 0(s0)
13     li a0, 1
14     j num_to_str_done
15
16  check_sign:
17     # Check for negative number
18     bgez s1, convert_digits
19     li s3, 1             # Set negative flag
20     neg s1, s1           # Make positive
21
22  convert_digits:
23     # Convert digits in reverse order
24     mv t0, s0
25  digit_loop:
26     beqz s1, finalize_string
27     li t1, 10
28     rem t2, s1, t1       # Get last digit
29     div s1, s1, t1
30     addi t2, t2, 48      # Convert to ASCII
31     sb t2, 0(t0)
32     addi t0, t0, 1
33     addi s2, s2, 1
34     j digit_loop
35  str_reverse:

```



```

36      # a0 = start address
37      # a1 = end address
38      bge a0, a1, str_rev_done
39
40      # Swap characters
41      lb t0, 0(a0)
42      lb t1, 0(a1)
43      sb t1, 0(a0)
44      sb t0, 0(a1)
45
46      # Move pointers
47      addi a0, a0, 1
48      addi a1, a1, -1
49      j str_reverse
50
51 str_rev_done:
52      ret

```

Memory Management

- Efficient use of stack for storing temporary variables
- Careful register allocation and preservation
- Minimizes memory overhead during sorting operations

```

1 quick_sort_logic:
2     # Allocate stack space and save registers
3     addi sp, sp, -24
4     sw ra, 20(sp)      # Return address
5     sw s0, 16(sp)      # Saved register 0
6     sw s1, 12(sp)      # Saved register 1
7     sw s2, 8(sp)       # Saved register 2
8     sw s3, 4(sp)       # Saved register 3
9     sw s4, 0(sp)       # Saved register 4
10
11     # Function body...
12
13     # Restore registers and deallocate stack
14     lw ra, 20(sp)
15     lw s0, 16(sp)
16     lw s1, 12(sp)
17     lw s2, 8(sp)
18     lw s3, 4(sp)
19     lw s4, 0(sp)
20     addi sp, sp, 24
21     ret

```

User Interaction

- Provides a menu-driven interface for sorting algorithm selection
- Allows multiple sorting attempts on the same input data
- Offers an option to exit the program

```

1 menu_loop:
2     # Display menu options
3     li a7, 4

```

```

4      la a0, menu
5      ecall
6
7      # Menu string content
8      # "\nUser select sorting algorithm:
9      # 1. Bubble Sort
10     # 2. Insertion Sort
11     # 3. Selection Sort
12     # 4. Quick Sort
13     # 5. Close
14     # Choice: "
15
16     # Read user choice
17     li a7, 5
18     ecall
19
20     # Branch to selected sorting algorithm
21     li t0, 1
22     beq a0, t0, bubble_sort_array
23     li t0, 2
24     beq a0, t0, insertion_sort_array
25     li t0, 3
26     beq a0, t0, selection_sort_array
27     li t0, 4
28     beq a0, t0, quick_sort_array
29     li t0, 5
30     beq a0, t0, exit
31
32     # Invalid choice handling implicitly falls through to menu
33     j menu_loop
34
35 # Exit program
36 exit:
37     li a7, 10          # Exit syscall
38     ecall
39 file_error_open:
40     # Display error message for file opening failure
41     li a7, 4
42     la a0, error_msg   # "\nError opening file\n"
43     ecall
44
45     # Return to main menu or exit
46     j exit
47
48 msg_file_error_openor:
49     # Error message for file writing failure
50     li a7, 4
51     la a0, msg_file_error_open
52     ecall
53
54     # Cleanup and return
55     lw ra, 12(sp)
56     lw s0, 8(sp)
57     lw s1, 4(sp)
58     lw s2, 0(sp)
59     addi sp, sp, 16
60     ret

```

3.4 Source Code

```

1  .data
2      # Phan du lieu hien co
3      numbers:      .space  80000
4      input_buffer_size:  .word  80000
5      count:        .word  0
6
7      # Them mang bitmask cho so am (1 bit moi so)
8      # Voi 80000 byte so (20000 so nguyen), chung ta can 20000 bit = 2500
9      byte
10     neg_bitmask:    .space  2500
11
12     input_filename: .space  256
13     file_read_buffer: .space 1024
14     msg_prompt_input: .string "Enter filename: "
15     error_msg: .string "\nError opening file\n"
16     menu: .string "\nUser select sorting algorithm:\n1. Bubble Sort\n2
17             . Insertion Sort\n3. Selection Sort\n4. Quick Sort\n5.Close\nChoice:
18             "
19
20     fd: .word 0
21     newline: .string "\n"
22     space: .string " "
23     start_time: .word 0
24     end_time: .word 0
25
26     msg_execution_time: .string "\nExecution time (ms): "
27
28     # Du lieu moi cho file output
29     output_filename: .string "/Users/truonglinhduyen/Documents/Prj/Final/
30         ouuuu.txt"
31     out_fd: .word 0
32     buffer_number: .space 12
33     msg_file_error_open: .string "\nError writing to output file\n"
34     char_minus: .string "-"
35     # Them buffer tam cho sap xep tron
36     tmp_sort_buffer: .space 80000
37
38  .text
39  .globl main
40  main:
41      # In ra msg_prompt_input
42      li a7, 4
43      la a0, msg_prompt_input
44      ecall
45      # Doc input_filename
46      li a7, 8
47      la a0, input_filename
48      li a1, 256
49      ecall
50      # Loai bo newline khoi input_filename
51      la t0, input_filename
52  remove_newline_from_filename:
53      lb t1, 0(t0)
54      beqz t1, open_input_file
55      li t2, 10
56      beq t1, t2, replace_null
57      addi t0, t0, 1
58      j remove_newline_from_filename
59  replace_null:

```

```

55     sb zero, 0(t0)
56
57 open_input_file:
58     li a7, 1024
59     la a0, input_filename
60     li a1, 0
61     ecall
62     bltz a0, file_error_open
63     la t1, fd
64     sw a0, 0(t1)
65     jal read_numbers
66
67 menu_loop:
68     # Hien thi menu
69     li a7, 4
70     la a0, menu
71     ecall
72
73     # Doc lua chon
74     li a7, 5
75     ecall
76
77     # Nhanh den lua chon
78     li t0, 1
79     beq a0, t0, bubble_sort_array
80     li t0, 2
81     beq a0, t0, insertion_sort_array
82     li t0, 3
83     beq a0, t0, selection_sort_array
84     li t0, 4
85     beq a0, t0, quick_sort_array
86     li t0, 5
87     beq a0, t0, exit
88     j exit
89 file_error_open:
90     # In ra thong bao loi
91     li a7, 4
92     la a0, error_msg
93     ecall
94     j exit
95 read_numbers:
96     addi sp, sp, -16
97     sw ra, 12(sp)
98     sw s0, 8(sp)
99     sw s1, 4(sp)
100    sw s2, 0(sp)
101
102    # Reset count
103    la t1, count          # Tai dia chi cua count
104    sw zero, 0(t1)        # Luu gia tri zero vao count
105
106    # Khoi tao bien
107    li t0, 0              # So hien tai dang duoc xay dung
108    li t1, 0              # Co cho biet dang trong so
109    li t6, 0              # Co danh dau (0 = duong, 1 = am)
110
111 read_loop:
112    # Doc mot ky tu tu file
113    li a7, 63              # Syscall ReadFile
114    lw a0, fd              # Mo ta file

```

