VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

OPERATING SYSTEMS

Submitted by

VANSHIKA KATARIA(1WA23CS031)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Feb-2025 to June-2025

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "OPERATING SYSTEMS – 23CS4PCOPS" carried out by Vanshika kataria(1WA23CS031), who is Bonafide student of B. M. S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year Feb 2025- June 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a OPERATING SYSTEMS - (23CS4PCOPS) work prescribed for the said degree.

Seema Patil
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course Outcomes

C01	Apply the different concepts and functionalities of Operating System
C02	Analyse various Operating system strategies and techniques
C03	Demonstrate the different functionalities of Operating System.
C04	Conduct practical experiments to implement the functionalities of Operating system.

Program -1

Question:

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

→FCFS

→ SJF (pre-emptive & Non-preemptive)

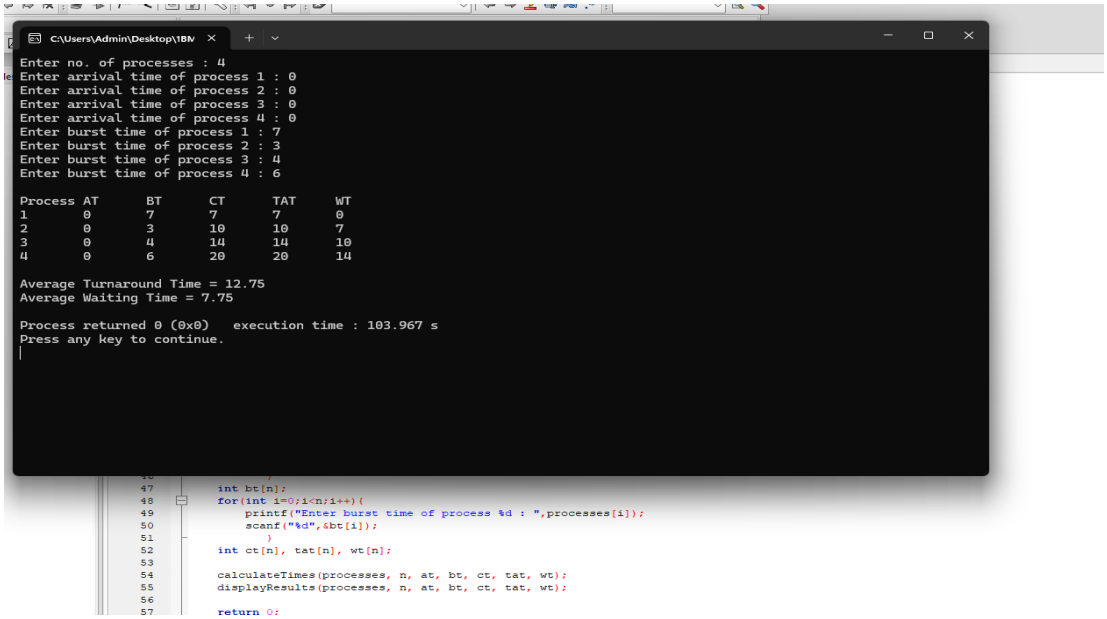
Code:

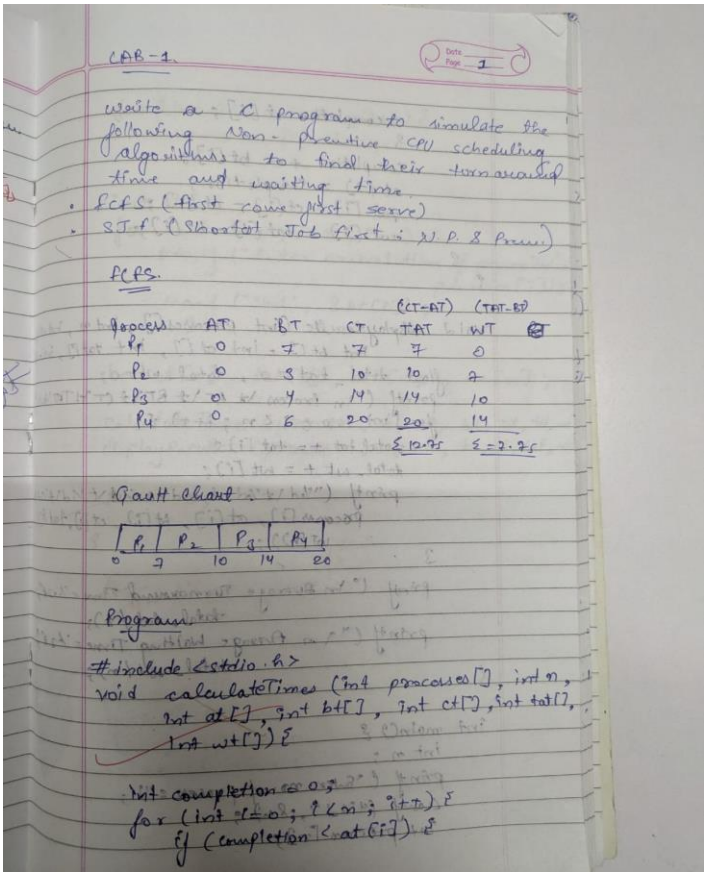
=>FCFS:

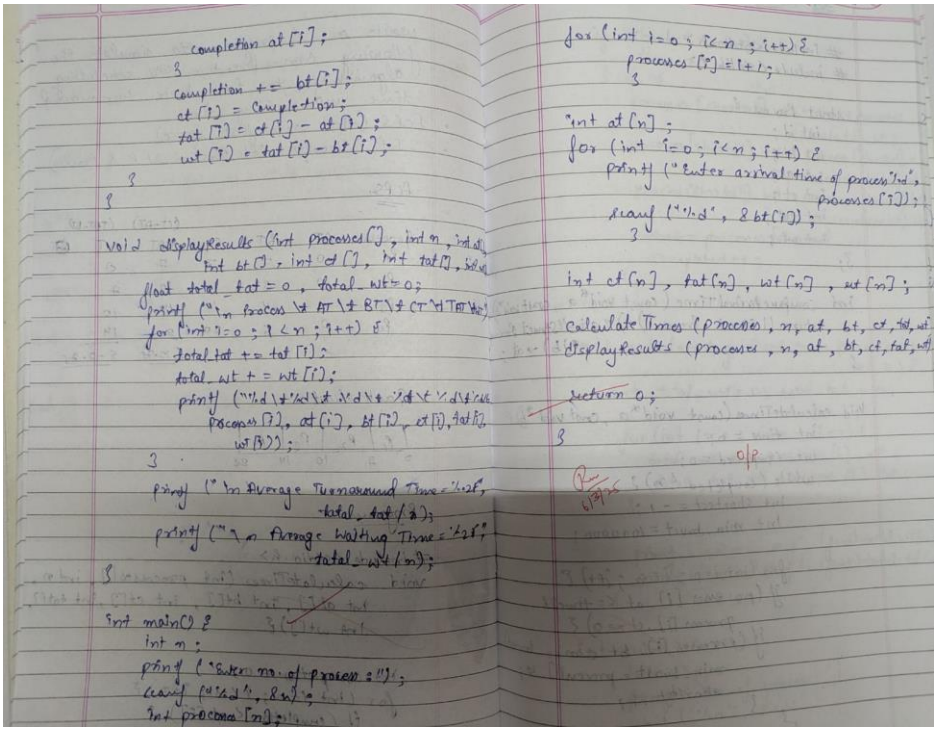
```
#include <stdio.h>
void calculateTimes(int processes[], int n, int at[], int bt[], int ct[], int tat[], int wt[]) {
    int completion = 0;
    for (int i = 0; i < n; i++) {
        if (completion < at[i]) {
            completion = at[i];
        }
        completion += bt[i];
        ct[i] = completion;
        tat[i] = ct[i] - at[i];
        wt[i]=tat[i]-bt[i];
    }
}

void displayResults(int processes[], int n, int at[], int bt[], int ct[], int tat[], int wt[]) {
    float total_tat = 0, total_wt = 0;
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        total_tat += tat[i];
        total_wt += wt[i];
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", processes[i], at[i], bt[i], ct[i], tat[i], wt[i]);
```

```
    }
    printf("\nAverage Turnaround Time = %.2f", total_tat / n);
    printf("\nAverage Waiting Time = %.2f\n", total_wt / n);
}
int main() {
    int n ;
    printf("Enter no. of processes : ");
    scanf("%d",&n);
    int processes[n];
    for(int i=0;i<n;i++){
        processes[i]=i+1;
          }
    int at[n];
    for(int i=0;i<n;i++){
        printf("Enter arrival time of process %d : ",processes[i]);
        scanf("%d",&at[i]);
          }
    int bt[n];
    for(int i=0;i<n;i++){
        printf("Enter burst time of process %d : ",processes[i]);
        scanf("%d",&bt[i]);
          }
    int ct[n], tat[n], wt[n];
    calculateTimes(processes, n, at, bt, ct, tat, wt);
    displayResults(processes, n, at, bt, ct, tat, wt);

    return 0;
}
```

Result:


C:\Users\Admin\Desktop\1BM
Enter no. of processes : 4
Enter arrival time of process 1 : 0
Enter arrival time of process 2 : 0
Enter arrival time of process 3 : 0
Enter arrival time of process 4 : 0
Enter burst time of process 1 : 7
Enter burst time of process 2 : 3
Enter burst time of process 3 : 4
Enter burst time of process 4 : 6
Process AT BT CT TAT WT
1 0 7 7 7 0
2 0 3 10 10 7
3 0 4 14 14 10
4 0 6 20 20 14
Average Turnaround Time = 12.75
Average Waiting Time = 7.75
Process returned 0 (0x0) execution time : 103.967 s
Press any key to continue.
int bt[n];
for(int i=0;i<n;i++){
printf("Enter burst time of process %d : ",processes[i]);
scanf("%d",&bt[i]);
}
int ct[n], tat[n], wt[n];
calculateTimes(processes, n, at, bt, ct, tat, wt);
displayResults(processes, n, at, bt, ct, tat, wt);
return 0;


LAB-1.
Write a C program to simulate the following Non-Preemptive CPU scheduling algorithms to find their turnaround time and waiting time.
FCFS (first come first serve)
SJF (Shortest Job first : N.P. & Preem)
FCFS.
Process AT BT CT (CT-AT) TAT (TAT-BT) WT
P1 0 7 7 7 0
P2 0 3 10 10 7
P3 0 4 14 14 10
P4 0 6 20 20 14
Σ 12.75 Σ = 7.75
Gantt Chart :
P1 P2 P3 P4
0 7 10 14 20
Program
#include <stdio.h>
void calculateTimes (int processes[], int n,
int at[], int bt[], int ct[], int tat[],
int wt[]) {
int completion = 0;
for (int i=0; i<n; i++) {
if (completion < at[i]) {



=>SJF(Non-preemptive):

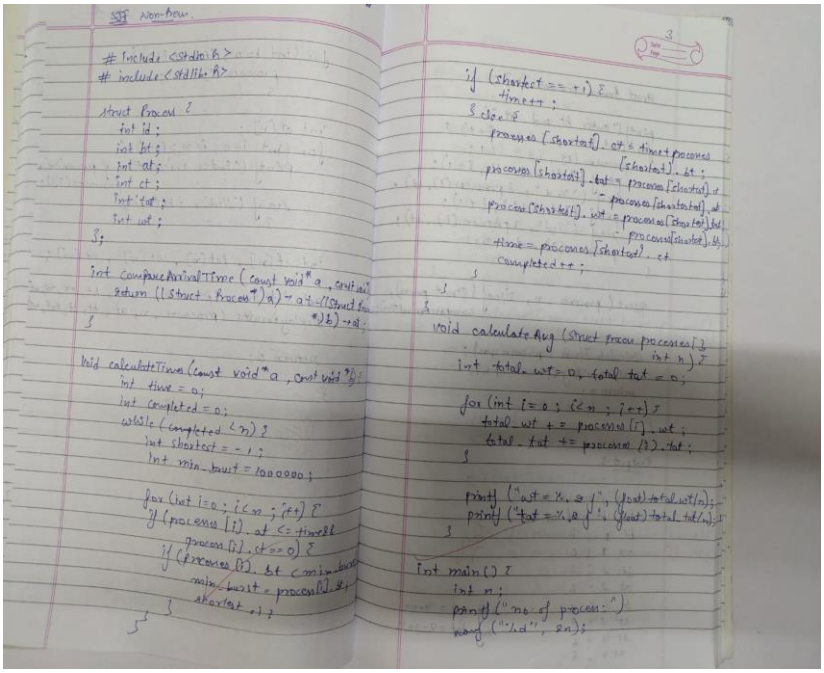
```
#include <stdio.h>
#include <stdlib.h>
struct Process {
    int id;
    int bt;
    int at;
    int ct;
    int tat;
    int wt;
};

int compareArrivalTime(const void *a, const void *b) {
    return ((struct Process*)a)->at - ((struct Process*)b)->at;
}

void calculateTimes(struct Process processes[], int n) {
    int time = 0;
```

```
    int completed = 0;
    while (completed < n) {
        int shortest = -1;
        int min_burst = 1000000;
        for (int i = 0; i < n; i++) {
            if (processes[i].at <= time && processes[i].ct == 0) {
                if (processes[i].bt < min_burst) {
                    min_burst = processes[i].bt;
                    shortest = i;
                }
            }
        }

        if (shortest == -1) {
            time++;
        } else {
            processes[shortest].ct = time + processes[shortest].bt;
            processes[shortest].tat = processes[shortest].ct - processes[shortest].at;
            processes[shortest].wt = processes[shortest].tat - processes[shortest].bt;
            time = processes[shortest].ct;
            completed++;
        }
    }
}

void calculateAvg(struct Process processes[], int n) {
    int total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += processes[i].wt;
        total_tat += processes[i].tat;
    }
    printf("\navg wt = %.2f", (float)total_wt / n);
    printf("\navg tat = %.2f", (float)total_tat / n);
```

```
}

int main() {
    int n;
    printf("no. of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    printf("enter bt and at: \n");
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("bt %d: ", i + 1);
        scanf("%d", &processes[i].bt);
        printf("at %d: ", i + 1);
        scanf("%d", &processes[i].at);
        processes[i].ct = 0;
    }
    qsort(processes, n, sizeof(struct Process), compareArrivalTime);
    calculateTimes(processes, n);
    calculateAvg(processes, n);
    return 0;
}
```

