PROGRAM CODE

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

struct process {
	int num;
	float burst;
	float arrival;
	int priority;
	float remaining;
	float waiting;
	float turnaround;
};

void sort_arrival(struct process *proc, int n){
	int i, j;
	struct process temp;

	for(i=1; i<n; i++){
		temp = proc[i];
		for(j=i-1; j>=0; j--){
			if(temp.arrival < proc[j].arrival)
				proc[j+1] = proc[j];
			else {
				break;
			}
		}
		proc[j+1] = temp;
	}
}

void sort_num(struct process *proc, int n){
	int i, j;
	struct process temp;

	for(i=1; i<n; i++){
		temp = proc[i];
		for(j=i-1; j>=0; j--){
			if(temp.num < proc[j].num)
				proc[j+1] = proc[j];
			else {
				break;
			}
		}
		proc[j+1] = temp;
	}
}

void fcfs(struct process *proc, int n){
	int i, j;

	sort_num(proc, n); //For when processes with the same arrival time are sorted somehow in another algorithm
	sort_arrival(proc, n);

	printf("\nFCFS\n\nThe sorted process chart is given below\n");
	printf("\nProcess\t\tBurst Time (ms)\t\tArrival Time (ms)\n");
	for(i=0; i<n; i++) {
		printf("P%d\t\t%.3f\t\t\t%.3f\n", proc[i].num, proc[i].burst, proc[i].arrival);
	}

	printf("\nThe Gantt chart is given below\n");
```

```
    printf("\n|");
    for(i=0; i<n; i++){
        printf("\tP%d\t|", proc[i].num);
    }

    float total_time = 0, avg_wait = 0, avg_turn = 0;

    //Calculation
    printf("\n%.1f", total_time);
    for(i=0; i<n; i++) {
        if(total_time < proc[i].arrival) { //CPU idle case
            total_time = proc[i].arrival;
            printf("\\%.1f", total_time);
        }
        total_time += proc[i].burst;
        printf("\t\t%.1f", total_time);
        proc[i].turnaround = total_time - proc[i].arrival;
        proc[i].waiting = proc[i].turnaround - proc[i].burst;
    }

    printf("\n\nWaiting Time chart:\n");
    for(i=0; i<n; i++){
        avg_wait += proc[i].waiting;
        printf("P%d : %.2f\n", proc[i].num, proc[i].waiting);
    }
    printf("\nAverage Waiting Time = %.2f ms\n", avg_wait/n);

    printf("\nTurnaround Time chart:\n");
    for(i=0; i<n; i++){
        avg_turn += proc[i].turnaround;
        printf("P%d : %.2f\n", proc[i].num, proc[i].turnaround);
    }
    printf("\nAverage Turnaround Time = %.2f ms\n____________________________________\n", avg_turn/n);
}

void sjf(struct process *proc, int n){
    int i, j;

    //Sorting based on arrival time
    sort_num(proc, n);
    sort_arrival(proc, n);

    //Sorting based on arrival AND burst times
    for(i=0; i<n; i++){
        for(j=i+1; j<n; j++){
            if(proc[i].arrival != proc[j].arrival)
                break;
            else if(proc[i].arrival == proc[j].arrival && proc[i].burst > proc[j].burst){
                struct process temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }

    //For test cases where processes with lower burst times arrive later
    int total = 0;
    for(i=1; i<n; i++){
        if(total < proc[i-1].arrival)
            total = proc[i-1].arrival;
        total += proc[i-1].burst;
        for(j=i+1; j<n; j++){
            if(total >= proc[j].arrival){
                if(proc[j].burst < proc[i].burst){
                    struct process temp = proc[i];
                    proc[i] = proc[j];
                    proc[j] = temp;
                } else if (proc[j].burst == proc[i].burst && proc[i].num > proc[j].num) {
                    struct process temp = proc[i];
                    proc[i] = proc[j];
                    proc[j] = temp;
                }
            } else {
                break;
            }
        }
    }

    printf("\nSJF\n\nThe sorted process chart is given below\n");
    printf("\nProcess\t\tBurst Time (ms)\t\tArrival Time (ms)\n");
    for(i=0; i<n; i++) {
        printf("P%d\t\t%.3f\t\t\t%.3f\n", proc[i].num, proc[i].burst, proc[i].arrival);
    }

    printf("\nThe Gantt chart is given below\n");
    printf("\n|");
    for(i=0; i<n; i++){
        printf("\tP%d\t|", proc[i].num);
    }

    float total_time = 0, avg_wait = 0, avg_turn = 0;