```

115     la a1, file_read_buffer      # Dia chi buffer
116     li a2, 1                    # Doc mot ky tu
117     ecall
118
119     # Kiem tra neu cuoi file
120     beqz a0, read_done
121
122     # Tai ky tu
123     lb t2, 0(a1)
124
125     # Kiem tra dau tru
126     li t3, 45                    # ASCII cho '-'
127     bne t2, t3, not_char_minus
128     beqz t1, set_negative        # Chi dat am neu o dau so
129     j read_loop
130
131 set_negative:
132     li t6, 1                     # Dat co danh dau la am
133     li t1, 1                     # Dat co trong so
134     j read_loop
135
136 not_char_minus:
137     # Kiem tra neu la dau cach hoac newline
138     li t3, 32                    # Space
139     beq t2, t3, save_number
140     li t3, 10                    # Newline
141     beq t2, t3, save_number
142
143     # Chuyen doi ASCII thanh so va cong vao so hien tai
144     addi t2, t2, -48              # Chuyen doi ASCII thanh so
145     li t3, 10
146     mul t0, t0, t3                # So hien tai * 10
147     add t0, t0, t2                # Cong vao chu so moi
148     li t1, 1                     # Dat co trong so
149     j read_loop
150
151 save_number:
152     beqz t1, read_loop           # Neu khong trong so, tiep tuc
153
154     # Ap dung dau neu la am
155     beqz t6, save_positive
156     neg t0, t0                   # Doi dau so neu co am duoc dat
157
158 save_positive:
159     # Luu so vao mang
160     la t3, count                 # Tai dia chi cua count
161     lw t3, 0(t3)                 # Tai gia tri count
162     slli t4, t3, 2               # t4 = count * 4
163     la t5, numbers
164     add t5, t5, t4
165     sw t0, 0(t5)                 # Luu so
166
167     # Tang count
168     addi t3, t3, 1
169     la t4, count                 # Tai dia chi cua count
170     sw t3, 0(t4)                 # Luu count moi
171
172     # Reset cho so tiep theo
173     li t0, 0                     # Reset so hien tai
174     li t1, 0                     # Reset co trong so

```

```

175     li t6, 0                # Reset co am
176     j read_loop
177
178 read_done:
179     # Neu chung ta dang trong so khi file ket thuc, luu no
180     beqz t1, close_file
181
182     # Ap dung dau neu la am
183     beqz t6, save_last_positive
184     neg t0, t0              # Doi dau so neu co am duoc dat
185
186 save_last_positive:
187     la t3, count           # Tai dia chi cua count
188     lw t3, 0(t3)          # Tai gia tri count
189     slli t4, t3, 2
190     la t5, numbers
191     add t5, t5, t4
192     sw t0, 0(t5)
193     addi t3, t3, 1
194     la t4, count           # Tai dia chi cua count
195     sw t3, 0(t4)          # Luu count moi
196
197 close_file:
198     # Dong file
199     li a7, 57              # Syscall Close file
200     lw a0, fd
201     ecalls
202
203     lw ra, 12(sp)
204     lw s0, 8(sp)
205     lw s1, 4(sp)
206     lw s2, 0(sp)
207     addi sp, sp, 16
208     ret
209
210 flag_negative_numbers:
211     # a0 = dia chi mang
212     # a1 = size
213     addi sp, sp, -16
214     sw ra, 12(sp)
215     sw s0, 8(sp)
216     sw s1, 4(sp)
217     sw s2, 0(sp)
218
219     mv s0, a0              # s0 = dia chi mang
220     mv s1, a1              # s1 = size
221     li s2, 0              # s2 = bo dem
222
223 flag_loop:
224     bge s2, s1, flag_done
225
226     # Tai so hien tai
227     slli t0, s2, 2         # t0 = bo dem * 4
228     add t0, s0, t0
229     lw t1, 0(t0)          # Tai so
230
231     # Bo qua neu duong
232     bgez t1, skip_flag
233
234     # Tinh toan byte va vi tri bit trong bitmask

```

```

235     mv t0, s2           # Sao chep chi so
236     srai t1, t0, 3      # Byte offset = chi so / 8
237     andi t2, t0, 0x7    # Vi tri bit = chi so % 8
238     li t3, 1
239     sll t3, t3, t2       # Dich chuyen 1 den vi tri bit chinh xac
240
241     # Dat bit trong bitmask
242     la t4, neg_bitmask
243     add t4, t4, t1       # Them byte offset
244     lb t5, 0(t4)         # Tai byte hien tai
245     or t5, t5, t3        # Dat bit
246     sb t5, 0(t4)        # Luu byte da cap nhat
247
248 skip_flag:
249     addi s2, s2, 1
250     j flag_loop
251
252 flag_done:
253     lw ra, 12(sp)
254     lw s0, 8(sp)
255     lw s1, 4(sp)
256     lw s2, 0(sp)
257     addi sp, sp, 16
258     ret
259 quick_sort_array:
260     # Lay thoi gian bat dau
261     jal get_time
262     sw a0, start_time, t0
263
264     # Khoi tao quicksort
265     la a0, numbers
266     li a1, 0
267     lw a2, count
268     addi a2, a2, -1
269     jal quick_sort_logic
270
271     # Danh dau cac so am
272     la a0, numbers
273     lw a1, count
274     jal flag_negative_numbers
275
276     # Lay thoi gian ket thuc va tinh thoi gian thuc thi
277     jal get_time
278     sw a0, end_time, t0
279     jal print_time
280
281     # Ghi ket qua vao file
282     jal write_results
283     j menu_loop
284
285 quick_sort_logic:
286     # a0 = dia chi mang
287     # a1 = chi so ben trai
288     # a2 = chi so ben phai
289     addi sp, sp, -24
290     sw ra, 20(sp)
291     sw s0, 16(sp)
292     sw s1, 12(sp)
293     sw s2, 8(sp)
294     sw s3, 4(sp)

```

```

295     sw s4, 0(sp)
296
297     # Luu cac tham so
298     mv s0, a0          # s0 = dia chi mang
299     mv s1, a1          # s1 = ben trai
300     mv s2, a2          # s2 = ben phai
301
302     # Dieu kien dung: neu ben trai >= ben phai, thoat
303     bge s1, s2, quick_sort_end
304
305     # Goi ham partition_elements
306     mv a0, s0          # dia chi mang
307     mv a1, s1          # chi so ben trai
308     mv a2, s2          # chi so ben phai
309     jal partition_elements
310     mv s3, a0          # s3 = chi so pivot
311
312     # De quy sap xep phan ben trai
313     mv a0, s0          # dia chi mang
314     mv a1, s1          # chi so ben trai
315     addi a2, s3, -1    # pivot - 1
316     jal quick_sort_logic
317
318     # De quy sap xep phan ben phai
319     mv a0, s0          # dia chi mang
320     addi a1, s3, 1     # pivot + 1
321     mv a2, s2          # chi so ben phai
322     jal quick_sort_logic
323
324 quick_sort_end:
325     lw ra, 20(sp)
326     lw s0, 16(sp)
327     lw s1, 12(sp)
328     lw s2, 8(sp)
329     lw s3, 4(sp)
330     lw s4, 0(sp)
331     addi sp, sp, 24
332     ret
333
334 partition_elements:
335     addi sp, sp, -24
336     sw ra, 20(sp)
337     sw s0, 16(sp)
338     sw s1, 12(sp)
339     sw s2, 8(sp)
340     sw s3, 4(sp)
341     sw s4, 0(sp)
342
343     mv s0, a0          # s0 = dia chi mang
344     mv s1, a1          # s1 = ben trai
345     mv s2, a2          # s2 = ben phai
346
347     slli t0, s2, 2     # t0 = ben phai * 4
348     add t0, s0, t0
349     lw s3, 0(t0)       # s3 = gia tri pivot
350
351     addi s4, s1, -1    # i = ben trai - 1
352     mv t1, s1          # j = ben trai
353
354 partition_loop_elements:

```



```

355     bge t1, s2, partition_elements_done
356
357     # Tai phan tu hien tai
358     slli t0, t1, 2
359     add t0, s0, t0
360     lw t2, 0(t0)      # t2 = arr[j]
361
362     # So sanh voi pivot
363     bgt t2, s3, skip_swap
364
365     # Tang i
366     addi s4, s4, 1
367
368     # Doi phan tu neu i != j
369     slli t0, s4, 2
370     add t0, s0, t0     # Dia chi cua arr[i]
371     slli t3, t1, 2
372     add t3, s0, t3     # Dia chi cua arr[j]
373
374     lw t4, 0(t0)      # t4 = arr[i]
375     lw t5, 0(t3)      # t5 = arr[j]
376     sw t5, 0(t0)      # arr[i] = arr[j]
377     sw t4, 0(t3)      # arr[j] = arr[i]
378
379 skip_swap:
380     addi t1, t1, 1     # j++
381     j partition_loop_elements
382
383 partition_elements_done:
384     # Dat pivot vao vi tri cuoi cung
385     addi s4, s4, 1     # i++
386
387     # Doi pivot voi phan tu tai i
388     slli t0, s4, 2
389     add t0, s0, t0     # Dia chi cua arr[i]
390     slli t1, s2, 2
391     add t1, s0, t1     # Dia chi cua arr[right]
392
393     lw t2, 0(t0)      # t2 = arr[i]
394     lw t3, 0(t1)      # t3 = arr[right]
395     sw t3, 0(t0)      # arr[i] = arr[right]
396     sw t2, 0(t1)      # arr[right] = arr[i]
397
398     # Tra ve chi so pivot
399     mv a0, s4
400
401     lw ra, 20(sp)
402     lw s0, 16(sp)
403     lw s1, 12(sp)
404     lw s2, 8(sp)
405     lw s3, 4(sp)
406     lw s4, 0(sp)
407     addi sp, sp, 24
408     ret
409 #####
410 # bubble_sort
411 #####
412 bubble_sort_array:
413     # Lay thoi gian bat dau
414     jal get_time

```