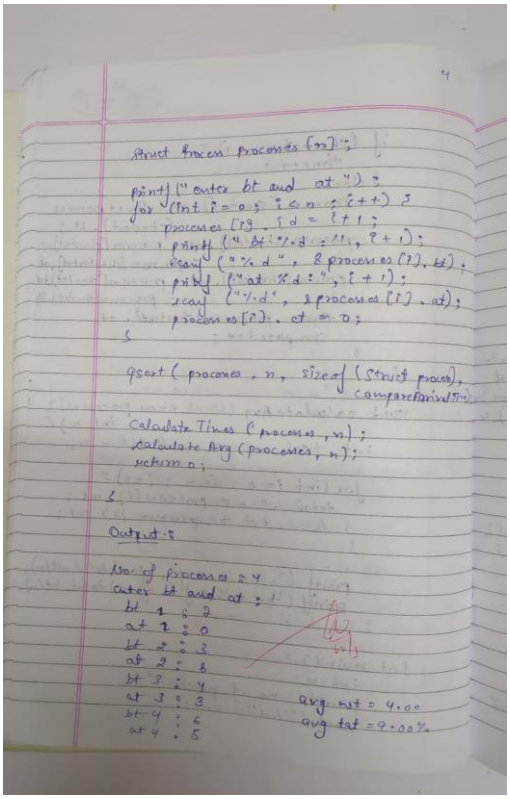
Result:

```
no. of processes: 4
enter bt and at:
bt 1: 7
at 1: 0
bt 2: 3
at 2: 8
bt 3: 4
at 3: 3
bt 4: 6
at 4: 5

avg wt = 4.00
avg tat = 9.00%
```



```
SJF Non-Preem.

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

struct Process {
    int id;
    int bt;
    int at;
    int ct;
    int tat;
    int wt;
};

int compareArrivalTime(const void* a, const void*
    return ((struct Process*)a)->at - ((struct
                                        *)b)->at;
}

void calculateTimes(const void* a, const void*
    int time = 0;
    int completed = 0;
    while (completed < n) {
        int shortest = -1;
        int min_burst = 1000000;

        for (int i=0; i<n; i++) {
            if (process[i].at <= time&&
                process[i].ct == 0) {
                if (process[i].bt < min_burst
                    min_burst = process[i].bt;
                    shortest = i;
                }
            }
        }
```

```
        if (shortest == -1) {
            time++;
        } else {
            process[shortest].ct = time+process
                                   [shortest].bt;
            process[shortest].tat = process[shortest].
                                    - process[shortest].at
            process[shortest].wt = process[shortest].tat
                                   - process[shortest].bt;
            time = process[shortest].ct;
            completed++;
        }
    }
}

void calculateAvg (struct Process processes[],
                   int n) {
    int total_wt=0, total_tat = 0;

    for (int i = 0; i<n; i++) {
        total_wt += process[i].wt;
        total_tat += process[i].tat;
    }

    printf ("wt = %.2f", (float) total_wt/n);
    printf ("tat = %.2f", (float) total_tat/n);
}

int main() {
    int n;
    printf("no of process:")
    scanf("%d", &n);
```



```
struct Process processes[n];

printf("enter bt and at");
for(int i=0; i<n; i++) {
    processes[i].id = i+1;
    printf("\n bt:%d :", i+1);
    scanf("%d", &processes[i].bt);
    printf("at %d:", i+1);
    scanf("%d", &processes[i].at);
    processes[i].ct = 0;
}

qsort(processes, n, sizeof(struct Process),
                    compareArrivalTi

calculateTimes(processes, n);
calculateAvg(processes, n);
return 0;
}

Output:

No. of processes : 4
enter bt and at :
bt 1 : 2
at 1 : 0
bt 2 : 3
at 2 : 1
bt 3 : 4
at 3 : 3
bt 4 : 5
at 4 : 5

avg wt = 4.00
avg tat = 9.00
```

=>SJF(Preemptive):

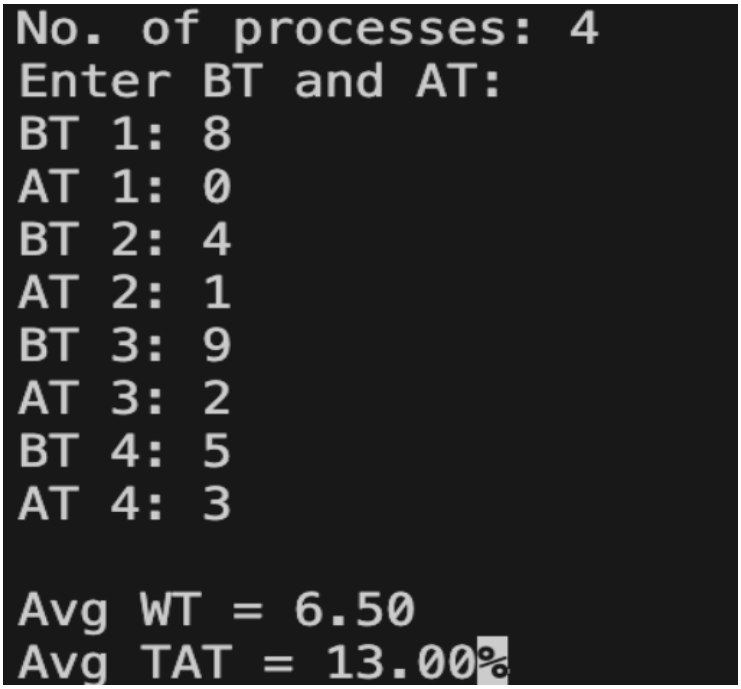
#include <stdio.h>

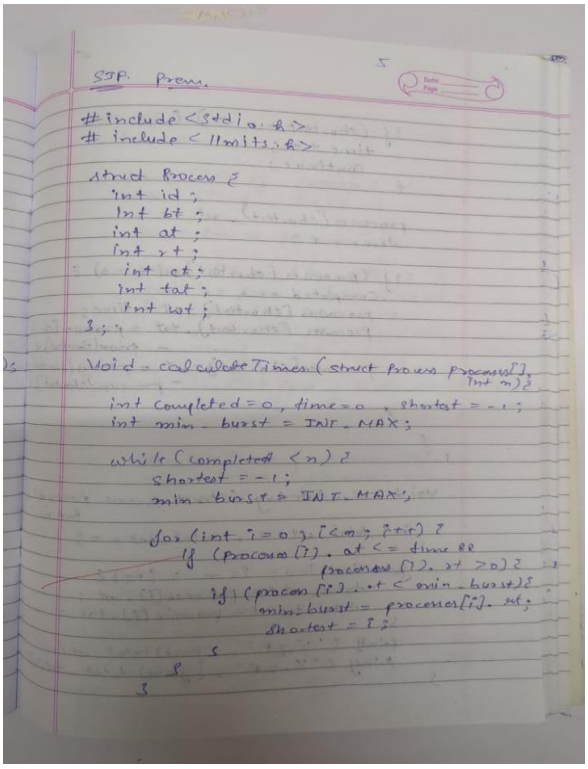
```
#include <limits.h>

struct Process {
    int id;
    int bt;
    int at;
    int rt;
    int ct;
    int tat;
    int wt;
};

void calculateTimes(struct Process processes[], int n) {
    int completed = 0, time = 0, shortest = -1;
    int min_burst = INT_MAX;
    while (completed < n) {
        shortest = -1;
        min_burst = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (processes[i].at <= time && processes[i].rt > 0) {
                if (processes[i].rt < min_burst) {
                    min_burst = processes[i].rt;
                    shortest = i;
                }
            }
        }
        if (shortest == -1) {
            time++;
            continue;
```

```
        }
        processes[shortest].rt--;
        time++;
        if (processes[shortest].rt == 0) {
            completed++;
            processes[shortest].ct = time;
            processes[shortest].tat = processes[shortest].ct - processes[shortest].at;
            processes[shortest].wt = processes[shortest].tat - processes[shortest].bt;
        }
    }
}

void calculateAvg(struct Process processes[], int n) {
    int total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += processes[i].wt;
        total_tat += processes[i].tat;
    }
    printf("\nAvg WT = %.2f", (float)total_wt / n);
    printf("\nAvg TAT = %.2f", (float)total_tat / n);
}

int main() {
    int n;
    printf("No. of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    printf("Enter BT and AT: \n");
    for (int i = 0; i < n; i++) {
```

```
        processes[i].id = i + 1;
        printf("BT %d: ", i + 1);
        scanf("%d", &processes[i].bt);
        printf("AT %d: ", i + 1);
        scanf("%d", &processes[i].at);
        processes[i].rt = processes[i].bt;
    }
    calculateTimes(processes, n);
    calculateAvg(processes, n);
    return 0;
}
```

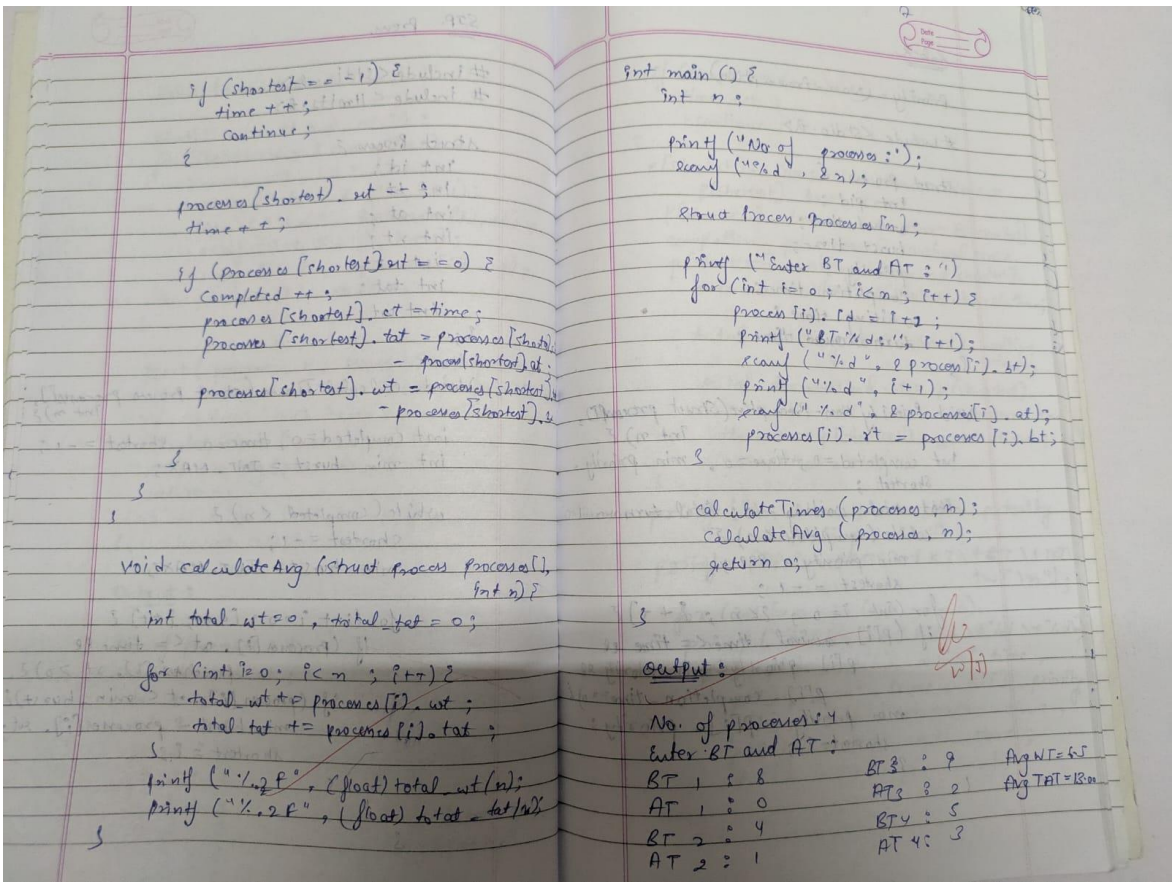
Result:





SJF Prem.

