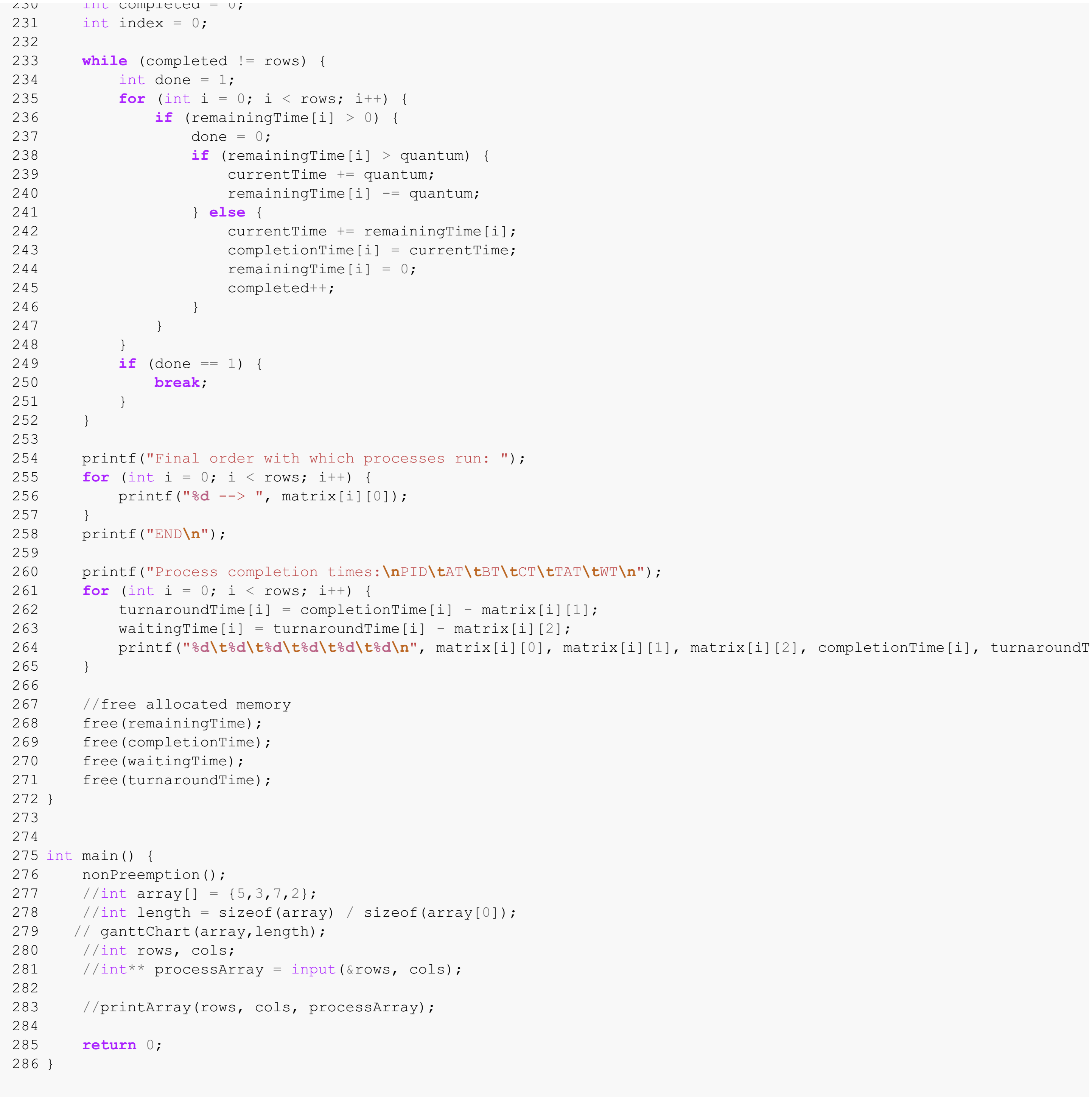
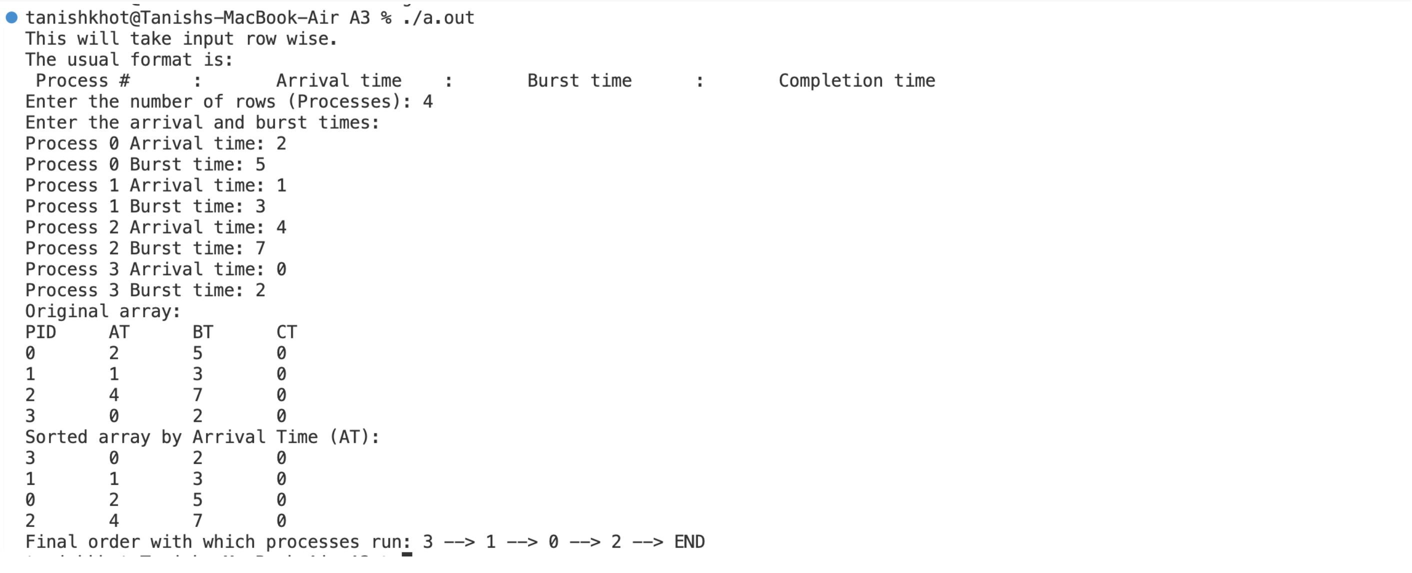
```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

//working
void printArray(int rows, int cols, int** array) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d\t", array[i][j]);
        }
        printf("\n");
    }
}

//working
int** input(int* rows, int cols) {
    cols = 4;
    printf("This will take input row wise.\n");

    printf("The usual format is: \n Process #\t:\tArrival time\t:\tBurst time\t:\tCompletion time\n");
    printf("Enter the number of rows (Processes): ");
    scanf("%d", rows);
    //Dynamic memory allocation is when the data structure is changed at runtime.
    int** array = (int**)malloc(*rows * sizeof(int*));  //refer to malloc.c
    for (int i = 0; i < *rows; i++) {
        array[i] = (int*)malloc(cols * sizeof(int));
    }

    printf("Enter the arrival and burst times:\n");
    for (int i = 0; i < *rows; i++) {
        array[i][0] = i;

        printf("Process %d Arrival time: ", i);
        scanf("%d", &array[i][1]);

        printf("Process %d Burst time: ", i);
        scanf("%d", &array[i][2]);

        array[i][3] = 0; //initialize completion time to 0
    }

    printf("Original array: \nPID\tAT\tBT\tCT\n");

    printArray(*rows, cols, array);

    return array;
}

// Not workig: row int ptr issue
void sortRowsBySecondColumn(int** array, int rows) {