```

415     sw a0, start_time, t0
416
417     # Thuc hien bubble sort
418     la a0, numbers      # Tai dia chi mang
419     lw a1, count        # Tai kich thuoc mang
420     jal bubble_sort_core
421     # Danh dau cac so am
422     la a0, numbers
423     lw a1, count
424     jal flag_negative_numbers
425     # Lay thoi gian ket thuc va tinh thoi gian thuc thi
426     jal get_time
427     sw a0, end_time, t0
428     jal print_time
429
430     # Ghi ket qua vao file
431     jal write_results
432     j exit
433 #####
434 # insertion_sort_array:
435 #####
436 insertion_sort_array:
437     # Lay thoi gian bat dau
438     jal get_time
439     sw a0, start_time, t0
440
441     # Thuc hien insertion sort
442     la a0, numbers      # Tai dia chi mang
443     lw a1, count        # Tai kich thuoc mang
444     jal insertion_sort_array_impl
445     # Danh dau cac so am
446     la a0, numbers
447     lw a1, count
448     jal flag_negative_numbers
449     # Lay thoi gian ket thuc va tinh thoi gian thuc thi
450     jal get_time
451     sw a0, end_time, t0
452     jal print_time
453
454     # Ghi ket qua vao file
455     jal write_results
456     j exit
457 #####
458 # selection_sort_array:
459 #####
460 selection_sort_array:
461     # Lay thoi gian bat dau
462     jal get_time
463     sw a0, start_time, t0
464
465     # Thuc hien selection sort
466     la a0, numbers      # Tai dia chi mang
467     lw a1, count        # Tai kich thuoc mang
468     jal selection_sort_array_impl
469
470     # Danh dau cac so am
471     la a0, numbers
472     lw a1, count
473     jal flag_negative_numbers
474

```

```

475     # Lay thoi gian ket thuc va tinh thoi gian thuc thi
476     jal get_time
477     sw a0, end_time, t0
478     jal print_time
479
480     # Ghi ket qua vao file
481     jal write_results
482     j exit
483
484 bubble_sort_core:
485     # a0 = dia chi mang
486     # a1 = kich thuoc
487     addi sp, sp, -16
488     sw ra, 12(sp)
489     sw s0, 8(sp)
490     sw s1, 4(sp)
491     sw s2, 0(sp)
492
493     mv s0, a0          # s0 = dia chi mang
494     mv s1, a1          # s1 = kich thuoc
495     li s2, 0           # s2 = i
496
497 outer_loop_bubble_sort:
498     bge s2, s1, bubble_done
499     li t0, 0           # j = 0
500
501 inner_loop_bubble_sort:
502     sub t1, s1, s2
503     addi t1, t1, -1
504     bge t0, t1, inner_done_bubble_sort
505
506     # So sanh cac phan tu ke tiep nhau
507     slli t2, t0, 2
508     add t2, s0, t2
509     lw t3, 0(t2)       # arr[j]
510     lw t4, 4(t2)       # arr[j+1]
511
512     ble t3, t4, no_swap_bubble_sort
513
514     # Hoan doi cac phan tu
515     sw t4, 0(t2)
516     sw t3, 4(t2)
517
518 no_swap_bubble_sort:
519     addi t0, t0, 1
520     j inner_loop_bubble_sort
521
522 inner_done_bubble_sort:
523     addi s2, s2, 1
524     j outer_loop_bubble_sort
525
526 bubble_done:
527     lw ra, 12(sp)
528     lw s0, 8(sp)
529     lw s1, 4(sp)
530     lw s2, 0(sp)
531     addi sp, sp, 16
532     ret
533
534 insertion_sort_array_impl:

```

```

535     # a0 = dia chi mang
536     # a1 = kich thuoc
537     addi sp, sp, -16
538     sw ra, 12(sp)
539     sw s0, 8(sp)
540     sw s1, 4(sp)
541     sw s2, 0(sp)
542
543     mv s0, a0          # s0 = dia chi mang
544     mv s1, a1          # s1 = kich thuoc
545     li s2, 1           # s2 = i = 1
546
547 outer_loop_insertion:
548     bge s2, s1, insertion_done
549
550     # Lay phan tu hien tai
551     slli t0, s2, 2      # t0 = i * 4
552     add t0, s0, t0
553     lw t1, 0(t0)        # key = arr[i]
554     addi t2, s2, -1     # j = i-1
555
556 inner_loop_insertion:
557     bltz t2, inner_done_insertion    # neu j < 0, thoat
558
559     # So sanh cac phan tu
560     slli t3, t2, 2
561     add t3, s0, t3
562     lw t4, 0(t3)        # arr[j]
563
564     ble t4, t1, inner_done_insertion
565
566     # Di chuyen phan tu
567     sw t4, 4(t3)        # arr[j+1] = arr[j]
568
569     addi t2, t2, -1     # j--
570     j inner_loop_insertion
571
572 inner_done_insertion:
573     # Dat key vao vi tri chinh xac
574     addi t2, t2, 1
575     slli t3, t2, 2
576     add t3, s0, t3
577     sw t1, 0(t3)
578
579     addi s2, s2, 1      # i++
580     j outer_loop_insertion
581
582 insertion_done:
583     lw ra, 12(sp)
584     lw s0, 8(sp)
585     lw s1, 4(sp)
586     lw s2, 0(sp)
587     addi sp, sp, 16
588     ret
589
590 selection_sort_array_impl:
591     # a0 = dia chi mang
592     # a1 = kich thuoc
593     addi sp, sp, -16
594     sw ra, 12(sp)

```

```

595     sw s0, 8(sp)
596     sw s1, 4(sp)
597     sw s2, 0(sp)
598
599     mv s0, a0          # s0 = dia chi mang
600     mv s1, a1          # s1 = kich thuoc
601     li s2, 0           # s2 = i
602
603 outer_loop_selection:
604     addi t0, s1, -1
605     bge s2, t0, selection_done
606
607     mv t1, s2           # min_idx = i
608     addi t2, s2, 1      # j = i + 1
609
610 inner_loop_selection:
611     bge t2, s1, inner_done_selection
612
613     # So sanh cac phan tu
614     slli t3, t2, 2
615     add t3, s0, t3
616     lw t4, 0(t3)        # arr[j]
617
618     slli t5, t1, 2
619     add t5, s0, t5
620     lw t6, 0(t5)        # arr[min_idx]
621
622     bge t4, t6, no_update_min
623     mv t1, t2           # Cap nhat min_idx
624
625 no_update_min:
626     addi t2, t2, 1
627     j inner_loop_selection
628
629 inner_done_selection:
630     # Hoan doi cac phan tu neu can
631     beq t1, s2, no_swap_selection
632
633     slli t2, s2, 2
634     add t2, s0, t2
635     lw t3, 0(t2)        # temp = arr[i]
636
637     slli t4, t1, 2
638     add t4, s0, t4
639     lw t5, 0(t4)        # arr[min_idx]
640
641     sw t5, 0(t2)        # arr[i] = arr[min_idx]
642     sw t3, 0(t4)        # arr[min_idx] = temp
643
644 no_swap_selection:
645     addi s2, s2, 1
646     j outer_loop_selection
647
648 selection_done:
649     lw ra, 12(sp)
650     lw s0, 8(sp)
651     lw s1, 4(sp)
652     lw s2, 0(sp)
653     addi sp, sp, 16
654     ret

```

