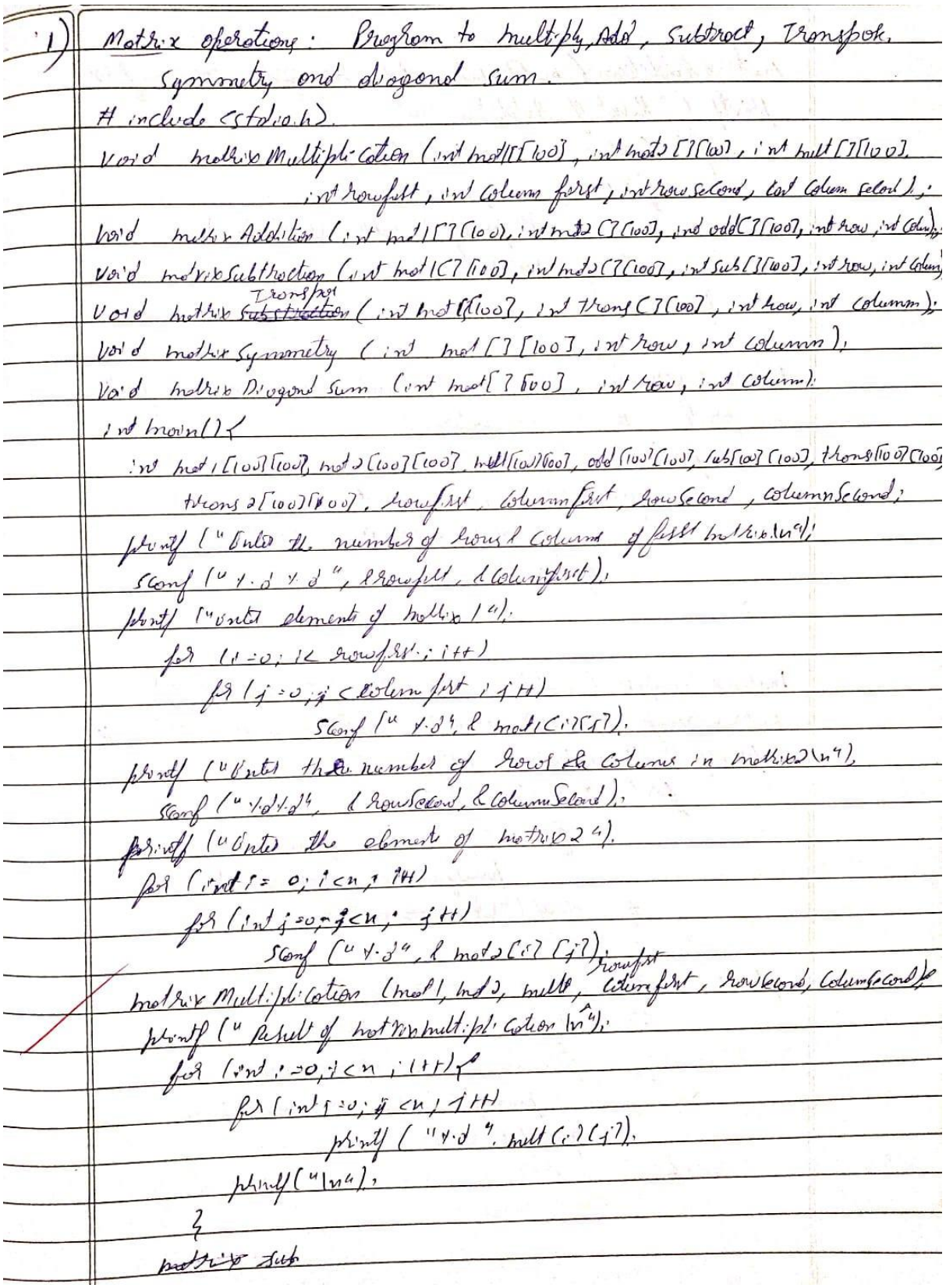
OS Lab

1.Matrix Operations:



1) Matrix operations: Program to multiply, Add, Subtract, Transpose, Symmetry and diagonal sum.

```
#include<stdio.h>
void matrixMultiplication(int mat1[][100], int mat2[][100], int mult[][100],
                          int rowfirst, int columnfirst, int rowSecond, int columnSecond);
void matrixAddition(int mat1[][100], int mat2[][100], int add[][100], int row, int column);
void matrixSubtraction(int mat1[][100], int mat2[][100], int sub[][100], int row, int column);
void matrixTranspose(int mat1[][100], int trans[][100], int row, int column);
void matrixSymmetry(int mat[][100], int row, int column);
void matrixDiagonalSum(int mat[][100], int row, int column);
int main(){
    int mat1[100][100], mat2[100][100], mult[100][100], add[100][100], sub[100][100], trans[100][100],
        trans2[100][100], rowfirst, columnfirst, rowSecond, columnSecond;
    printf("Enter the number of rows & column of first matrix\n");
    scanf("%d %d", &rowfirst, &columnfirst);
    printf("Enter element of matrix 1\n");
    for(i=0; i<rowfirst; i++)
        for(j=0; j<columnfirst; j++)
            scanf("%d", &mat1[i][j]);
    printf("Enter the number of rows & columns in matrix 2\n");
    scanf("%d %d", &rowSecond, &columnSecond);
    printf("Enter the element of matrix 2\n");
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            scanf("%d", &mat2[i][j]);
matrixMultiplication(mat1, mat2, mult, columnfirst, rowSecond, columnSecond);
    printf("Result of matrix multiplication\n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++)
            printf("%d", mult[i][j]);
        printf("\n");
    }
matrix Sub
```

```
matrix Addition (mat1, mat2, add, rowfirst, column first);
printf ("Result of Addition:\n");
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++)
            printf ("%d", add[i][j]);
        printf ("\n");
    }

matrix Subtraction (mat1, mat2, sub, rowfirst, column first);
    printf ("Result of subtraction:\n");
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++)
            printf ("%d", SUB[i][j]);
        printf ("\n");
    }

matrix Transpose (mat1, trans1, row first, column first);
matrix Transpose (mat2, trans2, row Second, column SecondI);
    printf ("Transpose of matrix1:\n")
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++)
                printf ("%d", trans1[i][j]);
            printf ("\n");
        }
    printf ("Transpose of matrix2:\n")
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++)
                printf ("%d", trans2[i,j]);
            printf ("\n");
        }
```

```
void matrixTranspose(int mat[][100], int trans[][100], int row,
    int column) {
    for(int i=0; i<row; i++)
        for(int j=0; j<column; j++)
            trans[j][i] = mat[i][j]
}

void matrixSymmetry(int mat[][100], int row, int column){
    for(int i=0; i<row; i++) {
        for(int j=0; j<column; j++) {
            if( mat[i][j] != mat[j][i]){
                flag=0;
                break;
            }}
        if (flag==0)
            break;
    }
    if (flag==1)
        printf("Symmetric");
    else
        printf("not symmetric");
}

void matrixDiagonalSum(int mat[][100], int row, int column){
    int principalSum=0, nonPrincipalSum=0;
    for(int i=0; i<row; i++)
        for(int j=0; j<column; j++){
            if(i==j)
                principalSum += mat[i][i];
            if(i+j==row-1)
                principalSum += mat[i][j];
        }
    printf("Sum of principal diagonal = %d\n", principalSum);
    printf("\n Sum of nonprincipal
```

output:

Enter no of rows and columns of 1st matrix

3 3

Enter the element of matrix1

1 3 2

45 67 6

8 4 3

Enter no of row & colum of second matrix

3 3

Enter elements of matrix2:

2 12 11

5 6 16

7 4 1

Result of matrix addition:

3 5 13

70 11 22

15 8 4

Result of matrix multiplication:

31 38 61

117 594 791

77 132 155

Matrix 1 is not symmetric

Matrix 2 is not symmetric

Transpose of matrix1:

1 45 8

3 67 4

2 6 3

Sum of principal diagonal is 9

sum of Non principal diagonal is 15.

```
matrix_Symmetry (mat1, rowFirst, columnFirst);
matrix_Symmetry (mat2, rowFirst, columnFirst);

multiDiagonalSum (mat1, rowFirst, columnFirst);
multiDiagonalSum (mat2, rowSecond, columSecond);
{
return 0;
}

void matrixMultiplication (int mat1[3][100], int mat2[3][100], int mult[100],
    int rowFirst, int columnFirst, int rowSecond, int columSecond){
int i, j, k;
  for (i=0; i< rowFirst; ++i){
      for (j=0; j< columnFirst; j++)
          mult[i][j]=0;
  for (i=0; i< rowFirst; ++i) {
      for (j=0; j< columSecond; j++)
        for (k=0; k< columFirst; k++)
            mult[i][j] += mat1[i][k] * mat2[k][j];

}

Void matrixAddition (int mat1[3][100], int mat2[3][100], int add[3][100],
      int row, int column){
  int i, j;
    for (i=0; i<row; i++)
      for (j=0; j< colm; j++)
          add[i][j] = mat1[i][j] + mat2[i][j];
{

Void MatrixSubtraction (int mat1[3][100], int mat2[3][100], int sub[3][100]
      int row, int column){
    for (int i=0; i<row; i++)
        for (int j=0; j< column; j++)
            sub[i][j] = mat1[i][j] - mat2[i][j];
```

Output:

```
Enter the number of rows and columns of first matrix: 3 3
Enter the elements of first matrix:
1 3 2
45 5 6
8 4 3
Enter the number of rows and columns of second matrix: 3 3
Enter the elements of second matrix:
2 12 11
5 6 16
7 4 1
Result of matrix multiplication:
31 38 61
157 594 581
57 132 155
Result of matrix addition:
3 15 13
50 11 22
15 8 4
Result of matrix subtraction:
-1 -9 -9
40 -1 -10
1 0 2
Transpose of first matrix:
1 45 8
3 5 4
2 6 3
Transpose of second matrix:
2 5 7
12 6 4
11 16 1
The matrix is not symmetrical.
The matrix is not symmetrical.
Sum of principal diagonal: 9
Sum of non-principal diagonal: 15
Sum of principal diagonal: 9
Sum of non-principal diagonal: 24
PS D:\Programs\OSlab>
```

2. Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time. i. FCFS ii. SJF (pre-emptive & non-pre-emptive)

Lab1

1. Write a C program to simulate non-preemptive CPU scheduling ~~FCFS~~ algorithm to find turnaround time and waiting time.

a) FCFS

```
#include <stdio.h>

void wait_time (int n, int burst_time[], int waiting_time[]);
void turnaround_time (int n, int burst_time[], int waiting_time[],
        int turnaround_time[]);
void avgtime (int n, int proc[], int burst_time[]);

int main() {
    int proc[] = {1, 2, 3};
    int burst_time[] = {10, 3, 4};
    int n;

    printf ("Enter the number of processes\n");
    scanf ("%d", &n);
    printf ("Enter the processes\n");
    ~~scanf~~
    for (int i = 0; i < n; i++) {
        scanf("%d", &proc[i]);
    printf ("\nEnter the burst times\n");
    for(int i = 0; i<n; i++)
        scanf ("%d", &burst_time[i]);
    avgtime (n, proc, burst_time);
    return 0;
}

void wait_time (int n, int burst_time[], int waiting_time[]
    int sum = 0;
    int i;
    for (i = 0; i<n; i++){
        waiting_time[i] = sum;
        sum += burst_time[i];
}
```

```
        waiting_time[i] = sum;
    }

void Turnaround_time(int n, int burst_time[], int waiting_time[],
    int turnaround_time[]) {
    for (int i=0; i<n; i++) {
        turnaround_time[i] = burst_time[i] + waiting_time[i];
    }
}

Void avg_time (int n, int proc[], int burst_time[]) {
    int waiting_time[n];
    int turnaround_time[n];
    float avg_wait, avg_tat;
    wait_time(n, burst_time, waiting_time);
    Turnaround_time(n, burst_time, waiting_time, turnaround_time
    int sum_wait = 0;
    int sum_tat = 0;
    for (int i=0; i<n; i++) {
        sum_wait += waiting_time[i];
        sum_tat += turnaround_time[i];
    }
    avg_tat = (float) sum_wait/n;
    avg_wait = (float) sum_wait/n;
    printf("Process \t Burst time \t Waiting time \t Turnaround time \n");
    for (int i=0; i<n; i++) {
        printf("%d \t\t %d \t\t %d \t\t %d \n", proc[i], burst_time[i]
        waiting_time[i], turnaround_time[i]);
    printf("\n Average waiting time is: %f", avg_wait);
    printf("\n Average turn-around time is: %f", avg_tat);
}
```