    for (int i = 0; i < rows - 1; i++) {
        for (int j = 0; j < rows - i - 1; j++) {
            if (array[j][1] > array[j + 1][1]){
                for (int k = 0; k < 4; k++) {
                    int temp = array[j][k];
                    array[j][k] = array[j + 1][k];
                    array[j + 1][k] = temp;
                }
            }
        }
    }
}

//working : manage how to pass an array here
void ganttChart(int* arr, int length){
    //top:
    for(int i = 0; i < length; i++){
        for(int j = 0; j < arr[i]; j++){
            printf("__");
        }
    }
    printf("\n");

    //inside block:
    for(int i = 0; i < length; i++){
        printf("|");
        for(int j = 0; j < arr[i]; j++){
            if(j == arr[i]/2){
                printf("%d ", arr[i]);
                if(arr[i] >= 10){   //for two digit numbers
                    j++;
                }
            }
            else{
                printf("  ");
            }
        }
    }
    printf("|\n");

    //bottom:
    for(int i = 0; i < length; i++){
        printf("|");
        for(int j = 0; j < arr[i]; j++){
            printf("__");
        }
    }
    printf("|\n");
}

//Not working, chnge it from 2D array to a ptr
void nonPreemption(){
    // Logic : sort the array according to arrival time. Then, directly display the order in which they are.