```

655
656 parse_loop:
657     bge s2, s0, read_loop    # Neu da parse het buffer, doc tiep
658
659     # Doc ky tu
660     add t0, s1, s2
661     lb t1, 0(t0)
662
663     # Kiem tra xem co phai space khong
664     li t2, 32                # ASCII cho space
665     beq t1, t2, next_char
666
667     # Chuyen doi ASCII sang so
668     addi t1, t1, -48          # Chuyen ASCII thanh so
669
670     # Luu so vao mang numbers
671     lw t3, count
672     slli t4, t3, 2           # t4 = count * 4 (de tinh offset)
673     la t5, numbers
674     add t5, t5, t4
675     sw t1, 0(t5)            # Luu so vao mang
676
677     # Tang count
678     addi t3, t3, 1
679     sw t3, count, t6
680
681 next_char:
682     addi s2, s2, 1
683     j parse_loop
684
685 # ===== IN THONG TIN =====
686 get_time:
687     li a7, 30                # Syscall GetTime
688     ecall
689     ret
690
691 print_time:
692     # Tinh toan va in thoi gian thuc thi
693     la t0, start_time
694     lw t1, 0(t0)
695     la t0, end_time
696     lw t2, 0(t0)
697     sub t3, t2, t1           # Thoi gian thuc thi
698
699     li a7, 4
700     la a0, msg_execution_time
701     ecall
702
703     li a7, 1
704     mv a0, t3
705     ecall
706     ret
707
708 # Ham tro giup chuyen doi so thanh chuoi
709 number_to_string:
710     # a0 = dia chi buffer
711     # a1 = so can chuyen doi
712     addi sp, sp, -24
713     sw ra, 20(sp)
714     sw s0, 16(sp)

```

```

715     sw s1, 12(sp)
716     sw s2, 8(sp)
717     sw s3, 4(sp)
718     sw s4, 0(sp)
719
720     mv s0, a0          # Luu dia chi buffer
721     mv s1, a1          # Luu so can chuyen doi
722     li s2, 0           # Khoi tao bo dem do dai
723     li s3, 0           # Co danh dau so am
724
725     # Xu ly truong hop so 0
726     bnez s1, check_sign
727     li t0, 48          # ASCII '0'
728     sb t0, 0(s0)
729     li a0, 1           # Do dai la 1
730     j num_to_str_done
731
732 check_sign:
733     # Kiem tra so am
734     bgez s1, convert_digits
735     li s3, 1           # Dat co danh dau am
736     neg s1, s1          # Chuyen so thanh duong
737
738 convert_digits:
739     # Chuyen doi cac chu so theo thu tu nguoc lai
740     mv t0, s0
741 digit_loop:
742     beqz s1, finalize_string
743     li t1, 10
744     rem t2, s1, t1      # Lay chu so cuoi cung
745     div s1, s1, t1
746     addi t2, t2, 48     # Chuyen thanh ASCII
747     sb t2, 0(t0)
748     addi t0, t0, 1
749     addi s2, s2, 1
750     j digit_loop
751
752 finalize_string:
753     # Them dau tru neu la so am
754     beqz s3, reverse_string
755     li t1, 45          # ASCII '-'
756     sb t1, 0(t0)
757     addi t0, t0, 1
758     addi s2, s2, 1
759
760 reverse_string:
761     mv a0, s0
762     addi a1, t0, -1     # Cuoi chuoi
763     jal str_reverse
764
765     mv a0, s2           # Tra ve do dai
766
767 num_to_str_done:
768     lw ra, 20(sp)
769     lw s0, 16(sp)
770     lw s1, 12(sp)
771     lw s2, 8(sp)
772     lw s3, 4(sp)
773     lw s4, 0(sp)
774     addi sp, sp, 24

```

```

775     ret
776
777 # Ham tro giup dao nguoc chuoi tai cho
778 str_reverse:
779     # a0 = dia chi bat dau
780     # a1 = dia chi ket thuc
781     bge a0, a1, str_rev_done
782
783     # Hoan doi ky tu
784     lb t0, 0(a0)
785     lb t1, 0(a1)
786     sb t1, 0(a0)
787     sb t0, 0(a1)
788
789     # Di chuyen con tro
790     addi a0, a0, 1
791     addi a1, a1, -1
792     j str_reverse
793
794 str_rev_done:
795     ret
796
797 # =====
798 # Ghi ket qua vao file (da cap nhat)
799 # =====
800 write_results:
801     addi sp, sp, -16
802     sw ra, 12(sp)
803     sw s0, 8(sp)
804     sw s1, 4(sp)
805     sw s2, 0(sp)
806
807     # Mo file results.txt de ghi
808     li a7, 1024          # Syscall Open file
809     la a0, output_filename # input_filename
810     li a1, 1             # Chi ghi
811     li a2, 0x1ff         # QUYEN truy cap file (777 trong octal)
812     ecall
813
814     # Kiem tra neu mo file thanh cong
815     bltz a0, msg_file_error_openor
816     sw a0, out_fd, t0 # Luu mo ta file
817
818     # Ghi so vao file
819     li s0, 0             # Khoi tao bo dem
820     lw s1, count         # Tai tong so luong
821     la s2, numbers       # Tai dia chi mang
822
823 write_loop:
824     bge s0, s1, write_done
825
826     # Tai so hien tai
827     slli t0, s0, 2
828     add t1, s2, t0
829     lw t2, 0(t1)         # Tai so
830
831     # Chuyen so thanh chuoi
832     la a0, buffer_number
833     mv a1, t2
834     jal number_to_string

```



```

835     mv t3, a0          # t3 = do dai cua chuoi
836
837     # Ghi so vao file
838     li a7, 64          # Syscall WriteFile
839     lw a0, out_fd
840     la a1, buffer_number
841     mv a2, t3          # Do dai chinh xac
842     ecall
843
844     # Ghi dau cach sau so (tru so cuoi cung)
845     addi t0, s1, -1
846     bge s0, t0, skip_space
847
848     li a7, 64
849     lw a0, out_fd
850     la a1, space
851     li a2, 1
852     ecall
853
854 skip_space:
855     addi s0, s0, 1
856     j write_loop
857
858 write_done:
859     # Ghi newline o cuoi
860     li a7, 64
861     lw a0, out_fd
862     la a1, newline
863     li a2, 1
864     ecall
865
866     # Dong file output
867     li a7, 57          # Syscall Close file
868     lw a0, out_fd
869     ecall
870
871     lw ra, 12(sp)
872     lw s0, 8(sp)
873     lw s1, 4(sp)
874     lw s2, 0(sp)
875     addi sp, sp, 16
876     ret
877
878 msg_file_error_openor:
879     # In thong bao loi
880     li a7, 4
881     la a0, msg_file_error_open
882     ecall
883
884     lw ra, 12(sp)
885     lw s0, 8(sp)
886     lw s1, 4(sp)
887     lw s2, 0(sp)
888     addi sp, sp, 16
889     ret
890 # ===== GHI =====
891
892 write_positive_numbers:
893     # Tai so
894     slli t0, s0, 2

```