    //Calculation
    printf("\n%.1f", total_time);
    for(i=0; i<n; i++) {
        if(total_time < proc[i].arrival) {
            total_time = proc[i].arrival;
            printf("\\%.1f", total_time);
        }
        total_time += proc[i].burst;
        printf("\t\t%.1f", total_time);
        proc[i].turnaround = total_time - proc[i].arrival;
        proc[i].waiting = proc[i].turnaround - proc[i].burst;
    }

    printf("\n\nWaiting Time chart:\n");
    for(i=0; i<n; i++){
        avg_wait += proc[i].waiting;
        printf("P%d : %.2f\n", proc[i].num, proc[i].waiting);
    }
    printf("\nAverage Waiting Time = %.2f ms\n", avg_wait/n);

    printf("\nTurnaround Time chart:\n");
    for(i=0; i<n; i++){
        avg_turn += proc[i].turnaround;
        printf("P%d : %.2f\n", proc[i].num, proc[i].turnaround);
    }
    printf("\nAverage Turnaround Time = %.2f ms\n___________________________________________\n", avg_turn/n);
}

void non_pre_priority(struct process *proc, int n){
    int i, j;

    sort_num(proc, n);
    printf("\n");
    for(i=0; i<n; i++) {
```

```
    printf("Enter PRIORITY of process %d (1 (highest) to %d (lowest))\n", proc[i].num, n);
    scanf("%d", &proc[i].priority);
}

//Sorting based on arrival time
sort_arrival(proc, n);

//Sorting based on arrival time AND priority
for(i=0; i<n; i++){
    for(j=i+1; j<n; j++){
        if(proc[i].arrival != proc[j].arrival)
            break;
        else if(proc[i].arrival == proc[j].arrival && proc[i].priority > proc[j].priority){
            struct process temp = proc[i];
            proc[i] = proc[j];
            proc[j] = temp;
        }
    }
}

//For test cases where processes with higher priority arrive later
int total = 0;
for(i=1; i<n; i++){
    if(total < proc[i-1].arrival)
        total = proc[i-1].arrival;
    total += proc[i-1].burst;
    for(j=i+1; j<n; j++){
        if(total >= proc[j].arrival){
            if(proc[j].priority < proc[i].priority){
                struct process temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            } else if (proc[j].priority == proc[i].priority && proc[i].num > proc[j].num) {
                struct process temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        } else {
            break;
        }
    }
}

printf("\nPRIORITY SCHEDULING (NON-PREEMPTIVE)\n\nThe sorted  process chart is given below\n");
printf("\nProcess\t\tBurst Time (ms)\t\tArrival Time (ms)\tPriority\n");
for(i=0; i<n; i++) {
    printf("P%d\t\t%.3f\t\t\t%.3f\t\t\t%d\n", proc[i].num, proc[i].burst, proc[i].arrival, proc[i].priority);
}

printf("\nThe Gantt chart is given below\n");
printf("\n|");
for(i=0; i<n; i++){
    printf("\tP%d\t|", proc[i].num);
}

float total_time = 0, avg_wait = 0, avg_turn = 0;

//Calculation
printf("\n%.1f", total_time);
for(i=0; i<n; i++) {
    if(total_time < proc[i].arrival) {
        total_time = proc[i].arrival;
        printf("\\%.1f", total_time);
```

```
		}
		total_time += proc[i].burst;
		printf("\t\t%.1f", total_time);
		proc[i].turnaround = total_time - proc[i].arrival;
		proc[i].waiting = proc[i].turnaround - proc[i].burst;
	}

	printf("\n\nWaiting Time chart:\n");
	for(i=0; i<n; i++){
		avg_wait += proc[i].waiting;
		printf("P%d : %.2f\n", proc[i].num, proc[i].waiting);
	}
	printf("\nAverage Waiting Time = %.2f ms\n", avg_wait/n);

	printf("\nTurnaround Time chart:\n");
	for(i=0; i<n; i++){
		avg_turn += proc[i].turnaround;
		printf("P%d : %.2f\n", proc[i].num, proc[i].turnaround);
	}
	printf("\nAverage Turnaround Time = %.2f ms\n____________________________________\n", avg_turn/n);
}

void rr(struct process *proc, int n){
	int i, j, count = 0;
	float q;

	printf("\nEnter TIME QUANTUM (in ms)\n");
	scanf("%f", &q);