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

struct Process {
    int id;
    int bt;
    int at;
    int rt;
    int ct;
    int tat;
    int wt;
};

void calculateTimes(struct Process processes[], int n) {
    int completed = 0, time = 0, shortest = -1;
    int min_burst = INT_MAX;

    while (completed < n) {
        shortest = -1;
        min_burst = INT_MAX;

        for (int i = 0; i < n; i++) {
            if (processes[i].at <= time &&
                processes[i].rt > 0) {
                if (procon[i].rt < min_burst) {
                    min_burst = processes[i].rt;
                    shortest = i;
                }
            }
        }
```



```
        if (shortest == -1) {
            time++;
            continue;
        }

        processes[shortest].rt--;
        time++;

        if (processes[shortest].rt == 0) {
            completed++;
            processes[shortest].ct = time;
            processes[shortest].tat = processes[shortest].
                                      - procon[shortest].at;
            processes[shortest].wt = processes[shortest].
                                     - processes[shortest].
        }
    }
}

void calculateAvg (struct Process processes[],
                   int n) {
    int total_wt = 0, total_tat = 0;

    for (int i = 0; i < n; i++) {
        total_wt += processes[i].wt;
        total_tat += processes[i].tat;
    }
    printf ("%.2f", (float) total_wt/n);
    printf ("%.2f", (float) total_tat/n);
}

int main () {
    int n;

    printf ("No of processes:");
    scanf ("%d", &n);

    struct Process processes[n];

    printf ("Enter BT and AT:\n");
    for (int i = 0; i < n; i++) {
        procon[i].id = i+1;
        printf ("BT %d:", i+1);
        scanf ("%d", &procon[i].bt);
        printf ("%d", i+1);
        scanf ("%d", &processes[i].at);
        processes[i].rt = processes[i].bt;
    }

    calculateTimes (processes, n);
    calculateAvg (processes, n);
    return 0;
}
```

Output:

```
No. of processes: 4
Enter BT and AT:
BT 1 : 8
AT 1 : 0
BT 2 : 4
AT 2 : 1
BT 3 : 9
AT 3 : 2
BT 4 : 5
AT 4 : 3
Avg WT = 6.5
Avg TAT = 13.00
```

Program-2

Question

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

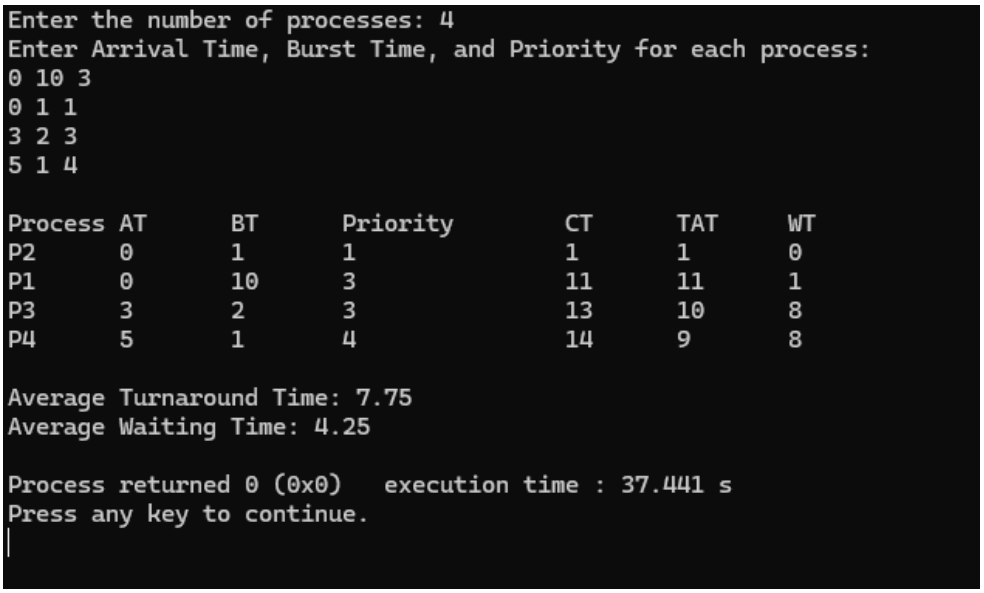
=>PRIORITY(pre-emptive)

```
#include <stdio.h>

struct Process {
   int id, arrival_time, burst_time, priority, completion_time, turnaround_time, waiting_time;
};

void swap(struct Process *a, struct Process *b) {
   struct Process temp = *a;
   *a = *b;
   *b = temp;
}

void sort_by_priority(struct Process p[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (p[j].priority > p[j + 1].priority ||
            (p[j].priority == p[j + 1].priority && p[j].arrival_time > p[j + 1].arrival_time)) {
            swap(&p[j], &p[j + 1]);
         }
      }
```

```
    }
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process p[n];
    printf("Enter Arrival Time, Burst Time, and Priority for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        scanf("%d %d %d", &p[i].arrival_time, &p[i].burst_time, &p[i].priority);
    }

    sort_by_priority(p, n);
    int current_time = 0;
    float total_tat = 0, total_wt = 0;
    for (int i = 0; i < n; i++) {
        if (current_time < p[i].arrival_time) {
            current_time = p[i].arrival_time;
        }
        p[i].completion_time = current_time + p[i].burst_time;
        p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
        p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
        current_time = p[i].completion_time;
        total_tat += p[i].turnaround_time;
        total_wt += p[i].waiting_time;
    }
    printf("\nProcess AT      BT      Priority        CT      TAT    WT\n");
```

```
    for (int i = 0; i < n; i++) {
        printf("P%d %d %d %d %d %d %d\n", p[i].id, p[i].arrival_time, p[i].burst_time, p[i].priority, p[i].completion_time, p[i].turnaround_time, p[i].waiting_time);
    }
    printf("\nAverage Turnaround Time: %.2f", total_tat / n);
    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);
    return 0;
}
```

Result:



Priority (Prem.)

```
#include <stdio.h>
struct Process {
    int pid;
    arrival_time;
    burst_time;
    priority;
    remaining_time;
    completion_time;
    waiting_time;
    turnaround_time;
};
void priorityPreemptive(struct Process p[], int n) {
    int time = 0, min_priority, shortest,
        completed = 0;
    float total_waiting = 0
    total_turnaround = 0;

    for (int i = 0; i < n; i++)
    p[i].remaining_time = p[i].burst_time
    while (completed < n) {
        min_priority = 9999;
        shortest = -1;
    for (int i = 0; i < n; i++) {
        if p[i].arrival_time <= time;
    && p[i].remaining_time > 0 &&
    min_priority = p[i].priority;
    }
}
```

```
if (shortest == -1) {
    time++;
    continue;
}
p[shortest].remaining_time--;
    if (p[shortest].remaining_time == 0) {
    completed++;
    p[shortest].turnaround_time = p[shortest].
    completion_time = p[shortest].arrival_time
    p[shortest].waiting_time = p[shortest].
    turnaround_time = p[shortest].burst_time
    total_waiting += p[shortest].waiting_
                                time;
    total_turnaround += p[shortest].turn-
                                around_time;
    }
    time++;
}
printf("Prem. Priority Scheduling");
printf("PID\tAT\tBT\tP\tCT\tTA
                    \tWT\n")
    for(int i=0; i<n; i++)
    printf("%d\t%d\t%d\t%d\t%d\t%d"
                    p[i].pid)
    p[i].arrival_time, p[i].burst_time
    p[i].priority.
printf("Avg. waiting time: %.2f\n",
                    total_waiting/n);
```

```
printf("Avg Turnaround Time : %.2f\n",
    total_turnaround/n);
}
int main() {
    int n;
    Printf("Enter no. of processes");
    Scanf("%d", &n);

    Struct Process p[n];
    for(int i=0; i<n; i++) {
        printf("Enter Process ID, Arrival Time,
                Burst Time, Priority")
        scanf("%d %d %d %d", &p[i].pid,
            &p[i].arrival_time,
            &p[i].burst_time,
            &p[i].priority);
    }

    priorityPreemptive(p, n);
    return 0;
}
```

15

Output.

```
Enter no. of process: 7
Enter Process ID, AT, BT, Priority: 1 0 8 3
        "    "    "    : 2 1 2 4
        "    "    "    : 3 3 4 4
        "    "    "    : 4 4 1 5
        "    "    "    : 5 5 6 2
        "    "    "    : 6 6 5 6
        "    "    "    : 7 10 1 1
```

Preemptive Priority Scheduling

PID	AT	BT	P	CT	TAT	WT
1	0	8	3	15	15	7
2	1	2	4	17	16	14
3	3	4	4	21	18	14
4	4	1	5	22	18	17
5	5	6	2	12	7	1
6	6	5	6	29	23	18
7	10	1	1	11	1	0

Avg WT: 9.86
Avg TAT: 18.71

=>PRIORITY(Non-pre-emptive)

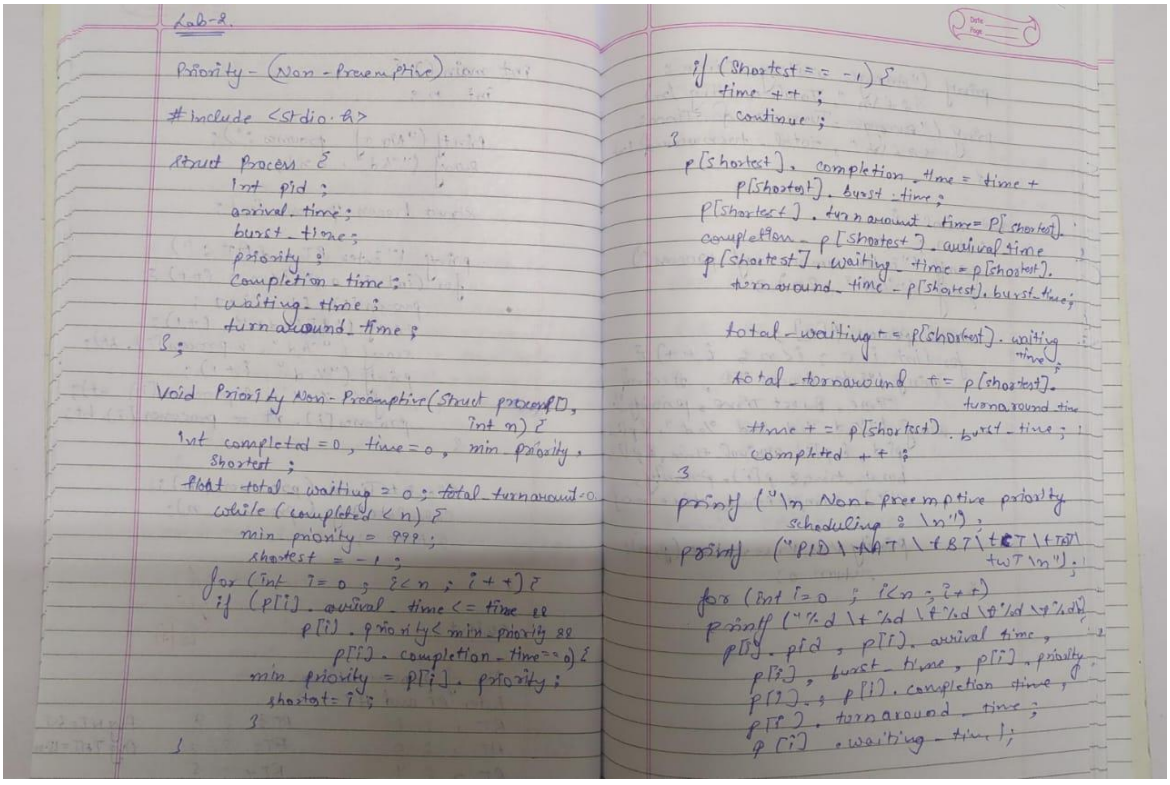
```
#include <stdio.h>

struct Process {
    int id, arrival_time, burst_time, priority, completion_time, turnaround_time,
waiting_time;
};

void swap(struct Process *a, struct Process *b) {
    struct Process temp = *a;
    *a = *b;
    *b = temp;
}

void sort_by_priority(struct Process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].priority > p[j + 1].priority ||
                (p[j].priority == p[j + 1].priority && p[j].arrival_time > p[j +
1].arrival_time)) {
                swap(&p[j], &p[j + 1]);
            }
        }
    }
}