```

895     add t1, s2, t0
896     lw t2, 0(t1)
897
898     # Lay gia tri tuyet doi thu cong
899     bgez t2, skip_abs
900     neg t2, t2
901 skip_abs:
902
903     la a0, buffer_number
904     mv a1, t2
905     jal number_to_string
906
907     # Ghi so
908     li a7, 64
909     lw a0, out_fd
910     la a1, buffer_number
911     mv a2, a0          # Do dai tra ve boi number_to_string
912     ecall
913
914     # Ghi dau cach (tru so cuoi cung)
915     addi t0, s1, -1
916     bge s0, t0, skip_space
917
918     li a7, 64
919     lw a0, out_fd
920     la a1, space
921     li a2, 1
922     ecall
923
924 check_negative:
925     bgez s1, positive_conversion    # Neu so >= 0, bo qua xu ly so am
926
927     # Xu ly so am
928     li t0, 45          # ASCII '-'
929     sb t0, 0(s0)       # Luu ky tu tru
930     addi s0, s0, 1     # Di chuyen con tro buffer
931     addi s2, s2, 1     # Tang do dai
932     neg s1, s1         # Chuyen so thanh duong
933
934 positive_conversion:
935     mv t0, s0          # Vi tri hien tai trong buffer
936     mv t1, s1          # Ban sao lam viec cua so
937
938 reverse_digits:
939     mv a0, s0          # Bat dau cua chuoi
940     add a1, s0, s2
941     addi a1, a1, -1    # Cuoi chuoi
942
943     # Them ky tu ket thuc
944     add t0, s0, s2
945     sb zero, 0(t0)
946
947     jal str_reverse
948
949     mv a0, s2          # Tra ve do dai
950
951 exit:
952     li a7, 10          # Syscall Exit
953     ecall

```

3.5 Display Result

Overview of Sorting Algorithm Performance

The implementation compared four distinct sorting algorithms on a common input dataset, measuring their performance across multiple key metrics:

- Execution Time
- Sorted Output
- Computational Complexity

Comparative Analysis of Sorting Algorithms

Bubble Sort

- Execution Time: 15385
- Sorting Type: In-place sorting algorithm
- Best Use Case: Suitable for small datasets or educational purposes
- Time Complexity: $O(n^2)$
- Space Complexity: $O(1)$
- Stability: Yes
- Characteristics: Iteratively compares and swaps adjacent elements until the list is sorted

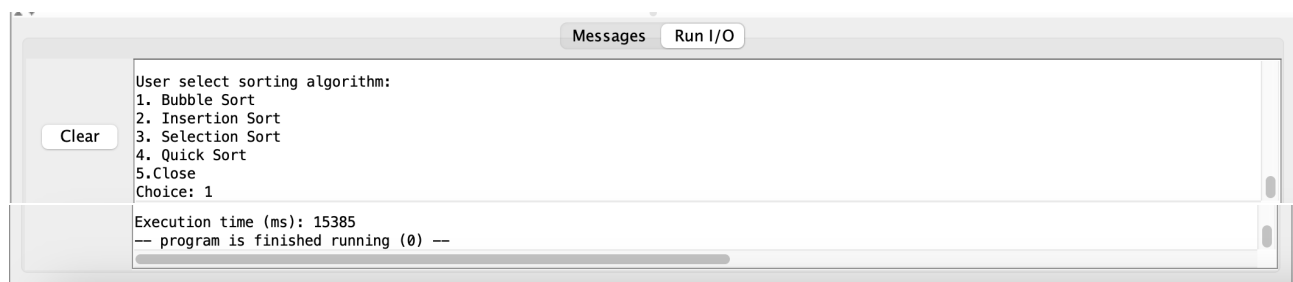


Figure 6: Bubble Sort Result

- **Total number of executed instructions:** 386,407 instructions.
- **Instruction breakdown:**
 - **R-type:** 56,704 instructions (14%).
 - **R4-type:** 0 instructions (0%).
 - **I-type:** 183,332 instructions (47%) – the largest portion, primarily including load and immediate instructions such as `lw` and `addi`.
 - **S-type:** 27,779 instructions (7%).
 - **B-type:** 68,573 instructions (17%) – related to branch instructions like `beq` and `bne` within the Bubble Sort loops.
 - **U-type:** 27,627 instructions (7%).
 - **J-type:** 22,900 instructions (5%) – jump instructions used for loop control.
- **Observations:** The temporal complexity of the Bubble Sort algorithm is $O(n^2)$, which accounts for a large percentage of branch and data-processing instructions. The high proportion of **I-type** and **B-type** instructions reflects the frequent data loading, comparison, and looping operations in the algorithm.

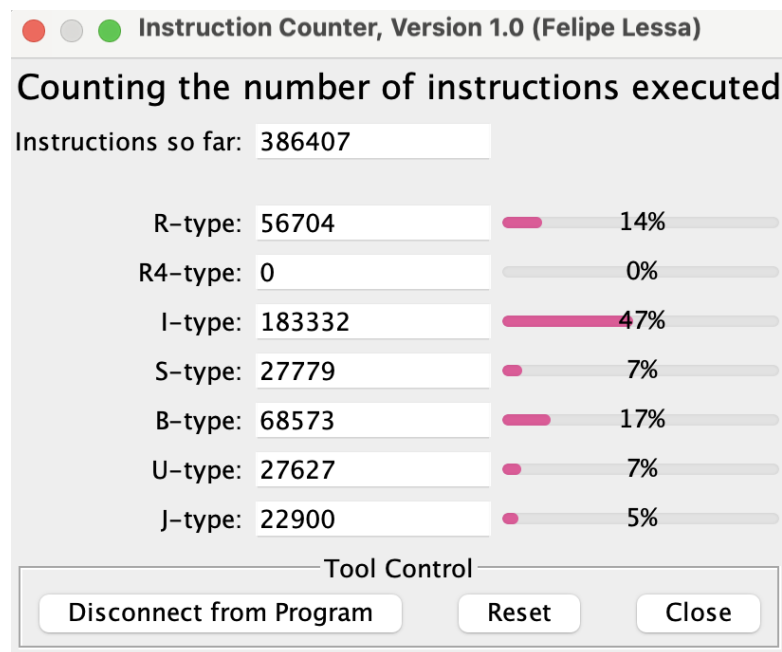


Figure 7: Instruction Count Analysis for Bubble Sort

Insertion Sort

- Execution Time: 5504
- Sorting Type: In-place sorting algorithm
- Best Use Case: Efficient for small or nearly sorted datasets
- Time Complexity: $O(n^2)$
- Space Complexity: $O(1)$
- Stability: Yes
- Visualization: See results in Figure 8.



Figure 8: Insertion Sort Result

- **Total number of executed instructions:** 2,706,132 instructions.
- **Instruction breakdown:**
 - **R-type:** 678,804 instructions (25%) – representing computation instructions like add and sub.
 - **R4-type:** 0 instructions (0%).
 - **I-type:** 677,678 instructions (25%) – primarily load and immediate instructions such as lw and addi.
 - **S-type:** 337,131 instructions (12%) – store instructions.
 - **B-type:** 675,388 instructions (24%) – branch instructions like beq and bne.
 - **U-type:** 3 instructions (0%).
 - **J-type:** 337,128 instructions (12%) – jump instructions for controlling program flow.