	//Sorting based on arrival time
	sort_num(proc, n);
	sort_arrival(proc, n);

	printf("\nROUND ROBIN\n\nThe sorted process chart is given below\n");
	printf("\nProcess\t\tBurst Time (ms)\t\tArrival Time (ms)\n");
	for(i=0; i<n; i++) {
		printf("P%d\t\t%.3f\t\t\t%.3f\n", proc[i].num, proc[i].burst, proc[i].arrival);
	}

	for(i=0; i<n; i++)
		proc[i].remaining = proc[i].burst;

	//Preemption Gantt chart
	printf("\n|");
	for(i=0; count!=n; i=(i+1)%n){
		if(proc[i].remaining != 0) {
			printf("\tP%d\t|", proc[i].num);
			if(q < proc[i].remaining)
				proc[i].remaining -= q;
			else {
				proc[i].remaining = 0;
				count++;
			}
		}
	}

	for(i=0; i<n; i++)
		proc[i].remaining = proc[i].burst;

	float total_time = 0, avg_wait = 0, avg_turn = 0;

	//Preemption and calculation
	printf("\n%.1f", total_time);
```

```
    for(i=0, count=0; count != n; i=(i+1)%n){
        if(proc[i].remaining != 0){
            if(proc[i].remaining <= q){
                if(total_time < proc[i].arrival) {
                    total_time = proc[i].arrival;
                    printf("\\%.1f", total_time);
                }
                total_time += proc[i].remaining;
                proc[i].turnaround = total_time - proc[i].arrival;
                proc[i].waiting = proc[i].turnaround - proc[i].burst;
                proc[i].remaining = 0;
                count++;
            } else {
                if(total_time < proc[i].arrival) {
                    total_time = proc[i].arrival;
                    printf("\\%.1f", total_time);
                }
                total_time += q;
                proc[i].remaining -= q;
            }
            printf("\t\t%.1f", total_time);
        }
    }

    printf("\n\nWaiting Time chart:\n");
    for(i=0; i<n; i++){
        avg_wait += proc[i].waiting;
        printf("P%d : %.2f\n", proc[i].num, proc[i].waiting);
    }
    printf("\nAverage Waiting Time = %.2f ms\n", avg_wait/n);

    printf("\nTurnaround Time chart:\n");
    for(i=0; i<n; i++){
        avg_turn += proc[i].turnaround;
        printf("P%d : %.2f\n", proc[i].num, proc[i].turnaround);
    }
    printf("\nAverage Turnaround Time = %.2f ms\n____________________________________\n", avg_turn/n);
}

void main() {
    int n, m = 0;
    printf("Enter the number of processes\n");
    scanf("%d", &n);

    struct process proc[n];

    for(int i=0; i<n; i++) {
        printf("\nEnter BURST TIME of process %d (in ms)\n", i+1);
        scanf("%f", &proc[i].burst);
        printf("Enter ARRIVAL TIME of process %d (in ms)\n", i+1);
        scanf("%f", &proc[i].arrival);
        proc[i].num = i+1;
    }

    printf("\nThe process chart is given below\n");
    printf("\nProcess\t\tBurst Time (ms)\t\tArrival Time (ms)\n");
    for(int i=0; i<n; i++) {
        printf("P%d\t\t%.3f\t\t\t%.3f\n", proc[i].num, proc[i].burst, proc[i].arrival);
    }

    while(1) {
        printf("\nChoose the scheduling algorithm\n");
        printf("1. FCFS\n2. SJF\n3. Priority Scheduling\n4. Round Robin\n5. Exit\n");
        scanf("%d", &m);

        switch(m) {
            case 1:
                fcfs(proc, n);
                break;
            case 2:
                sjf(proc, n);
                break;
            case 3:
                non_pre_priority(proc, n);
                break;
            case 4:
                rr(proc, n);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid option!\n");
                break;
        }
    }
}
```

SAMPLE OUTPUT

```
Enter the number of processes
5

Enter BURST TIME of process 1 (in ms)
2
Enter ARRIVAL TIME of process 1 (in ms)
2

Enter BURST TIME of process 2 (in ms)
3
Enter ARRIVAL TIME of process 2 (in ms)
1

Enter BURST TIME of process 3 (in ms)
2
Enter ARRIVAL TIME of process 3 (in ms)
3

Enter BURST TIME of process 4 (in ms)
4
Enter ARRIVAL TIME of process 4 (in ms)
4