int main() {
```

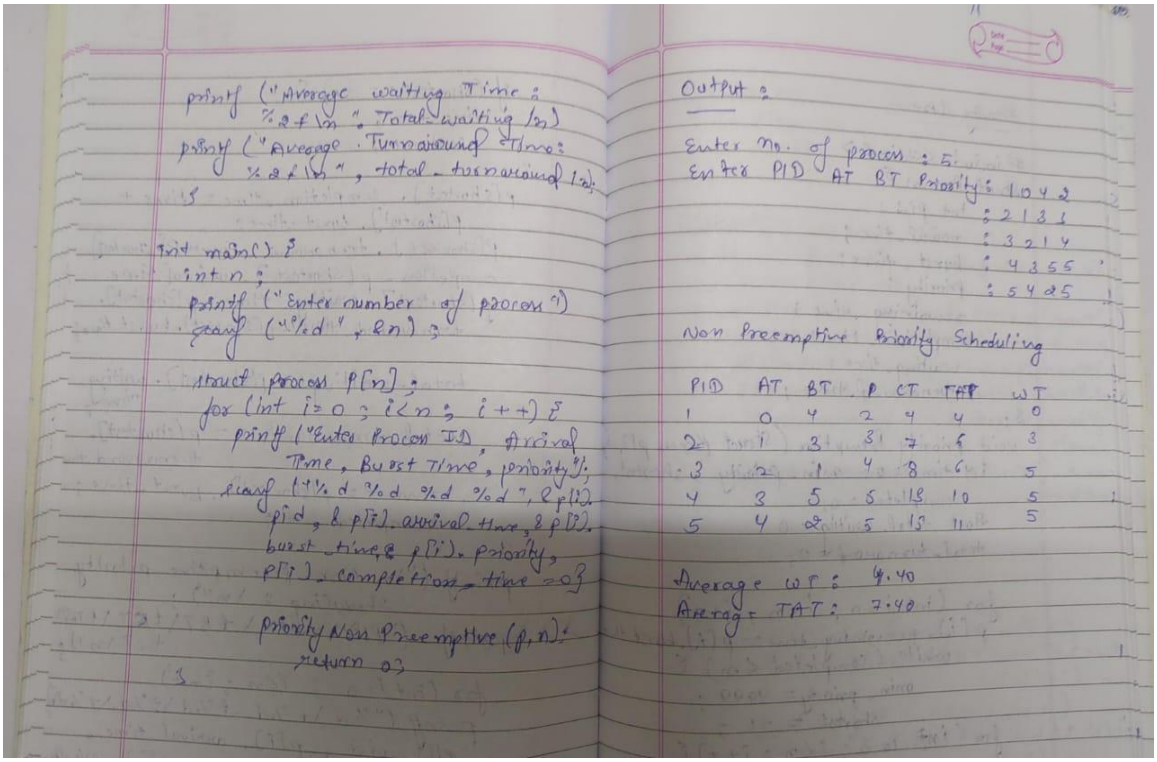
```
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process p[n];
    printf("Enter Arrival Time, Burst Time, and Priority for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        scanf("%d %d %d", &p[i].arrival_time, &p[i].burst_time, &p[i].priority);
    }
    sort_by_priority(p, n);
    int current_time = 0;
    float total_tat = 0, total_wt = 0;
    for (int i = 0; i < n; i++) {
        if (current_time < p[i].arrival_time) {
            current_time = p[i].arrival_time;
        }
        p[i].completion_time = current_time + p[i].burst_time;
        p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
        p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
        current_time = p[i].completion_time;
        total_tat += p[i].turnaround_time;
        total_wt += p[i].waiting_time;
    }
    printf("\nProcess\tAT\tBT\tPriority\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t\t%d\t%d\t%d\n", p[i].id, p[i].arrival_time,
p[i].burst_time, p[i].priority, p[i].completion_time, p[i].turnaround_time,
p[i].waiting_time);
    }
```

```
    printf("\nAverage Turnaround Time: %.2f", total_tat / n);
    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);
    return 0;
}
```

Result:

```
Enter the number of processes: 4
Enter Arrival Time, Burst Time, and Priority for each process:
0 5 4
2 4 2
2 2 6
4 4 3

Process AT      BT      Priority        CT      TAT     WT
P2      2       4       2               6       4       0
P4      4       4       3               10      6       2
P1      0       5       4               15      15      10
P3      2       2       6               17      15      13

Average Turnaround Time: 10.00
Average Waiting Time: 6.25

Process returned 0 (0x0)   execution time : 27.552 s
Press any key to continue.
```





```
printf ("Average waiting Time :
    %.2f\n", Total_waiting/n)
printf ("Average Turnaround Time:
    %.2f\n", total_turnaround/n);
}

int main() {
  int n;
  printf ("Enter number of process");
  scanf ("%d", &n);

  struct process p[n];
  for (int i=0; i<n; i++) {
    printf ("Enter Process ID, Arrival
      Time, Burst Time, priority%:
    scanf ("%d %d %d %d", &p[i].
      pid, &p[i].arrival_time, &p[i].
      burst_time, &p[i].priority,
      p[i].completion_time =0}

    priority Non Preemptive (p, n);
      return 0;
}
```

```
Output :

Enter no. of process : 5
Enter PID AT BT Priority : 1 0 4 2
                         : 2 1 3 1
                         : 3 2 1 4
                         : 4 3 5 5
                         : 5 4 2 5

Non Preemptive Priority Scheduling

PID  AT  BT  P  CT  TAT  WT
1    0   4   2  4   4    0
2    1   3   3  7   6    3
3    2   1   4  8   6    5
4    3   5   5  13  10   5
5    4   2   5  15  11   5

Average WT :  4.40
Average TAT :  7.40
```

=>ROUND ROBIN

```
#include <stdio.h>

struct Process {
    int id, bt, at, rt, ct, tat, wt;
};

void roundRobin(struct Process processes[], int n, int quantum) {
    int time = 0, completed = 0;
    while (completed < n) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (processes[i].rt > 0 && processes[i].at <= time) {
```

```
            done = 0;
            if (processes[i].rt > quantum) {
                time += quantum;
                processes[i].rt -= quantum;
            } else {
                time += processes[i].rt;
                processes[i].ct = time;
                processes[i].tat = processes[i].ct - processes[i].at;
                processes[i].wt = processes[i].tat - processes[i].bt;
                processes[i].rt = 0;
                completed++;
            }
        }
    }
    if (done) time++;
  }
}

void calculateAvg(struct Process processes[], int n) {
  int total_wt = 0, total_tat = 0;
  for (int i = 0; i < n; i++) {
    total_wt += processes[i].wt;
    total_tat += processes[i].tat;
  }
  printf("\nAvg WT = %.2f", (float)total_wt / n);
  printf("\nAvg TAT = %.2f", (float)total_tat / n);
}

int main() {
```

```
    int n, quantum;
    printf("No. of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("BT %d: ", i + 1);
        scanf("%d", &processes[i].bt);
        printf("AT %d: ", i + 1);
        scanf("%d", &processes[i].at);
        processes[i].rt = processes[i].bt;
    }
    printf("Time Quantum: ");
    scanf("%d", &quantum);
    roundRobin(processes, n, quantum);
    calculateAvg(processes, n);
    return 0;
}
```

Result:

```
No. of processes: 5
BT 1: 8
AT 1: 0
BT 2: 2
AT 2: 5
BT 3: 7
AT 3: 1
BT 4: 3
AT 4: 6
BT 5: 5
AT 5: 8
Time Quantum: 3

Avg WT = 10.40
Avg TAT = 15.40
```

Program-3

Question:

Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

=>MULTILEVEL QUEUE

```
#include <stdio.h>

#define MAX_PROCESSES 10
#define TIME_QUANTUM 2

typedef struct {
    int id;
    int burst_time;
    int arrival_time;
    int queue; // 1 for system process (RR), 2 for user process (FCFS)
    int remaining_time;
    int waiting_time;
    int turnaround_time;
    int response_time;
} Process;

void round_robin(Process processes[], int n, int tq, int *time) {
    int done, i;
    do {
        done = 1;
        for (i = 0; i < n; i++) {
```

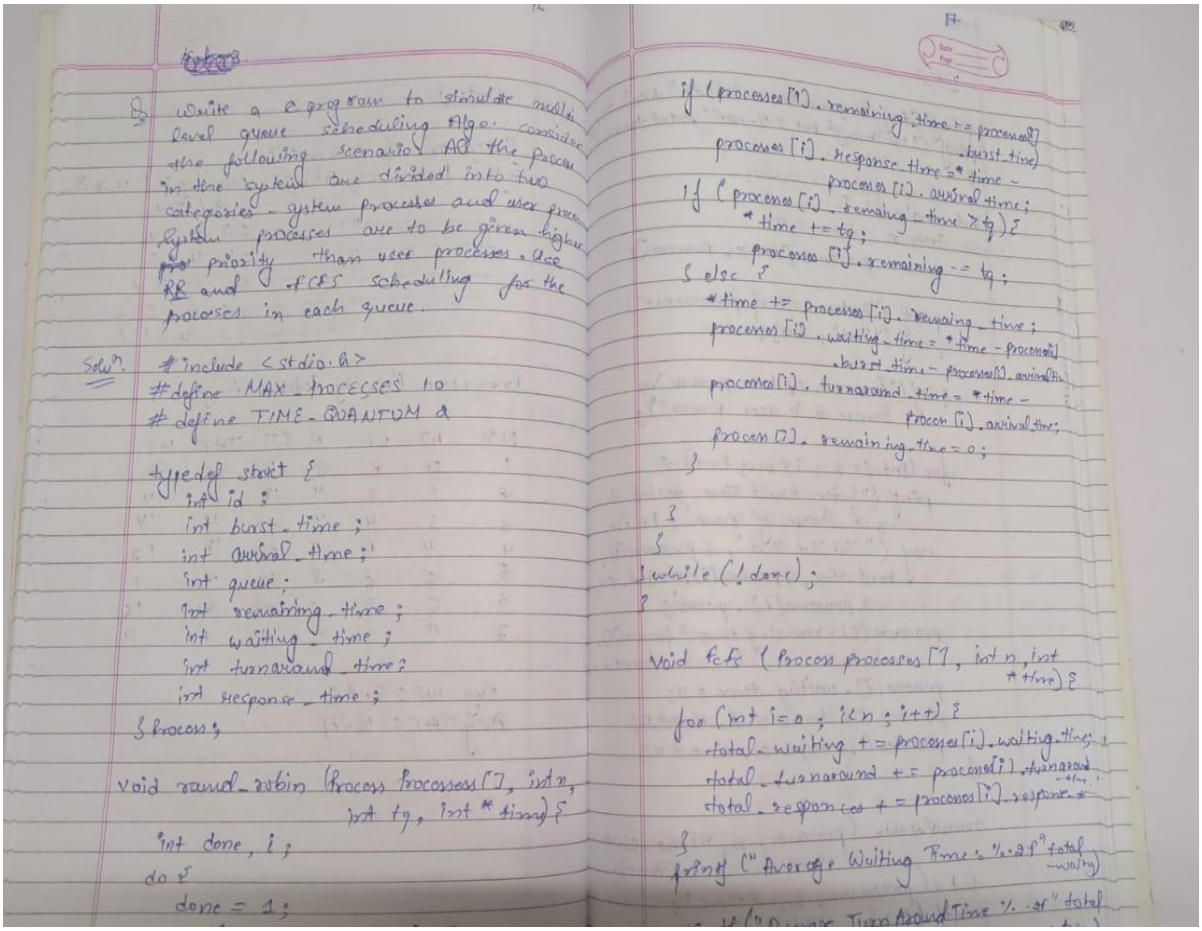
```
            if (processes[i].queue == 1 && processes[i].remaining_time > 0) {
                done = 0;
                if (processes[i].remaining_time == processes[i].burst_time)
                    processes[i].response_time = *time - processes[i].arrival_time;
                if (processes[i].remaining_time > tq) {
                    *time += tq;
                    processes[i].remaining_time -= tq;
                } else {
                    *time += processes[i].remaining_time;
                    processes[i].waiting_time = *time - processes[i].burst_time -
processes[i].arrival_time;
                    processes[i].turnaround_time = *time - processes[i].arrival_time;
                    processes[i].remaining_time = 0;
                }
            }
        }
    } while (!done);
}

void fcfs(Process processes[], int n, int *time) {
    for (int i = 0; i < n; i++) {
        if (processes[i].queue == 2) {
            if (*time < processes[i].arrival_time)
                *time = processes[i].arrival_time;
            processes[i].response_time = *time - processes[i].arrival_time;
            processes[i].waiting_time = *time - processes[i].arrival_time;
            *time += processes[i].burst_time;
            processes[i].turnaround_time = *time - processes[i].arrival_time;
        }
```

```
    }
}

void calculate_average(Process processes[], int n) {
    float total_waiting = 0, total_turnaround = 0, total_response = 0;
    for (int i = 0; i < n; i++) {
        total_waiting += processes[i].waiting_time;
        total_turnaround += processes[i].turnaround_time;
        total_response += processes[i].response_time;
    }
    printf("\nAverage Waiting Time: %.2f", total_waiting / n);
    printf("\nAverage Turn Around Time: %.2f", total_turnaround / n);
    printf("\nAverage Response Time: %.2f", total_response / n);
    printf("\nThroughput: %.2f\n", n / (total_turnaround / n));
}

int main() {
    int n, time = 0;
    Process processes[MAX_PROCESSES];
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Queue 1 is system process\nQueue 2 is User Process\n");
    for (int i = 0; i < n; i++) {
        printf("Enter Burst Time, Arrival Time and Queue of P%d: ", i + 1);
        scanf("%d %d %d", &processes[i].burst_time, &processes[i].arrival_time,
&processes[i].queue);
        processes[i].id = i + 1;
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].waiting_time = 0;
```