- **Observations:** The execution reflects a balanced distribution between **R-type**, **I-type**, and **B-type** instructions, each accounting for approximately a quarter of the total instructions. The presence of branch and jump instructions indicates significant looping and control flow operations, consistent with sorting algorithms.

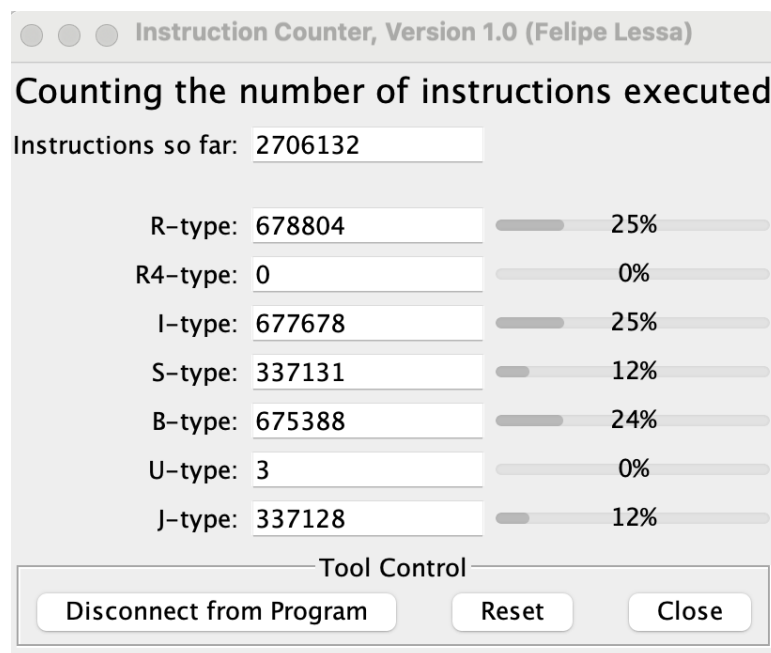


Figure 9: Instruction Count Analysis for Insertion Sort

Selection Sort

- Execution Time: 18110
- Sorting Type: Simple selection-based approach
- Consistent $O(n^2)$ performance
- Minimal number of swaps compared to Bubble Sort
- Visualization: See results in Figure 10.

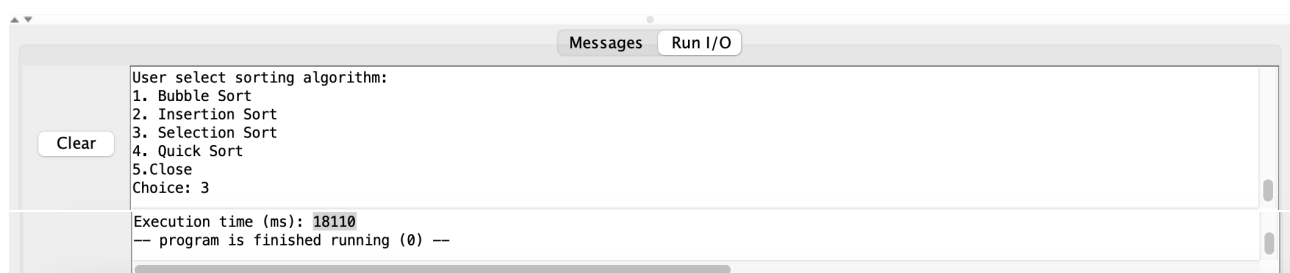


Figure 10: Selection Sort Result

- **Total number of executed instructions:** 3,994,477 instructions.
- **Instruction breakdown:**
 - **R-type:** 1,598,444 instructions (40%) – representing computation instructions like add and sub.
 - **R4-type:** 0 instructions (0%).
 - **I-type:** 1,197,911 instructions (29%) – primarily load and immediate instructions such as lw and addi.

- **S-type**: 425 instructions (0%) – store instructions.
 - **B-type**: 798,533 instructions (19%) – branch instructions like beq and bne.
 - **U-type**: 3 instructions (0%).
 - **J-type**: 399,161 instructions (9%) – jump instructions for controlling program flow.
- **Observations**: The execution shows that **R-type** and **I-type** instructions dominate, together accounting for nearly 70% of the total instructions. **B-type** instructions (19%) highlight frequent conditional branches, while **J-type** instructions (9%) indicate program flow control. The minimal presence of **S-type** and **U-type** reflects limited store operations and unused upper immediate instructions.

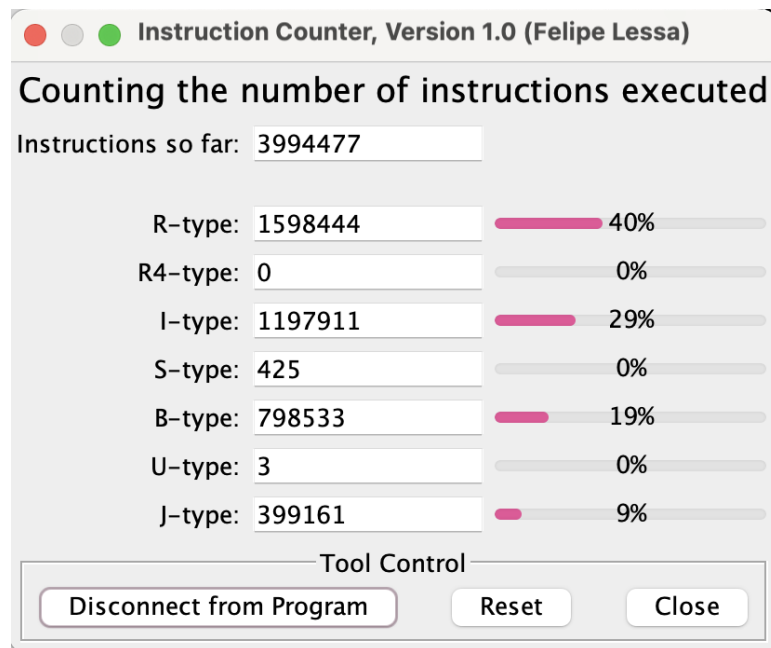


Figure 11: Instruction Count Analysis for Selection Sort

Quick Sort

- Execution Time: 218
- Sorting Type: Divide-and-conquer algorithm
- Most efficient for large datasets
- Average time complexity of $O(n \log n)$
- Space Complexity: $O(\log n)$
- Stability: No
- Visualization:



Figure 12: Quick Sort Result

- **Total number of executed instructions**: 434,841 instructions.

- **Instruction breakdown:**
 - **R-type:** 145,131 instructions (33%) – representing computation instructions like add and sub.
 - **R4-type:** 0 instructions (0%).
 - **I-type:** 143,536 instructions (33%) – primarily load and immediate instructions such as lw and addi.
 - **S-type:** 53,023 instructions (12%) – store instructions.
 - **B-type:** 60,072 instructions (13%) – branch instructions like beq and bne.
 - **U-type:** 1,037 instructions (0%) – upper immediate operations.
 - **J-type:** 32,042 instructions (7%) – jump instructions for controlling program flow.
- **Observations:** The execution shows that **R-type** and **I-type** dominate, each contributing 33% of the total instructions. This reflects the frequent arithmetic computations and memory accesses required by the Quick Sort algorithm. **B-type** instructions (13%) highlight conditional branches for partitioning and recursive calls, while **J-type** instructions (7%) reflect program flow control in recursion. The relatively smaller percentage of **S-type** instructions indicates moderate memory store operations.

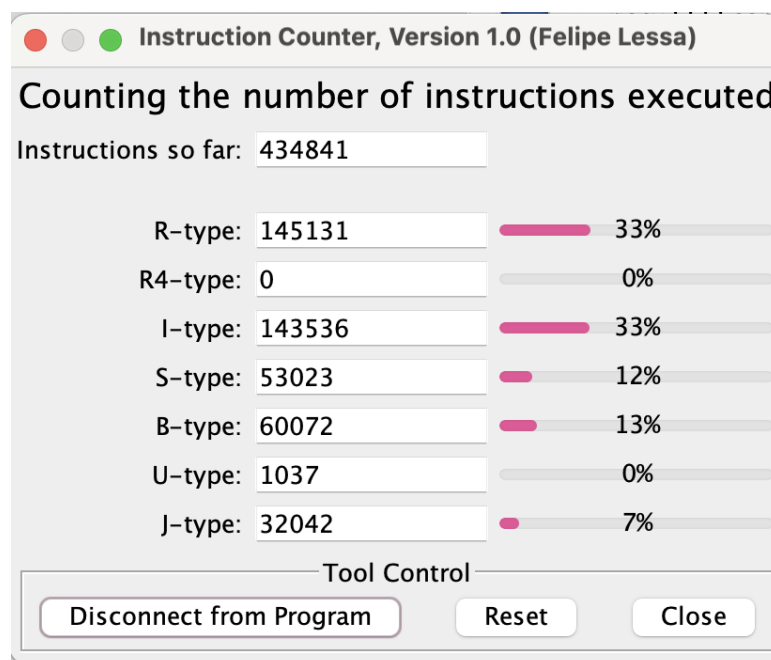


Figure 13: Instruction Count Analysis for Quick Sort

Performance Comparison Table

Algorithm	Execution Time (ms)	Time Complexity	Space Complexity	Stability
Bubble Sort	[15385]	$O(n^2)$	$O(1)$	Yes
Insertion Sort	[5504]	$O(n^2)$	$O(1)$	Yes
Selection Sort	[18110]	$O(n^2)$	$O(1)$	No
Quick Sort	[359]	$O(n \log n)$	$O(\log n)$	No

Table 1: Sorting Algorithm Performance Comparison

Key Observations

- **Execution Time Variation:**

- Quick Sort demonstrated the most consistent and efficient performance.
- Bubble Sort showed the longest execution times.
- Insertion Sort performed moderately well, especially with partially sorted data.

- **Algorithm Efficiency:**

- For small datasets: Insertion Sort shows comparable performance.
- For large datasets: Quick Sort significantly outperforms other algorithms.

Sample Output Demonstration