output:

Enter the number of processes
3
Enter the processes
1 2 3
Enter the burst time
4 3 12

Process	Burst time	Waiting time	Turn around time
1	4	0	4
2	3	4	7
3	12	7	19

Average waiting time is : 3.666667
Average turn around time is : 10.00

```
void turnaround_time (int n, int burst_time[], int waiting_time[]
        int turnaround_time[]){
    for (int i=0; i<n; i++){
        turnaround_time[i] = burst_time[i] + waiting_time[i];
}

void sort (int n, int proc[], int burst_time[]){ int temp=0;
    for (int i=0; i<n; i++){
        for (int j=i; j<n; j++){
            if ( burst_time[i] > burst_time[j]){
                temp = burst_time[i];
                burst_time[i] = burst_time[j];
                burst_time[j] = temp;
                temp = proc[j];
                proc[j] = proc[i];
                proc[i] = temp;
            }
        }
    }
}
void avgtime (int n, int proc[], int burst_time[]){
    int waiting_time[n];
    int turnaround_time[n];
    float avg_wait, avg_tat;
    wait_time (n, burst_time, waiting_time);
    turnaround_time (n, burst_time, waiting_time, turnaround_tat
    int sum_wait=0;
    int sum_tat=0;
    for (int i=0; i<n; i++)
        sum_wait += waiting_time[i];
    for (int i=0; i<n; i++)
    sum_tat += turnaround_time[i];
```

```
avg_tat = (float) sum_tat / n;
avg_wat = (float) sum_wait / n;
printf("Process \t Burst time \t Waiting time \t Turnaround time \n");
for (int i=0; i<n; i++) {
        printf("%d \t %d \t %d \t %d \n", proc[i], burst_time[i],
            waiting_time[i], turnaround_time[i]);
printf("average waiting time: %f", avg_wait);
printf("average turn around time: %f", avg_tat);
}
```

output:

```
Enter no of processes
3
Enter the processes
1 2 3
Enter the burst time
3 3 12
```

Process	Burst time	Waiting time	Turn around time
2	3	0	3
1	4	3	7
3	12	7	19

```
Average waiting time : 3.333
Average turn around time : 9.666
```