```
        processes[i].turnaround_time = 0;
        processes[i].response_time = 0;
    }
    round_robin(processes, n, TIME_QUANTUM, &time);
    fcfs(processes, n, &time);
    printf("\nProcess\tWaiting Time\tTurn Around Time\tResponse Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t\t%d\n", processes[i].id, processes[i].waiting_time,
processes[i].turnaround_time, processes[i].response_time);
    }
    calculate_average(processes, n);
    return 0;
}
```

Result:

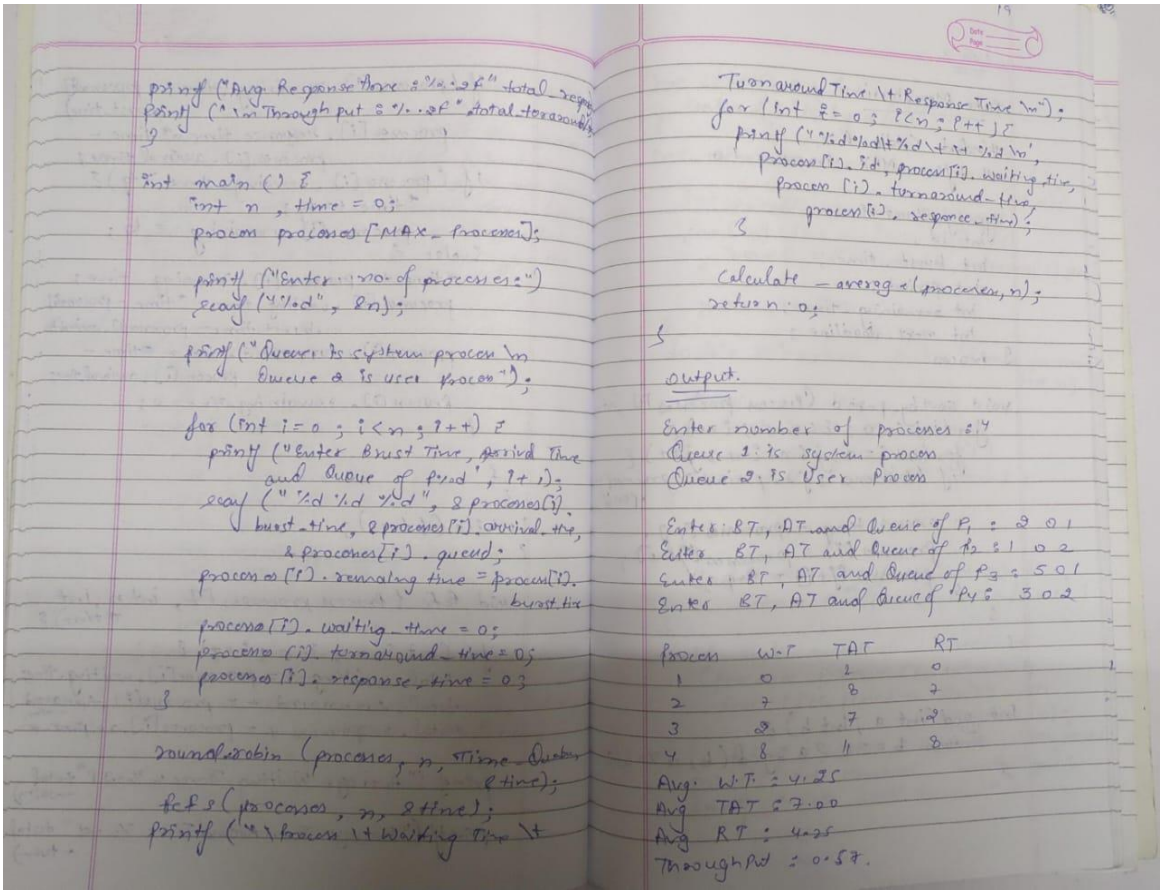
```
Enter number of processes: 4
Queue 1 is system process
Queue 2 is User Process
Enter Burst Time, Arrival Time and Queue of P1: 2 0 1
Enter Burst Time, Arrival Time and Queue of P2: 1 0 2
Enter Burst Time, Arrival Time and Queue of P3: 5 0 1
Enter Burst Time, Arrival Time and Queue of P4: 3 0 2

Process Waiting Time    Turn Around Time        Response Time
1       0               2                       0
2       7               8                       7
3       2               7                       2
4       8               11                      8

Average Waiting Time: 4.25
Average Turn Around Time: 7.00
Average Response Time: 4.25
Throughput: 0.57

Process returned 0 (0x0)   execution time : 71.241 s
Press any key to continue.
```



Q. Write a program to simulate multi level queue scheduling algo. consider the following scenario. All the process in the system are divided into two categories - system processes and user processes. System processes are to be given higher priority than user processes. Use RR and FCFS scheduling for the process in each queue.

Soln.

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define TIME_QUANTUM 4

typedef struct {
    int id;
    int burst_time;
    int arrival_time;
    int queue;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
    int response_time;
} Process;

void round_robin(Process processes[], int n,
                 int tq, int *time) {
    int done, i;
    do {
        done = 1;
        if (processes[i].remaining_time == processes[i]
                                          .burst_time)
            processes[i].response_time = *time -
                                processes[i].arrival_time;
        if (processes[i].remaining_time > tq) {
            *time += tq;
            processes[i].remaining_time -= tq;
        } else {
            *time += processes[i].remaining_time;
            processes[i].waiting_time = *time - processes[i]
                                .burst_time - processes[i].arrival_time;
            processes[i].turnaround_time = *time -
                                processes[i].arrival_time;
            processes[i].remaining_time = 0;
        }
    }
    } while (!done);
}

void fcfs (Process processes[], int n, int
                                    *time) {
    for (int i = 0; i < n; i++) {
        total_waiting += processes[i].waiting_time;
        total_turnaround += processes[i].turnaround
                                        _time;
        total_response += processes[i].response_time;
    }
    printf("Average Waiting Time: %.2f", total
                                        _waiting
```



Program-4

Question:

Write a C program to simulate Real-Time CPU Scheduling algorithms:
a)Rate- Monotonic
b)Earliest-deadline First
c)Proportional scheduling

=>RATE-MONOTONIC

```
#include <stdio.h>
#define MAX_PROCESSES 10

typedef struct {
    int id;
    int burst_time;
```

```
    int period;
    int remaining_time;
    int next_deadline;
} Process;

void sort_by_period(Process processes[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].period > processes[j + 1].period) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

int calculate_lcm(Process processes[], int n) {
    int result = processes[0].period;
    for (int i = 1; i < n; i++) {
        result = lcm(result, processes[i].period);
```

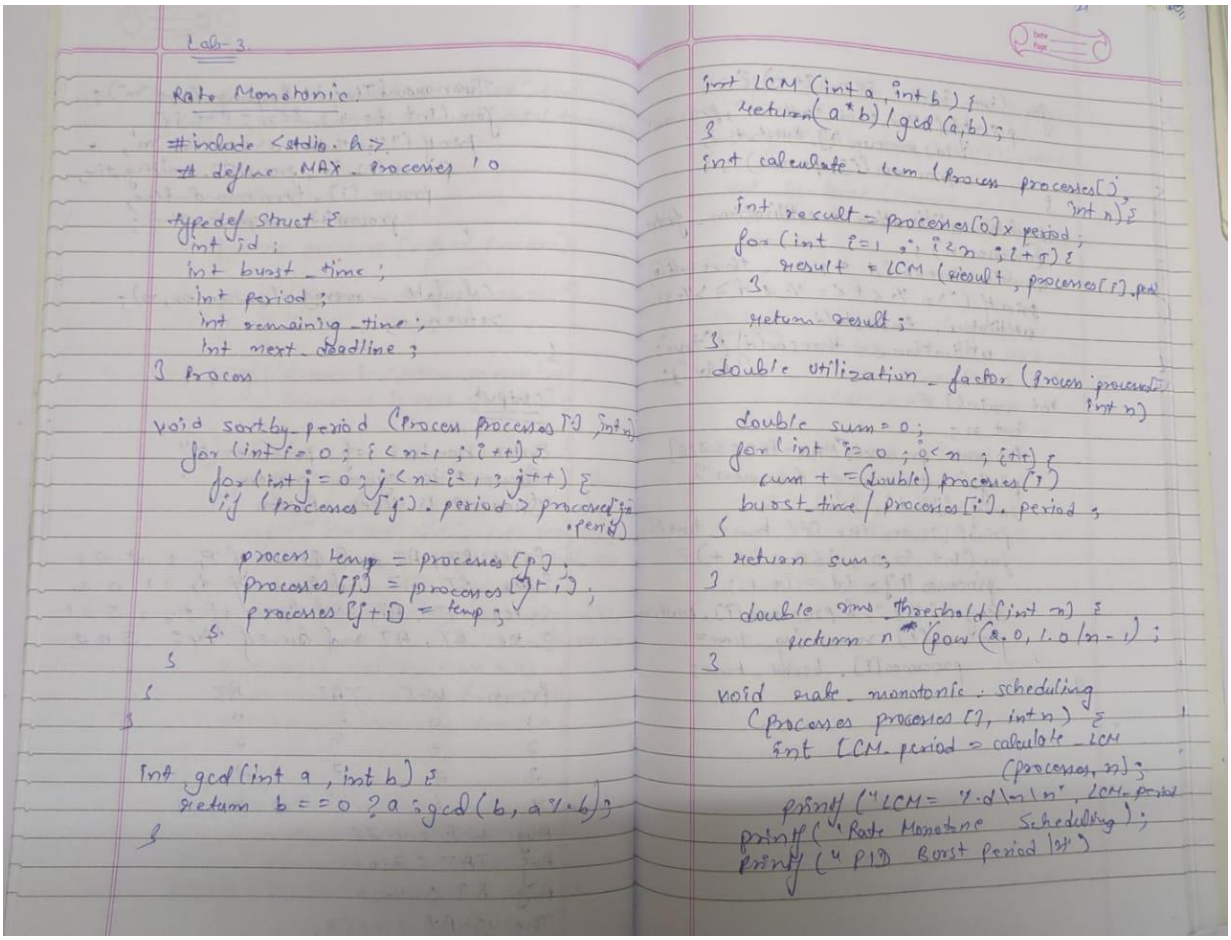
```
    }
    return result;
}

double utilization_factor(Process processes[], int n) {
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += (double)processes[i].burst_time / processes[i].period;
    }
    return sum;
}

double rms_threshold(int n) {
    return n * (pow(2.0, 1.0 / n) - 1);
}

void rate_monotonic_scheduling(Process processes[], int n) {
    int lcm_period = calculate_lcm(processes, n);
    printf("LCM=%d\n\n", lcm_period);

    printf("Rate Monotone Scheduling:\n");
    printf("PID  Burst  Period\n");
    for (int i = 0; i < n; i++) {
        printf("%d    %d      %d\n", processes[i].id, processes[i].burst_time,
processes[i].period);
    }

    double utilization = utilization_factor(processes, n);
    double threshold = rms_threshold(n);
```

```
    printf("\n%.6f <= %.6f => %s\n", utilization, threshold, (utilization <= threshold)
? "true" : "false");

    if (utilization > threshold) {
        printf("\nSystem may not be schedulable!\n");
        return;
    }

    int timeline = 0, executed = 0;
    while (timeline < lcm_period) {
        int selected = -1;
        for (int i = 0; i < n; i++) {
            if (timeline % processes[i].period == 0) {
                processes[i].remaining_time = processes[i].burst_time;
            }
            if (processes[i].remaining_time > 0) {
                selected = i;
                break;
            }
        }
        if (selected != -1) {
            printf("Time %d: Process %d is running\n", timeline, processes[selected].id);
            processes[selected].remaining_time--;
            executed++;
        } else {
            printf("Time %d: CPU is idle\n", timeline);
        }
        timeline++;
    }
```

```
}

int main() {
    int n;
    Process processes[MAX_PROCESSES];
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
    }
    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].period);
    }
    sort_by_period(processes, n);
    rate_monotonic_scheduling(processes, n);

    return 0;
```

Result:

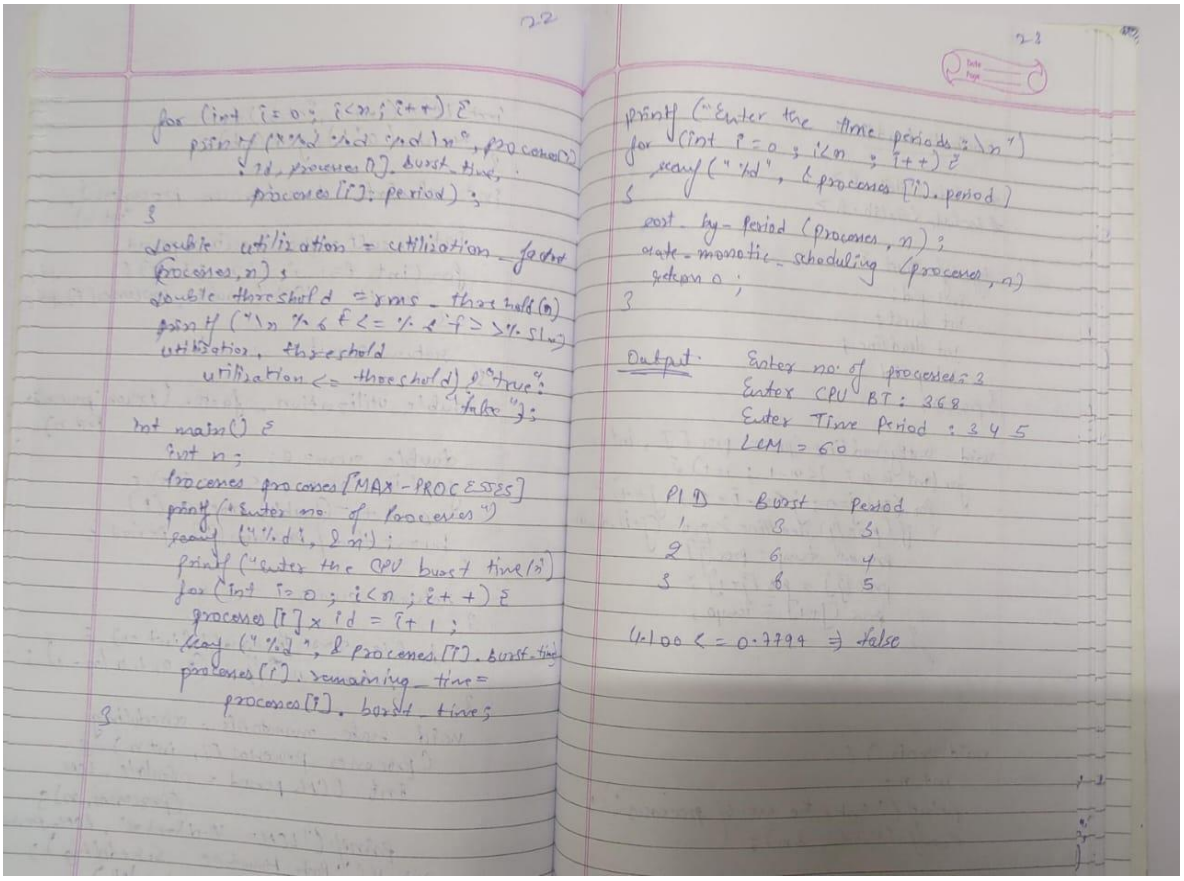
```
Enter the number of processes: 3
Enter the CPU burst times:
3
6 8
Enter the time periods:
3 4 5
LCM=60

Rate Monotone Scheduling:
PID  Burst  Period
1    3      3
2    6      4
3    8      5

4.100000 <= 0.779763 => false

System may not be schedulable!

Process returned 0 (0x0)   execution time : 18.410 s
Press any key to continue.
```



Lab-3

Rate Monotonic:

```
#include <stdio.h>
#define MAX_process 10

typedef struct {
    int id;
    int burst_time;
    int period;
    int remaining_time;
    int next_deadline;
} Process;

void sortby_period (Process processes[], int n)
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (processes[j].period > processes[j+1].period)
                process temp = processes[j];
                processes[j] = processes[j+1];
                processes[j+1] = temp;
            }
        }
    }
}

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}

int LCM (int a, int b) {
    return (a*b)/gcd(a,b);
}

int calculate_lcm (Process processes[], int n) {
    int result = processes[0].period;
    for (int i=1; i<n; i++) {
        result = LCM (result, processes[i].period);
    }
    return result;
}

double utilization_factor (Process processes[], int n)
    double sum = 0;
    for (int i = 0; i<n; i++) {
        sum += (double) processes[i].
        burst_time / processes[i].period;
    }
    return sum;
}

double rms_threshold (int n) {
    return n * (pow (2.0, 1.0/n - 1));
}

void rate_monotonic_scheduling
    (Process processes[], int n) {
    int LCM_period = calculate_LCM
        (processes, n);
    printf ("LCM = %d\n\n", LCM_period);
    printf ("Rate Monotone Scheduling:\n");
    printf ("PID Burst Period\n");
```



=>EARLIEST-DEADLINE

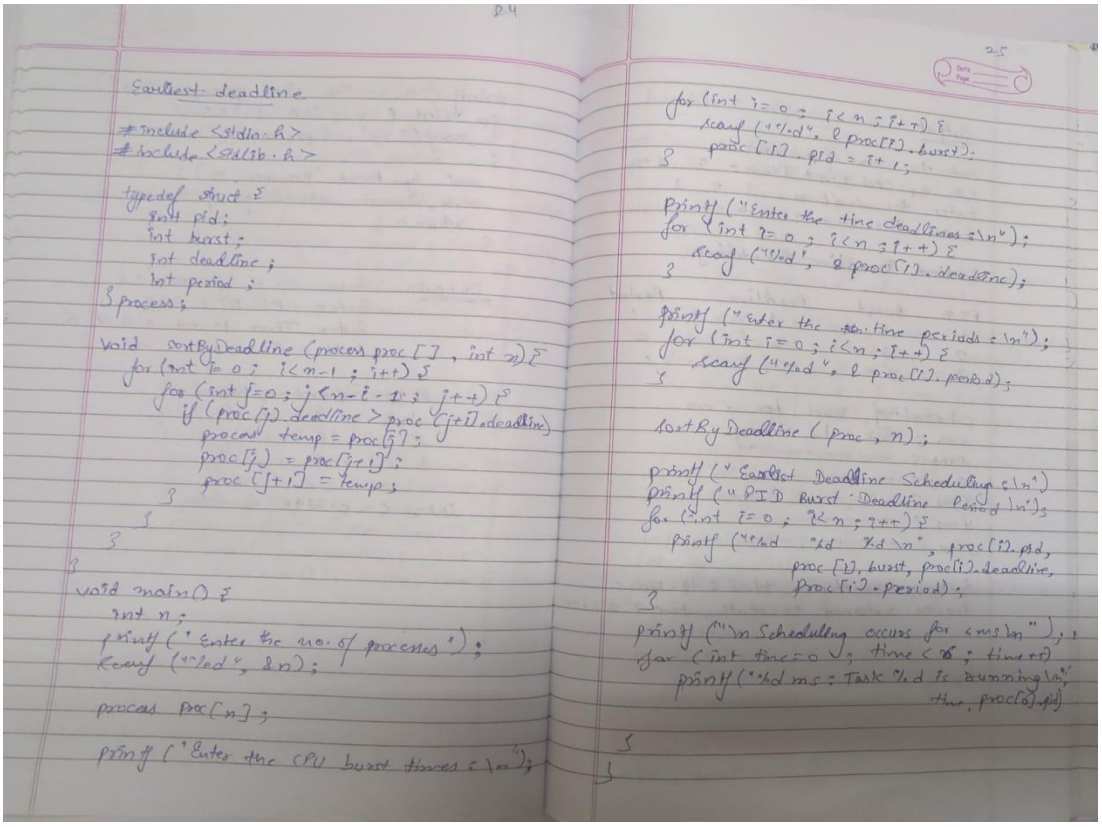
```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int pid;
    int burst;
    int deadline;
    int period;
} Process;

void sortByDeadline(Process proc[], int n) {
```

```
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (proc[j].deadline > proc[j + 1].deadline) {
                Process temp = proc[j];
                proc[j] = proc[j + 1];
                proc[j + 1] = temp;
            }
        }
    }
}

void main() {
    int n;
    printf("Enter the number of processes:");
    scanf("%d", &n);
    Process proc[n];
    printf("\nEnter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &proc[i].burst);
        proc[i].pid = i + 1;
    }
    printf("\nEnter the deadlines:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &proc[i].deadline);
    }
    printf("\nEnter the time periods:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &proc[i].period);
    }
```

```
    sortByDeadline(proc, n);
    printf("\nEarliest Deadline Scheduling:\n");
    printf("PID    Burst    Deadline    Period\n");
    for (int i = 0; i < n; i++) {
        printf("%d      %d        %d          %d\n", proc[i].pid, proc[i].burst,
proc[i].deadline, proc[i].period);
    }
    printf("\nScheduling occurs for 6 ms\n");
    for (int time = 0; time < 6; time++) {
        printf("%dms : Task %d is running.\n", time, proc[0].pid);
    }
}
```

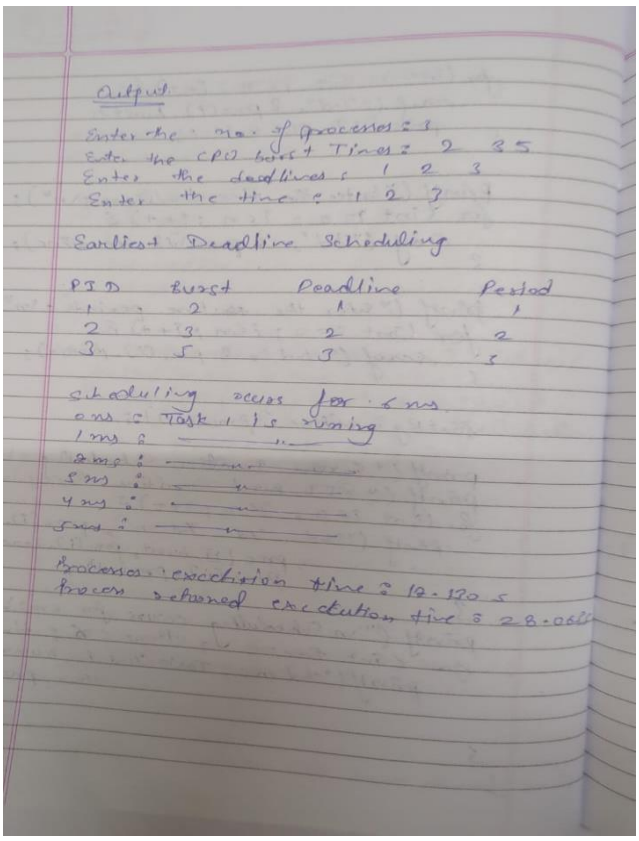
Result:

```
Enter the number of processes:3

Enter the CPU burst times:
2 3 5

Enter the deadlines:
1 2 3

Enter the time periods:
1 2 3

Earliest Deadline Scheduling:
PID     Burst     Deadline     Period
1       2         1            1
2       3         2            2
3       5         3            3

Scheduling occurs for 6 ms
0ms : Task 1 is running.
1ms : Task 1 is running.
2ms : Task 1 is running.
3ms : Task 1 is running.
4ms : Task 1 is running.
5ms : Task 1 is running.

Process returned 6 (0x6)   execution time : 17.130 s
Press any key to continue.

Process returned 0 (0x0)   execution time : 28.063 s
Press any key to continue.
```



Earliest deadline

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

typedef struct {
    int pid;
    int burst;
    int deadline;
    int period;
} process;

void sortByDeadline(process proc[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (proc[j].deadline > proc[j+1].deadline) {
                process temp = proc[j];
                proc[j] = proc[j+1];
                proc[j+1] = temp;
            }
        }
    }
}

void main() {
    int n;
    printf("Enter the no. of processes: ");
    scanf("%d", &n);

    process proc[n];

    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &proc[i].burst);
        proc[i].pid = i+1;
    }

    printf("Enter the time deadlines:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &proc[i].deadline);
    }

    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &proc[i].period);
    }

    sortByDeadline(proc, n);

    printf("Earliest Deadline Scheduling:\n");
    printf("PID Burst Deadline Period\n");
    for (int i = 0; i < n; i++) {
        printf("%d %d %d %d\n", proc[i].pid,
               proc[i].burst, proc[i].deadline,
               proc[i].period);
    }

    printf("\nScheduling occurs for %ms\n");
    for (int time = 0; time < n; time++)
        printf("%d ms: Task %d is running\n",
               time, proc[0].pid);
}
```



Output

```
Enter the no. of processes: 3
Enter the CPU burst Times: 2 3 5
Enter the deadlines: 1 2 3
Enter the time: 1 2 3

Earliest Deadline Scheduling

PID   Burst   Deadline   Period
1     2       1          1
2     3       2          2
3     5       3          3

Scheduling occurs for 5ms
0ms: Task 1 is running
1ms: ____ "____
2ms: ____ "____
3ms: ____ "____
4ms: ____ "____
5ms: ____ "____