```
-9996 -9983 -9978 -9951 -9951 -9949 -9927 -9902 -9900 -9893 -9889 -9878 -9860 -9856 -9852 -9833 -9806 -9804 -9803 -9797 -9787 -9785 -9750 -9746 -9742 -9714 -9670 -9658 -9657 -9646 -9642 -9633 -9633 -9632 -9632 -9624
-9596 -9591 -9587 -9582 -9576 -9572 -9572 -9564 -9512 -9510 -9486 -9477 -9475 -9443 -9436 -9435 -9432 -9413 -9410 -9400 -9386 -9363 -9341 -9337 -9325 -9318 -9309 -9307 -9301 -9295 -9289 -9289 -9276 -9261 -9255 -9252
-9230 -9213 -9207 -9195 -9187 -9181 -9175 -9174 -9156 -9137 -9128 -9114 -9113 -9106 -9106 -9096 -9094 -9085 -9079 -9073 -9052 -9046 -9041 -9039 -9027 -9014 -8997 -8993 -8993 -8961 -8922 -8916 -8906 -8888 -8886 -8883
-8866 -8821 -8818 -8815 -8812 -8796 -8782 -8762 -8736 -8718 -8711 -8701 -8692 -8686 -8672 -8669 -8648 -8641 -8626 -8624 -8607 -8606 -8603 -8561 -8555 -8551 -8546 -8528 -8528 -8511 -8504 -8489 -8472 -8460 -8459 -8459
-8450 -8441 -8436 -8422 -8387 -8386 -8361 -8370 -8359 -8358 -8357 -8352 -8298 -8297 -8296 -8286 -8286 -8266 -8257 -8253 -8268 -8258 -8250 -8245 -8244 -8267 -8267 -8205 -8197 -8158 -8128 -8127 -8116 -8115 -8102 -8093 -8079 -8077
-8071 -8071 -8068 -8064 -8044 -8035 -8004 -7993 -7990 -7983 -7979 -7978 -7977 -7973 -7959 -7945 -7930 -7929 -7923 -7909 -7908 -7906 -7902 -7896 -7860 -7849 -7827 -7782 -7781 -7758 -7747 -7742 -7734 -7734 -7733 -7722
-7715 -7707 -7701 -7693 -7680 -7678 -7676 -7659 -7659 -7656 -7643 -7637 -7628 -7610 -7602 -7583 -7568 -7564 -7525 -7518 -7512 -7476 -7468 -7466 -7453 -7433 -7418 -7414 -7412 -7406 -7399 -7398 -7383 -7379 -7378 -7372
-7338 -7332 -7327 -7324 -7320 -7315 -7298 -7293 -7285 -7279 -7269 -7268 -7268 -7248 -7242 -7240 -7235 -7231 -7230 -7216 -7209 -7205 -7203 -7198 -7188 -7184 -7184 -7177 -7169 -7166 -7166 -7165 -7158 -7145 -7141 -7134
-7132 -7101 -7100 -7090 -7085 -7066 -7044 -7037 -7033 -7014 -7014 -6999 -6993 -6988 -6968 -6953 -6940 -6935 -6900 -6869 -6847 -6843 -6842 -6833 -6825 -6818 -6817 -6795 -6788 -6743 -6731 -6728 -6716 -6713 -6669 -6661
-6658 -6651 -6647 -6637 -6627 -6625 -6622 -6592 -6589 -6586 -6565 -6555 -6546 -6546 -6545 -6543 -6516 -6503 -6499 -6465 -6454 -6452 -6448 -6447 -6440 -6422 -6394 -6381 -6381 -6372 -6355 -6342 -6337 -6320 -6310 -6310
-6298 -6278 -6254 -6235 -6233 -6228 -6226 -6215 -6186 -6158 -6143 -6143 -6103 -6089 -6089 -6089 -6065 -6065 -6060 -6058 -6054 -6045 -6041 -6026 -6003 -5984 -5979 -5977 -5974 -5966 -5964 -5963 -5955 -5952 -5938 -5932 -5917
-5914 -5911 -5909 -5908 -5894 -5880 -5861 -5858 -5847 -5841 -5838 -5820 -5819 -5812 -5797 -5792 -5783 -5781 -5781 -5766 -5746 -5718 -5717 -5714 -5709 -5708 -5690 -5685 -5676 -5655 -5653 -5647 -5638 -5622 -5613 -5604
-5590 -5586 -5579 -5577 -5569 -5564 -5563 -5552 -5545 -5545 -5509 -5494 -5474 -5471 -5463 -5462 -5455 -5444 -5428 -5425 -5408 -5408 -5376 -5370 -5365 -5363 -5359 -5358 -5355 -5355 -5338 -5327 -5326 -5324 -5311 -5292
-5288 -5269 -5262 -5259 -5258 -5252 -5244 -5218 -5213 -5212 -5205 -5195 -5177 -5171 -5164 -5162 -5144 -5130 -5130 -5129 -5126 -5125 -5124 -5116 -5115 -5102 -5097 -5096 -5090 -5071 -5044 -5043 -5043 -5042 -5042 -5041
-5036 -5026 -5025 -5024 -5008 -5007 -4993 -4989 -4979 -4978 -4957 -4936 -4917 -4915 -4914 -4911 -4908 -4898 -4886 -4882 -4867 -4861 -4859 -4851 -4849 -4804 -4799 -4798 -4792 -4784 -4783 -4780 -4770 -4759 -4756 -4736
-4733 -4729 -4723 -4712 -4711 -4705 -4702 -4701 -4695 -4690 -4651 -4630 -4618 -4614 -4606 -4605 -4604 -4597 -4596 -4577 -4552 -4548 -4547 -4539 -4526 -4507 -4503 -4478 -4469 -4464 -4459 -4451 -4449 -4414 -4406 -4391
-4388 -4383 -4381 -4319 -4385 -4249 -4236 -4230 -4229 -4224 -4224 -4208 -4204 -4200 -4195 -4187 -4183 -4172 -4167 -4159 -4138 -4135 -4124 -4121 -4113 -4112 -4111 -4108 -4089 -4085 -4083 -4070 -4065 -4061 -4053 -4024
-4005 -4005 -3996 -3981 -3963 -3955 -3908 -3894 -3871 -3864 -3834 -3832 -3823 -3819 -3809 -3807 -3795 -3784 -3780 -3776 -3773 -3763 -3750 -3746 -3743 -3741 -3673 -3657 -3650 -3646 -3639 -3625 -3616 -3578 -3565
-3561 -3557 -3555 -3549 -3532 -3521 -3516 -3512 -3507 -3501 -3497 -3496 -3488 -3486 -3481 -3473 -3454 -3438 -3400 -3390 -3387 -3364 -3361 -3357 -3357 -3354 -3348 -3334 -3330 -3329 -3325 -3321 -3318 -3311 -3309 -3298 -3296
-3269 -3259 -3254 -3250 -3247 -3214 -3204 -3196 -3194 -3190 -3187 -3187 -3167 -3163 -3154 -3143 -3121 -3113 -3087 -3087 -3069 -3069 -3056 -3054 -3050 -3041 -3002 -2999 -2975 -2967 -2951 -2941 -2938 -2909 -2908 -2907
-2885 -2874 -2859 -2853 -2848 -2843 -2819 -2808 -2807 -2807 -2794 -2791 -2788 -2787 -2782 -2777 -2772 -2735 -2731 -2729 -2722 -2719 -2706 -2704 -2688 -2666 -2658 -2641 -2618 -2617 -2616 -2606 -2602 -2601 -2597 -2585
-2572 -2565 -2552 -2543 -2538 -2526 -2518 -2515 -2498 -2473 -2464 -2429 -2424 -2413 -2406 -2384 -2371 -2368 -2367 -2333 -2299 -2288 -2284 -2278 -2274 -2272 -2271 -2268 -2252 -2243 -2238 -2222 -2222 -2213 -2212
-2205 -2192 -2173 -2170 -2156 -2154 -2143 -2139 -2132 -2124 -2112 -2107 -2078 -2066 -2064 -2057 -2043 -2037 -2031 -2025 -2023 -2021 -2020 -2016 -2008 -1998 -1978 -1974 -1938 -1917 -1915 -1901 -1898 -1883 -1882 -1859
-1840 -1821 -1808 -1798 -1792 -1780 -1779 -1771 -1770 -1766 -1759 -1754 -1746 -1736 -1726 -1712 -1709 -1688 -1680 -1645 -1644 -1643 -1642 -1628 -1627 -1623 -1621 -1618 -1611 -1610 -1574 -1569 -1568 -1563 -1546 -1548
-1538 -1536 -1529 -1523 -1521 -1510 -1507 -1505 -1497 -1484 -1471 -1465 -1424 -1423 -1420 -1404 -1393 -1379 -1377 -1358 -1343 -1333 -1294 -1245 -1231 -1215 -1213 -1195 -1188 -1185 -1174 -1143 -1142 -1127 -1127 -1125
-1120 -1089 -1087 -1085 -1084 -1081 -1073 -1067 -1060 -1060 -1059 -1055 -1042 -1038 -1026 -1011 -994 -990 -989 -988 -985 -980 -974 -974 -958 -956 -950 -942 -939 -935 -923 -922 -895 -891 -875 -856 -855 -848 -845