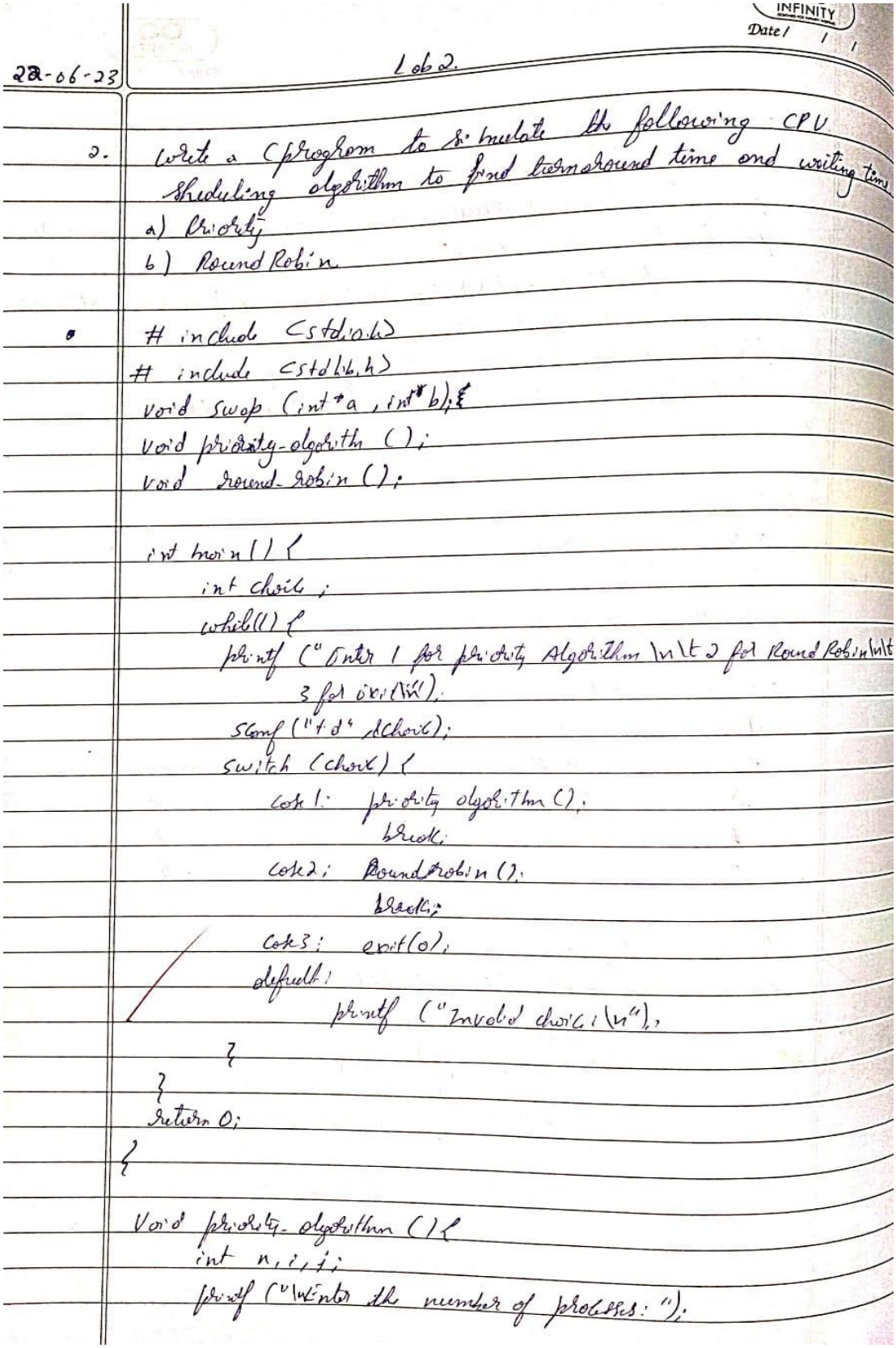
22/6/23

Upload one style pdf (Menu driven)

Output:

```
Menu:
1. First-Come, First-Served (FCFS)
2. Shortest Job First (SJF)
Enter your choice: 1
Enter the number of processes: 3
Enter burst time for each process:
P[1]: 4
P[2]: 3
P[3]: 12

Process Burst Time      Waiting Time    Turnaround Time
P[1]            4               0               4
P[2]            3               4               7
P[3]            12              7               19

Average Waiting Time = 3.666667
Average Turnaround Time = 10.000000
Menu:
1. First-Come, First-Served (FCFS)
2. Shortest Job First (SJF)
Enter your choice: 2
Enter number of processes: 3
Enter burst time for each process:
P[1]: 4
P[2]: 3
P[3]: 12

Process Burst Time      Waiting Time    Turnaround Time
P[2]            3               0               3
P[1]            4               3               7
P[3]            12              7               19

Average Waiting Time = 3.333333
Average Turnaround Time = 9.666667
```

3. Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. A) Priority (pre-emptive & non-pre-emptive) B) Round Robin (Experiment with different quantum sizes for RR algorithm)



22-06-23

Lab 2.