Process: execution time: 10.120 s
Process returned execution time: 28.06[illegible]
```

Program-5

Question:

Write a C program to simulate producer-consumer problem using semaphores

=>PRODUCER-CONSUMER

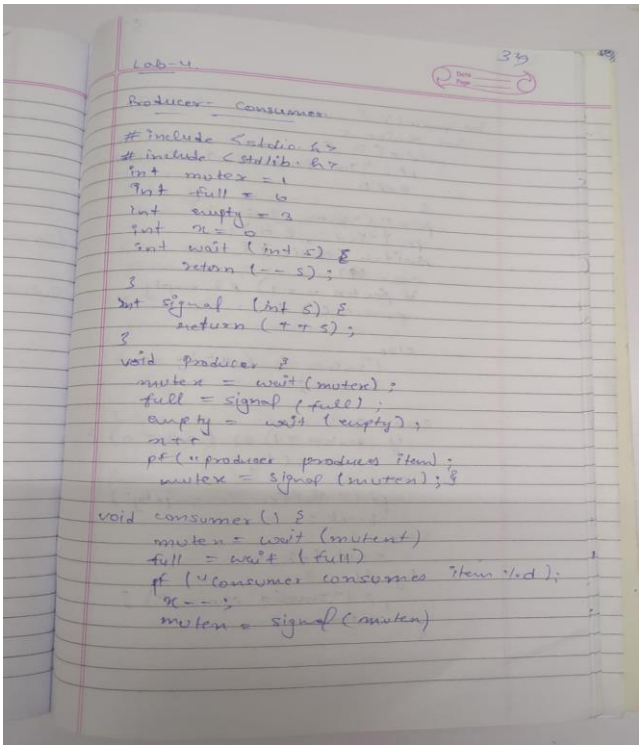
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int mutex = 1, full = 0, empty = 3, x = 0, buffer = 0;

int wait(int s) {
   return (--s);
}

int signal(int s) {
   return (++s);
}

void producer() {
   mutex = wait(mutex);
   full = signal(full);
   empty = wait(empty);
   x = rand() % 50;
   buffer = x;
   printf("Producer 1 produced %d\n", x);
   printf("Buffer:%d\n", buffer);
```

```
    mutex = signal(mutex);
}

void consumer() {
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("Consumer 2 consumed %d\n", buffer);
    printf("Current buffer len: 0\n");
    x--;
    mutex = signal(mutex);
}

void main() {
    int choice, p, c;
    srand(time(0));
    printf("Enter the number of Producers:");
    scanf("%d", &p);
    printf("Enter the number of Consumers:");
    scanf("%d", &c);
    printf("Enter buffer capacity:");
    scanf("%d", &empty);
    for (int i = 1; i <= p; i++)
        printf("Successfully created producer %d\n", i);
    for (int i = 1; i <= c; i++)
        printf("Successfully created consumer %d\n", i);

    while (1) {
        printf("\n1.Producer\n2.Consumer\n3.Exit\n");
```

```
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                if ((mutex == 1) && (empty != 0))
                    producer();
                else
                    printf("Buffer is full\n");
                break;
            case 2:
                if ((mutex == 1) && (full != 0))
                    consumer();
                else
                    printf("Buffer is empty\n");
                break;
            case 3:
                exit(0);
        }
    }
}
```

Result:

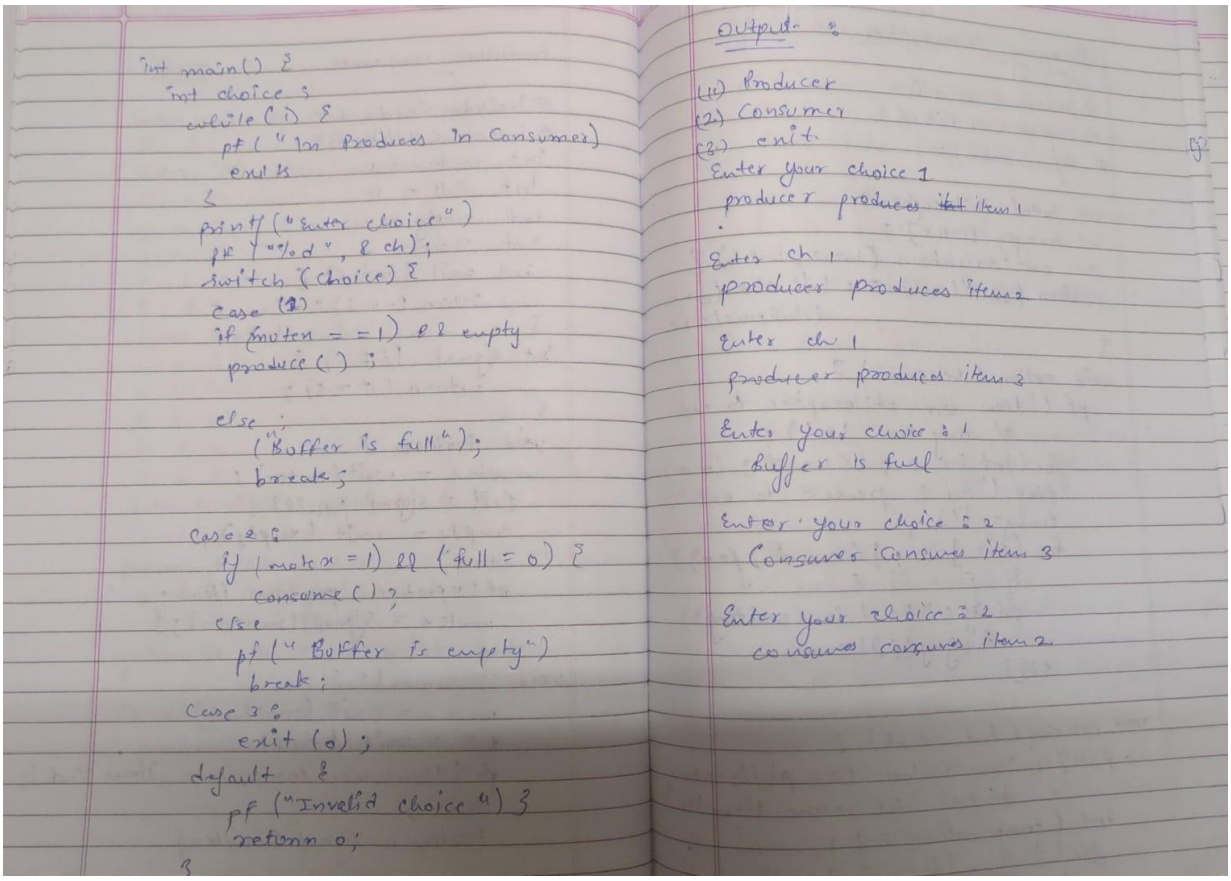
```
Enter the number of Producers:1
Enter the number of Consumers:1
Enter buffer capacity:1
Successfully created producer 1
Successfully created consumer 1

1.Producer
2.Consumer
3.Exit
1
Producer 1 produced 17
Buffer:17

1.Producer
2.Consumer
3.Exit
2
Consumer 2 consumed 17
Current buffer len: 0

1.Producer
2.Consumer
3.Exit
1
Producer 1 produced 16
Buffer:16

1.Producer
2.Consumer
3.Exit
2
Consumer 2 consumed 16
Current buffer len: 0

1.Producer
2.Consumer
3.Exit
```



Lab-4 33

Producer- Consumer.

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1
int full = 0
int empty = 3
int x = 0
int wait (int s) {
    return (--s);
}
int signal (int s) {
    return (++s);
}
void producer {
  mutex = wait (mutex);
  full = signal (full);
  empty = wait (empty);
  x++
  pf ("producer produces item");
    mutex = signal (mutex); }

void consumer () {
   mutex = wait (mutex)
   full = wait (full)
   pf ("consumer consumes item %d");
   x--;
   mutex = signal (mutex)
```



```
int main() {
  int choice;
  while (1) {
    pf ("In Produces In Consumer)
    exit
  {
  printf ("Enter choice")
  pf ("%d", & ch);
  switch (choice) {
  case (1)
  if (mutex == 1) && empty
    produce ();

  else
      ("Buffer is full");
      break;

case 2:
  if (mutex = 1) && (full = 0) {
      consume ();
  else
    pf ("Buffer is empty")
    break;
case 3:
    exit (0);
default {
    pf ("Invalid choice") }
    return 0;
}
```

Output:

(1) Producer
(2) Consumer
(3) exit
Enter your choice 1
producer produces ~~int~~ item 1

Enter ch 1
produces produces item 2

Enter ch 1
producer produces item 3

Enter your choice : 1
Buffer is full

Enter your choice : 2
Consumes Consumes item 3

Enter your choice : 2
consumes consumes item 2.

Program-6

Question:

Write a C program to simulate the concept of Dining Philosophers problem.

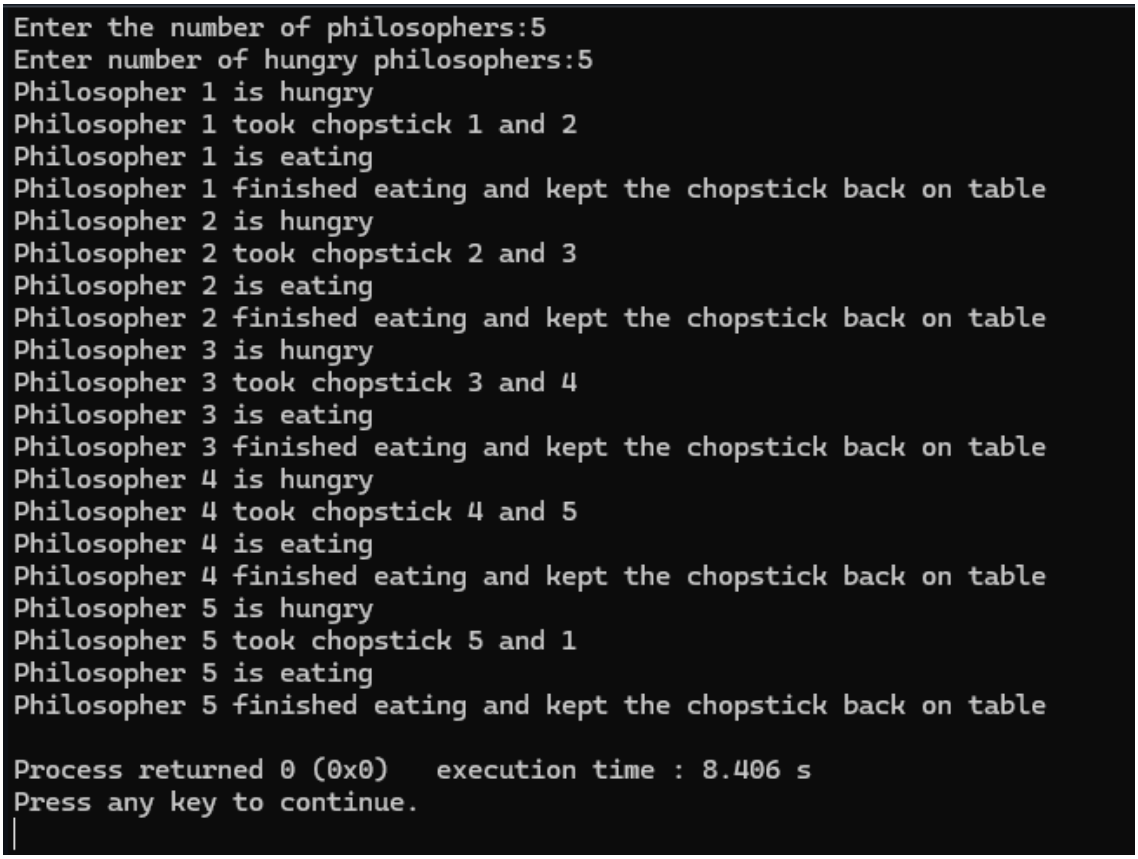
=>DINING-PHILOSOPHER

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

int n;
sem_t *chopstick;
pthread_t *philosopher;
int *phil_ids;

void* philosopher_fn(void* num) {
    int i = *(int*)num;
    int left = i;
    int right = (i + 1) % n;
    printf("Philosopher %d is hungry\n", i + 1);
    sem_wait(&chopstick[left]);
    sem_wait(&chopstick[right]);
    printf("Philosopher %d took chopstick %d and %d\n", i + 1, left + 1, right + 1);
    printf("Philosopher %d is eating\n", i + 1);
    sleep(1);
```

```
    printf("Philosopher %d finished eating and kept the chopstick back on table\n", i
+ 1);
    sem_post(&chopstick[left]);
    sem_post(&chopstick[right]);
    return NULL;
}

int main() {
    int hungry, i;
    printf("Enter the number of philosophers:");
    scanf("%d", &n);
    chopstick = malloc(n * sizeof(sem_t));
    philosopher = malloc(n * sizeof(pthread_t));
    phil_ids = malloc(n * sizeof(int));
    for (i = 0; i < n; i++) sem_init(&chopstick[i], 0, 1);
    printf("Enter number of hungry philosophers:");
    scanf("%d", &hungry);
    for (i = 0; i < hungry; i++) {
        phil_ids[i] = i;
        pthread_create(&philosopher[i], NULL, philosopher_fn, &phil_ids[i]);
        sleep(1);
    }
    for (i = 0; i < hungry; i++) {
        pthread_join(philosopher[i], NULL);
    }
    free(chopstick);
    free(philosopher);
    free(phil_ids);
    return 0;
```

}

Result:


Enter the number of philosophers:5
Enter number of hungry philosophers:5
Philosopher 1 is hungry
Philosopher 1 took chopstick 1 and 2
Philosopher 1 is eating
Philosopher 1 finished eating and kept the chopstick back on table
Philosopher 2 is hungry
Philosopher 2 took chopstick 2 and 3
Philosopher 2 is eating
Philosopher 2 finished eating and kept the chopstick back on table
Philosopher 3 is hungry
Philosopher 3 took chopstick 3 and 4
Philosopher 3 is eating
Philosopher 3 finished eating and kept the chopstick back on table
Philosopher 4 is hungry
Philosopher 4 took chopstick 4 and 5
Philosopher 4 is eating
Philosopher 4 finished eating and kept the chopstick back on table
Philosopher 5 is hungry
Philosopher 5 took chopstick 5 and 1
Philosopher 5 is eating
Philosopher 5 finished eating and kept the chopstick back on table
Process returned 0 (0x0) execution time : 8.406 s
Press any key to continue.