-815 -818 -791 -755 -747 -727 -707 -697 -697 -688 -688 -673 -664 -661 -657 -626 -612 -596 -555 -545 -537 -530 -527 -516 -514 -503 -498 -485 -474 -469 -465 -460 -456 -449 -445 -444 -436 -435 -430 -421 -391 -388 -386
-384 -376 -373 -363 -358 -355 -351 -345 -340 -337 -331 -321 -301 -298 -280 -276 -276 -274 -237 -230 -220 -204 -190 -190 -175 -143 -137 -134 -117 -101 -94 -97 -87 -82 -59 -59 -50 -41 -30 -23 -6 -3 3 10 30 39 53 65 85
115 130 136 142 183 187 200 200 210 210 243 245 248 261 268 293 294 297 324 332 338 343 347 349 350 373 424 437 444 458 465 479 482 487 536 550 553 557 565 599 608 610 627 635 669 680 710 713 714 718 731 734 742 751
753 757 761 777 782 802 803 810 811 821 823 824 833 835 837 847 860 922 937 960 975 981 989 989 1001 1005 1016 1042 1046 1049 1058 1063 1067 1074 1082 1082 1095 1124 1127 1141 1165 1166 1169 1190 1194 1203 1220 1224
1230 1234 1236 1243 1246 1275 1295 1297 1298 1307 1315 1324 1326 1326 1337 1342 1369 1386 1398 1401 1410 1411 1417 1418 1419 1424 1435 1457 1457 1482 1486 1486 1496 1517 1524 1525 1528 1543 1558 1552 1556 1581 1593
1595 1613 1620 1633 1635 1645 1645 1655 1673 1681 1683 1719 1726 1728 1731 1753 1757 1757 1760 1778 1781 1783 1785 1785 1804 1809 1813 1822 1823 1842 1849 1850 1853 1856 1867 1874 1882 1908 1921 1927 1941 1942
1960 1981 1996 1997 2011 2013 2018 2052 2059 2078 2089 2115 2123 2126 2127 2136 2139 2153 2164 2178 2185 2186 2189 2198 2198 2200 2224 2233 2246 2262 2263 2270 2280 2298 2308 2321 2341 2355 2357 2361 2365 2365 2366
2377 2379 2383 2399 2400 2415 2419 2438 2433 2434 2437 2443 2450 2454 2462 2466 2479 2479 2488 2502 2502 2520 2521 2522 2558 2553 2599 2603 2603 2605 2611 2611 2615 2631 2632 2635 2648 2663 2664 2669 2671 2673 2673
2686 2706 2726 2727 2729 2757 2767 2807 2822 2825 2825 2889 2923 2930 2932 2933 2948 2945 2945 2954 2974 2975 2977 2981 2987 2998 3006 3007 3012 3012 3015 3017 3017 3031 3057 3059 3073 3086 3089 3091 3096 3098 3100
3125 3141 3144 3169 3186 3197 3216 3216 3222 3223 3231 3240 3252 3269 3279 3304 3313 3321 3337 3341 3347 3350 3364 3367 3374 3388 3399 3399 3402 3418 3418 3435 3449 3453 3469 3460 3464 3483 3487 3489 3523 3525
3531 3534 3537 3569 3593 3596 3596 3597 3641 3648 3653 3676 3694 3700 3730 3731 3764 3766 3776 3786 3792 3798 3812 3822 3827 3838 3839 3841 3845 3860 3868 3869 3889 3896 3906 3924 3935 3939 3939 3968 3969 4002 4023
4035 4040 4068 4093 4112 4105 4106 4106 4113 4119 4125 4139 4159 4166 4182 4185 4194 4194 4204 4207 4211 4213 4218 4224 4241 4248 4252 4270 4275 4275 4312 4312 4314 4322 4346 4357 4371 4398 4404 4414 4417 4432 4432
4448 4452 4453 4489 4496 4529 4550 4552 4552 4555 4554 4567 4573 4581 4589 4593 4608 4620 4621 4635 4646 4668 4688 4686 4701 4707 4712 4720 4728 4732 4733 4739 4747 4757 4831 4832 4833 4850 4875 4891 4894 4913
4919 4922 4934 4936 4939 4940 4968 4984 4989 4990 4997 5007 5028 5031 5048 5052 5063 5074 5105 5108 5115 5139 5141 5154 5174 5199 5199 5203 5204 5206 5218 5227 5234 5239 5240 5246 5246 5249 5268 5262 5265 5267 5286
5287 5298 5294 5314 5314 5338 5334 5351 5352 5376 5378 5392 5404 5442 5473 5491 5495 5501 5508 5588 5533 5543 5575 5583 5592 5592 5593 5596 5615 5621 5622 5628 5635 5643 5646 5665 5665 5674 5688 5690 5696 5697 5698
5720 5730 5731 5736 5760 5763 5763 5764 5767 5786 5791 5815 5823 5838 5849 5850 5889 5895 5920 5925 5949 5959 5975 5988 5992 5995 6017 6028 6037 6050 6060 6066 6082 6111 6134 6136 6149 6155 6168 6168 6196 6212 6212
6225 6233 6240 6259 6267 6279 6321 6322 6326 6346 6352 6353 6358 6371 6377 6391 6392 6392 6396 6416 6428 6434 6436 6442 6447 6448 6466 6474 6478 6479 6492 6511 6528 6553 6576 6593 6604 6605 6607 6616 6628 6643 6648
6649 6656 6660 6672 6678 6720 6722 6724 6736 6743 6751 6755 6766 6816 6817 6820 6839 6845 6846 6878 6885 6909 6912 6935 6939 6942 6956 6961 6976 6990 6994 7027 7043 7058 7062 7085 7090 7092 7105 7111 7114 7131 7133
7139 7145 7149 7151 7160 7161 7170 7172 7193 7203 7211 7216 7234 7240 7242 7247 7258 7260 7275 7285 7288 7325 7329 7335 7336 7340 7341 7349 7365 7398 7429 7448 7458 7471 7495 7504 7512 7533 7549 7560 7561 7611 7613
7625 7652 7655 7661 7665 7683 7683 7744 7751 7761 7761 7763 7772 7783 7794 7806 7821 7822 7837 7837 7838 7842 7855 7868 7882 7882 7896 7899 7921 7925 7934 7936 7948 7949 7961 7969 7971 7974 7977 8008 8019 8023 8025 8044
8044 8052 8061 8064 8068 8075 8078 8081 8093 8097 8132 8139 8146 8152 8173 8185 8189 8200 8201 8232 8239 8262 8263 8288 8292 8329 8331 8342 8358 8382 8409 8414 8435 8435 8441 8458 8483 8488 8504 8515 8525 8528 8541
8547 8555 8557 8582 8588 8600 8613 8613 8633 8644 8657 8673 8678 8682 8698 8723 8729 8733 8752 8770 8775 8796 8796 8831 8836 8838 8841 8855 8863 8867 8868 8869 8875 8894 8896 8896 8901 8934 8941 8962 8963 8976 8976
9019 9035 9042 9052 9059 9063 9079 9087 9090 9093 9094 9133 9142 917 9150 918 912 9174 917 9181 9188 9196 9206 9208 9216 9226 9243 9256 9257 9285 9286 9293 9299 9305 9312 9318 9319 9322 9328 9328 9339 9349
9359 9362 9421 9436 9443 9445 9446 9454 9454 9456 9506 9571 9575 9577 9577 9621 9627 9653 9638 9686 9701 9707 9724 9727 9735 9743 9769 9783 9813 9812 9833 9848 9872 9879 9881 9889 9898
9911 9931 9940 9979 9988 9994
```

Figure 14: Output Demonstration

Error Handling and Robustness

- The program successfully handles various input scenarios.
- Supports both positive and negative integers.
- Provides clear error messages for file operations.

Recommendations

- For large datasets: Prefer Quick Sort.
- For small or nearly sorted datasets: Consider Insertion Sort.
- Avoid Bubble Sort for performance-critical applications.

Limitations

- Current implementation uses a fixed pivot selection strategy in Quick Sort.
- Memory usage is fixed and pre-allocated.
- No dynamic memory allocation implemented.

REFERENCES