2. Write a C program to simulate the following CPU Scheduling algorithm to find turnaround time and waiting time
a) Priority
b) Round Robin

```
#include<stdio.h>
#include<stdlib.h>
void swap(int *a, int *b);{
void priority_algorithm();
void round_robin();

int main(){
    int choice;
    while(1){
        printf("Enter 1 for priority Algorithm\n\t2 for Round Robin\n\t
            3 for exit\n");
        scanf("%d",&choice);
        switch(choice){
            case 1: priority algorithm();
                    break;
            case 2: RoundRobin();
                    break;
            case 3: exit(0);
            default:
                    printf("Invalid choice\n");
        }
    }
    return 0;
}

Void priority_algorithm(){
    int n,i,j;
    printf("\nEnter the number of processes:");
```

```
scanf("%d", &n);
int burst_time[n], priority[n], process_id[n];

for(i=0; i<n; i++){
    printf("Enter the burst time and priority for process %d:", i+1);
    scanf("%d %d", &burst_time[i], &priority[i]);
    process_id[i] = i+1;
}

for(i=0; i<n-1; i++){
    for(j=i+1; j<n; j++){
        if(priority[i] > priority[j]){
            swap(&priority[i], &priority[j]);
            swap(&burst_time[i], &burst_time[j]);
            swap(&process_id[i], &process_id[j]);
        }
    }
}

int waiting_time[n], turn_around_time[n], total_waiting_time=0,
    total_turnaround_time=0;

for(int i=0; i<n; i++){
    waiting_time[i] = total_waiting_time;
    total_waiting_time += burst_time[i];
}
for(int i=0; i<n; i++){
    turn_around_time[i] = waiting_time[i] + burst_time[i];
    total_turnaround_time += turn_around_time[i];
}
printf("Process ID\t Burst time\t Waiting time\t Turnaround Time\n");
for(i=0; i<n; i++){
    printf("%d\t\t %d\t\t %d\t\t %d\n", process_id[i],
    burst_time[i], priority[i], waiting_time[i], turn_around_time[i]);
    turn_around_time[i]);
```

Date / / /

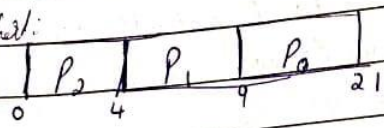
```
pf ("Average waiting time = %f \n", (float) total_waiting_time/n);
printf ("Average turnaround time = %f \n", (float) total_turnaround_time/n);
pf ("Total waiting time = %d \n", total_waiting_time);
pf ("Total turnaround time = %d \n", total_turnaround_time);
}

void Round_Robin() {
    int n, quantum_time, i, f;
    printf ("Enter the number of processes: \n");
    scanf ("%d", &n);

    int burst_time[n], remaining_time[n], waiting_time[n],
        turnaround_time[n];

    for (i=0; i<n; i++) {
        printf ("Enter burst time for process %d: ", i+1);
        scanf ("%d", &burst_time[i]);
        remaining_time[i] = burst_time[i];
    }
    printf ("Enter time quantum: ");
    scanf ("%d", &quantum);

    int time=0, done=0;

    while (done != n) {
        for (i=0; i<n; i++) {
            if (remaining_time[i] > 0) {
                if (remaining_time[i] > quantum) {
                    time += quantum;
                    remaining_time -= quantum;
                }
                else {
                    time += remaining_time[i];
```

```
            waiting_time[i] = time - burst_time[i];
            remaining_time[i] = 0;
            done++;
        }
    }
}
float total_waiting_time = 0, total_turnaround_time = 0;
for (i=0; i<n; i++){
    total_waiting_time += waiting_time[i];
    total_turnaround_time += turnaround_time[i];
}
printf("Process ID\t Burst Time\t Waiting Time\t Turnaround Time\n");
for (i=0; i<n; i++) {
    printf("%d\t\t %d\t\t %d\t\t %d\n", burst_time[i],
        waiting_time[i], turnaround_time[i]);
printf("Average waiting time : %f\n", (float) total_waiting_time/n);
printf("Average turnaround time : %f\n", (float) total_turnaround_time/n);

printf("Total waiting time : %d\n", total_waiting_time);
printf("Total turnaround time : %d\n", total_turnaround_time);

}
```