    int rows;
    int cols = 4;
    int** matrix = input(&rows, cols);

    sortRowsBySecondColumn(matrix, rows);

    //printing the sorted array
    printf("Sorted array by Arrival Time (AT):\n");
    printArray(rows, cols, matrix);

    int currentTime = 0;
    for(int i = 0; i < rows; i++){
        if(currentTime < matrix[i][1]){
            currentTime = matrix[i][1];
        }
        currentTime += matrix[i][2];
        matrix[i][3] = currentTime; // update completion time
    }

    printf("Final order with which processes run: ");
    for(int i = 0; i < rows; i++) {
        printf("%d --> ", matrix[i][0]);
    }
    printf("END\n");

    // Free allocated memory
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

void preemptionSRTF() {
    int rows;
    int cols = 4;
    int** matrix = input(&rows, cols);

    int* remainingTime = (int*)malloc(rows * sizeof(int));
    int* completionTime = (int*)malloc(rows * sizeof(int));
    int* startTime = (int*)malloc(rows * sizeof(int));
    int* waitingTime = (int*)malloc(rows * sizeof(int));

    for (int i = 0; i < rows; i++) {
        remainingTime[i] = matrix[i][2]; // Initialize remaining time with burst time
        completionTime[i] = 0;
        startTime[i] = 0;
        waitingTime[i] = 0;
    }

    int completed = 0, currentTime = 0, minBurstTime = INT_MAX;
    int completed = 0, currentTime = 0, minBurstTime = INT_MAX;
    int shortest = 0, finishTime;
    int check = 0;

    while (completed != rows) {
        for (int j = 0; j < rows; j++) {
            if ((matrix[j][1] <= currentTime) && (remainingTime[j] < minBurstTime) && remainingTime[j] > 0) {
                minBurstTime = remainingTime[j];
                shortest = j;
                check = 1;
            }
        }

        if (check == 0) {
            currentTime++;
            continue;
        }

        remainingTime[shortest]--;

        minBurstTime = remainingTime[shortest];
        if (minBurstTime == 0) {
            minBurstTime = INT_MAX;
        }

        if (remainingTime[shortest] == 0) {
            completed++;
            check = 0;

            finishTime = currentTime + 1;
            completionTime[shortest] = finishTime;
            startTime[shortest] = finishTime - matrix[shortest][2] - matrix[shortest][1];
            if (startTime[shortest] < 0) {
                startTime[shortest] = 0;
            }
        }
        currentTime++;
    }

    printf("Final order with which processes run: ");
    for (int i = 0; i < rows; i++) {
        printf("%d --> ", matrix[i][0]);
    }
    printf("END\n");

    printf("Process completion times:\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < rows; i++) {
        int tat = completionTime[i] - matrix[i][1];
        int wt = tat - matrix[i][2];
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", matrix[i][0], matrix[i][1], matrix[i][2], completionTime[i], tat, wt);
    }

    //free allocated memory
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);
    free(remainingTime);
    free(completionTime);
    free(startTime);
    free(waitingTime);
}

void roundRobin(int** matrix, int rows, int quantum) {
    int* remainingTime = (int*)malloc(rows * sizeof(int));
    int* completionTime = (int*)malloc(rows * sizeof(int));
    int* waitingTime = (int*)malloc(rows * sizeof(int));
    int* turnaroundTime = (int*)malloc(rows * sizeof(int));

    for (int i = 0; i < rows; i++) {
        remainingTime[i] = matrix[i][2]; // Initialize remaining time with burst time
        completionTime[i] = 0;
        waitingTime[i] = 0;
        turnaroundTime[i] = 0;
    }

    int currentTime = 0;
    int completed = 0;
```



```c
    int completed = 0;
    int index = 0;

    while (completed != rows) {
        int done = 1;
        for (int i = 0; i < rows; i++) {
            if (remainingTime[i] > 0) {
                done = 0;
                if (remainingTime[i] > quantum) {
                    currentTime += quantum;
                    remainingTime[i] -= quantum;
                } else {
                    currentTime += remainingTime[i];
                    completionTime[i] = currentTime;
                    remainingTime[i] = 0;
                    completed++;
                }
            }
        }
        if (done == 1) {
            break;
        }
    }

    printf("Final order with which processes run: ");
    for (int i = 0; i < rows; i++) {
        printf("%d --> ", matrix[i][0]);
    }
    printf("END\n");

    printf("Process completion times:\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < rows; i++) {
        turnaroundTime[i] = completionTime[i] - matrix[i][1];
        waitingTime[i] = turnaroundTime[i] - matrix[i][2];
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", matrix[i][0], matrix[i][1], matrix[i][2], completionTime[i], turnaroundT
    }

    //free allocated memory
    free(remainingTime);
    free(completionTime);
    free(waitingTime);
    free(turnaroundTime);
}


int main() {
    nonPreemption();
    //int array[] = {5,3,7,2};
    //int length = sizeof(array) / sizeof(array[0]);
   // ganttChart(array,length);
    //int rows, cols;
    //int** processArray = input(&rows, cols);

    //printArray(rows, cols, processArray);

    return 0;
}
```



```
tanishkhot@Tanishs-MacBook-Air A3 % ./a.out
This will take input row wise.
The usual format is:
 Process #      :       Arrival time    :       Burst time      :       Completion time
Enter the number of rows (Processes): 4
Enter the arrival and burst times:
Process 0 Arrival time: 2
Process 0 Burst time: 5
Process 1 Arrival time: 1
Process 1 Burst time: 3
Process 2 Arrival time: 4
Process 2 Burst time: 7
Process 3 Arrival time: 0
Process 3 Burst time: 2
Original array:
PID    AT     BT     CT
0      2      5      0
1      1      3      0
2      4      7      0
3      0      2      0
Sorted array by Arrival Time (AT):
3      0      2      0
1      1      3      0
0      2      5      0
2      4      7      0
Final order with which processes run: 3 --> 1 --> 0 --> 2 --> END
```