Enter BURST TIME of process 5 (in ms)
1
Enter ARRIVAL TIME of process 5 (in ms)
5

The process chart is given below

Process        Burst Time (ms)              Arrival Time (ms)
P1             2.000                2.000
```

```
P2          3.000           1.000
P3          2.000           3.000
P4          4.000           4.000
P5          1.000           5.000

Choose the scheduling algorithm
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. Exit
1

FCFS

The sorted process chart is given below

Process     Burst Time (ms)      Arrival Time (ms)
P2          3.000           1.000
P1          2.000           2.000
P3          2.000           3.000
P4          4.000           4.000
P5          1.000           5.000

The Gantt chart is given below

|     P2     |     P1     |     P3     |     P4     |     P5     |
0.0\1.0     4.0          6.0          8.0          12.0         13.0

Waiting Time chart:
P2 : 0.00
P1 : 2.00
P3 : 3.00
P4 : 4.00
P5 : 7.00

Average Waiting Time = 3.20 ms

Turnaround Time chart:
P2 : 3.00
P1 : 4.00
P3 : 5.00
P4 : 8.00
P5 : 8.00

Average Turnaround Time = 5.60 ms

Choose the scheduling algorithm
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. Exit
2

SJF
```

```
The sorted process chart is given below

Process         Burst Time (ms)                 Arrival Time (ms)
P2              3.000                   1.000
P1              2.000                   2.000
P5              1.000                   5.000
P3              2.000                   3.000
P4              4.000                   4.000

The Gantt chart is given below

|       P2      |       P1      |       P5      |       P3      |       P4      |
0.0\1.0         4.0             6.0             7.0             9.0             13.0

Waiting Time chart:
P2 : 0.00
P1 : 2.00
P5 : 1.00
P3 : 4.00
P4 : 5.00

Average Waiting Time = 2.40 ms

Turnaround Time chart:
P2 : 3.00
P1 : 4.00
P5 : 2.00
P3 : 6.00
P4 : 9.00

Average Turnaround Time = 4.80 ms
____________________________________

Choose the scheduling algorithm
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. Exit
3

Enter PRIORITY of process 1 (1 (highest) to 5 (lowest))
4
Enter PRIORITY of process 2 (1 (highest) to 5 (lowest))
3
Enter PRIORITY of process 3 (1 (highest) to 5 (lowest))
5
Enter PRIORITY of process 4 (1 (highest) to 5 (lowest))
1
Enter PRIORITY of process 5 (1 (highest) to 5 (lowest))
2

PRIORITY SCHEDULING (NON-PREEMPTIVE)

The sorted  process chart is given below
```

```
Process   Burst Time (ms)               Arrival Time (ms)   Priority
P2        3.000          1.000               3
P4        4.000          4.000               1
P5        1.000          5.000               2
P1        2.000          2.000               4
P3        2.000          3.000               5

The Gantt chart is given below

|    P2   |    P4   |    P5   |    P1   |    P3   |
0.0\1.0   4.0       8.0       9.0       11.0      13.0

Waiting Time chart:
P2 : 0.00
P4 : 0.00
P5 : 3.00
P1 : 7.00
P3 : 8.00

Average Waiting Time = 3.60 ms

Turnaround Time chart:
P2 : 3.00
P4 : 4.00
P5 : 4.00
P1 : 9.00
P3 : 10.00

Average Turnaround Time = 6.00 ms
___________________________

Choose the scheduling algorithm
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. Exit
4

Enter TIME QUANTUM (in ms)
2.5

ROUND ROBIN

The sorted process chart is given below

Process   Burst Time (ms)               Arrival Time (ms)
P2        3.000          1.000
P1        2.000          2.000
P3        2.000          3.000
P4        4.000          4.000
P5        1.000          5.000

|    P2   |    P1   |    P3   |    P4   |    P5   |    P2   |    P4
     |
0.0\1.0   3.5       5.5       7.5       10.0      11.0      11.5
     13.0
```

```
Waiting Time chart:
P2 : 7.50
P1 : 1.50
P3 : 2.50
P4 : 5.00
P5 : 5.00

Average Waiting Time = 4.30 ms

Turnaround Time chart:
P2 : 10.50
P1 : 3.50
P3 : 4.50
P4 : 9.00
P5 : 6.00

Average Turnaround Time = 6.70 ms
```

```
Choose the scheduling algorithm
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. Exit
5
```