Working of priority algorithm

Input	Process	priority	burst time.	(from Gantt Chart)	
				Waiting time	turn around time
	P_0	3	12		
	P_1	2	5		
	P_2	1	4		
Sorted according to priority.					
	P_2	1	4	0	4
	P_1	2	5	4	9
	P_0	3	12	9	21

Gantt Chart:

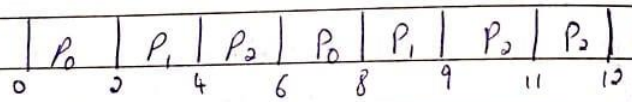


Tracing for Round Robin Scheduling:

Input process	burst time	Waiting time	turnaround
P_0	4	4	8
P_1	3	6	9
P_2	5	7	12

time quantum/slice = 2.

Gantt chart:



Output:

Enter 1 for priority algorithm
2 for Round Robin
3 for exit

1

Enter the number of processes: 3
Enter burst time and priority for process 1: 10 3
Enter burst time and priority for process 2: 5 2
Enter burst time and priority for process 3: 4 1

PROCESS ID	Burst time	Priority	Waiting time	Turnaround time
3	4	1	0	4
2	5	2	4	9
1	10	3	9	21

Average waiting time: 4.333333
Average turnaround time: 11.333333

Date / / /

Total waiting time : 13
Total Turnaround time : 34

Enter 1 for priority algorithm
2 for Round Robin
3 for EXIT
2
Enter number of processes : 3
Enter burst time for process 1 : 4
Enter burst time for process 2 : 3
Enter burst time for process 3 : 5
Enter time quantum : 2

Process ID	Burst time	Waiting time	Turnaround time
1	4	4	8
2	3	6	9
3	5	7	12.

Average waiting time : 5.666667
Average turnaround time : 9.666667
Total waiting time : ~~17.000000~~ 17.000000
Total turnaround time : 29.000000

Enter 1 for priority algorithm
2 for round robin
3 for EXIT
3

23/6/23

Output:

```
Enter 1 for Priority Algorithm
        2 for Round Robin
        3 for exit:
 1
Enter the number of processes: 3
Enter burst time and priority for process 1: 12 3
Enter burst time and priority for process 2: 5 2
Enter burst time and priority for process 3: 4 1
Process ID      Burst Time      Priority        Waiting Time    Turnaround Time
3               4               1               0               4
2               5               2               4               9
1               12              3               9               21
Average waiting time: 4.333333
Average turnaround time: 11.333333
Total Waiting time: 13
Total Turnaround time: 34


Enter 1 for Priority Algorithm
        2 for Round Robin
        3 for exit:
 2
Enter the number of processes: 3
Enter burst time for process 1: 4
Enter burst time for process 2: 3
Enter burst time for process 3: 5
Enter time quantum: 2
Process ID      Burst Time      Waiting Time    Turnaround Time
1               4               4               8
2               3               6               9
3               5               7               12
Average waiting time: 5.666667
Average turnaround time: 9.666667
Total Waiting time: 17.000000
Total Turnaround time: 29.000000


Enter 1 for Priority Algorithm
        2 for Round Robin
        3 for exit:
 3
```