Dining Philosophers

```
#include <stdio.h>
#include <stdlib.h>
#define max 10;

int totalphilosopher;
int hungry[MAX];
int are_Neighbours(int a, int b) {
  return (abs(a-b) == 1 || abs(a-b) == totalphil-1);
}
void option(int count) {
  pf("How are philosopher to eat at any time \n");
  for(int i = 0; i < count; i++) {
    printf("%d is granted to eat\n", hungry[i]);
    for(int j = 0; j < count; j++) {
      if(j == i) {
        printf("P%d is waiting\n", hungry[j]);
      }}}}

void options(int count) {
  printf("\n 1) How two philosopher to eat some time\n");
  int combination = 1;
  for(i = 0; i < count; i++)
    for(int j = (i+1); j < count; j++) {
```

```
if(!are_neighbor(hungry[i], hungry[j])) {
  pf("Comb. %d\n", combination);
  pf("%d & P%d are granted to eat\n", hungry[i], hungry[j]);
  for(int k = 0; k < count; k++) {
    if(k != i && k != j) {
      printf("P%d is waiting\n", hungry[k]);
    }}
  printf("\n");}}
if(combination == 1) {
  pf("no comb. found where two non-neighbors eat\n");
}}

int main() {
  int hungry_count;
  printf("Enter the total no. of philosophers");
  scanf("%d", &totalphilosophers);
  pf("How many are hungry");
  do {
    pf("one can eat at a time)
    2) two can eat at a time
    e) exit
```

38

```
printf("Enter your choice");
scanf("%d", &choice)
switch(choice) {
case 1:
  option(hungry[j] - count);
  breaks;
case 2:
  option(hungry count)
  break;
case e:
  printf("Exiting\n");
  break;
default:
  printf("Invalid choice\n");
}
while(choice != 3);
return 0;
}
```

Dining philosophers producer

Enter total no of philosopher

Enter philosophers 1 position : 2
2 position : 4
3 position : 5

(1) One can eat at a time
(2) Two can eat at a time
Enter your choice : 1
Allow one philosopher
P2 is waiting
P4

39

P5
P2
P2 is granted
P2 has finished
P3 is waiting
P4
P5
P4 is granted to eat
P4 is finished eating
P2 is waiting
P4
P5
P5 is granted to eat.
P5 has finished eating.

Program-7

Question:

Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

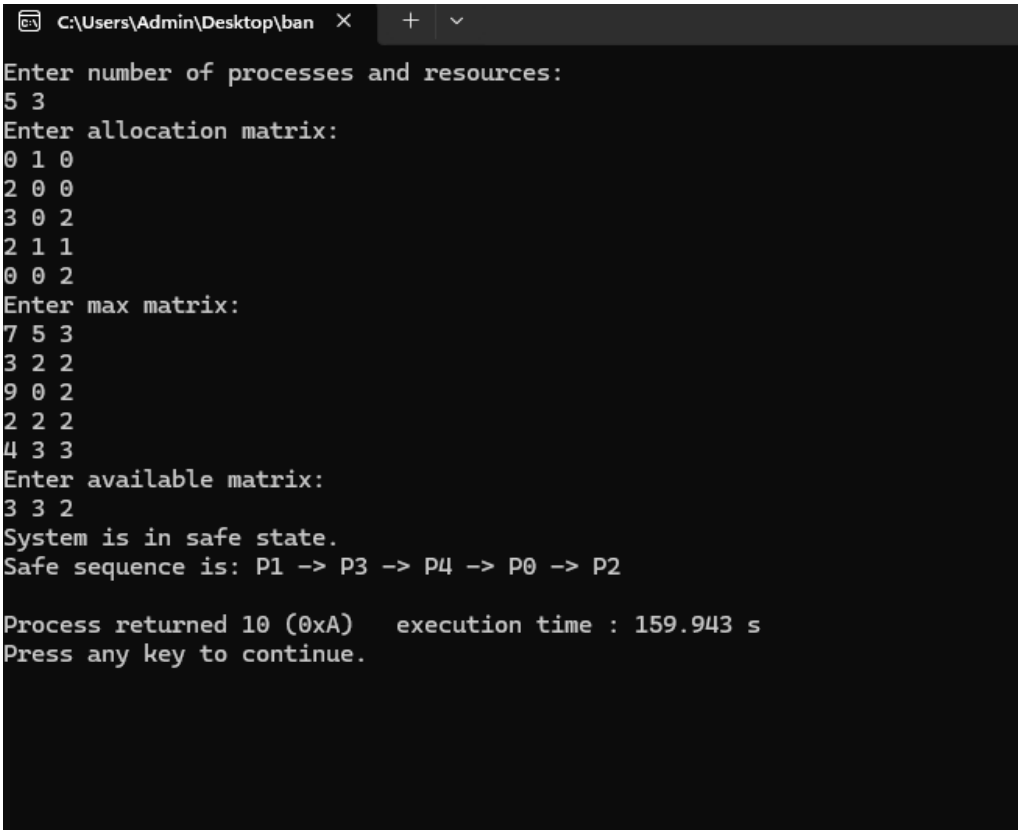
=>BANKERS ALGORITHM

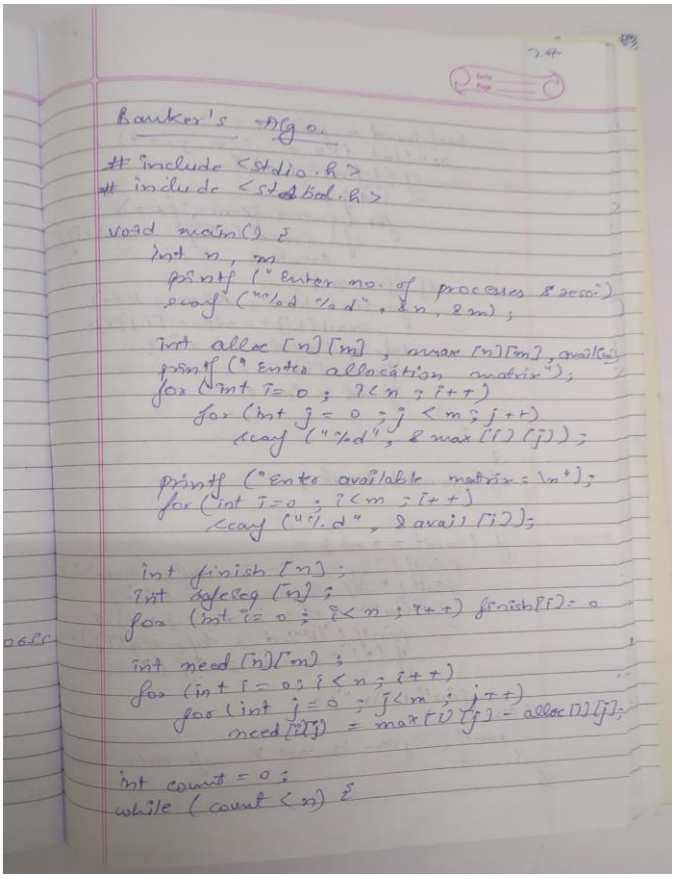
```
#include <stdio.h>
#include <stdbool.h>

void main() {
    int n, m;
    printf("Enter number of processes and resources:\n");
    scanf("%d %d", &n, &m);

    int alloc[n][m], max[n][m], avail[m];
    printf("Enter allocation matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);

    printf("Enter max matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &max[i][j]);

    printf("Enter available matrix:\n");
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);
```

```
int finish[n];
int safeSeq[n];
for (int i = 0; i < n; i++) finish[i] = 0;

int need[n][m];
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        need[i][j] = max[i][j] - alloc[i][j];
int count = 0;
while (count < n) {
    bool found = false;
    for (int i = 0; i < n; i++) {
        if (!finish[i]) {
            int j;
            for (j = 0; j < m; j++)
                if (need[i][j] > avail[j])
                    break;
            if (j == m) {
                for (int k = 0; k < m; k++)
                    avail[k] += alloc[i][k];
                safeSeq[count++] = i;
                finish[i] = 1;
                found = true;
            }
        }
    }
    if (!found) break;
}
if (count == n) {
```

```
        printf("System is in safe state.\n");
        printf("Safe sequence is: ");
        for (int i = 0; i < n; i++) {
            printf("P%d", safeSeq[i]);
            if (i != n - 1) printf(" -> ");
        }
        printf("\n");
    } else {
        printf("System is not in a safe state.\n");
    }
}
```

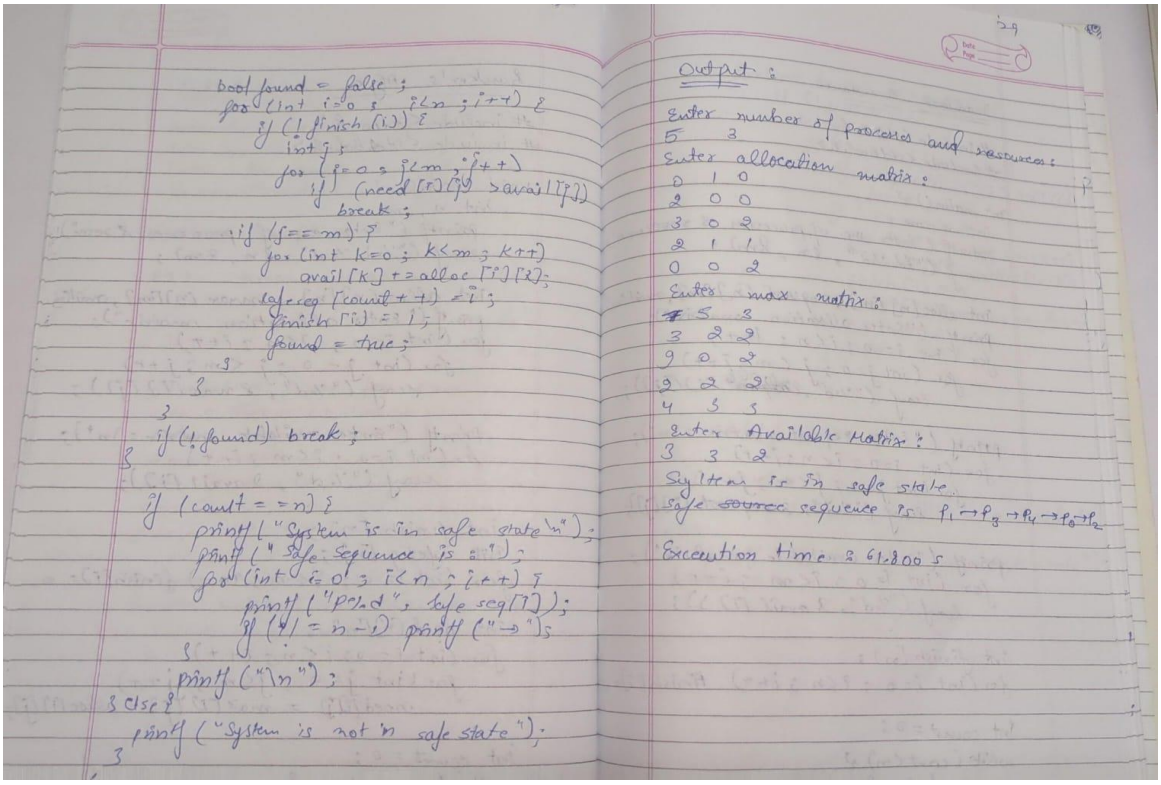
Result:





Banker's Algo.

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

void main() {
    int n, m
    printf ("Enter no. of processes & resources");
    scanf ("%d %d", &n, &m);

    int alloc [n] [m], max [n] [m], avail[m];
    printf ("Enter allocation matrix");
    for (int i = 0 ; i<n ; i++)
        for (int j = 0 ; j < m ; j++)
            scanf ("%d", &max [i] [j]);

    printf ("Enter available matrix = \n");
    for (int i=0 ; i<m ; i++)
        scanf ("%d", &avail [i]);

    int finish [n];
    int safeseq [n];
    for (int i = 0 ; i<n ; i++) finish[i] = 0

    int need [n][m];
    for (int i = 0; i<n ; i++)
        for (int j = 0 ; j<m ; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    int count = 0;
    while (count < n) {
```



```
        bool found = false;
        for (int i=0 ; i<n ; i++) {
            if (!finish [i]) {
                int j;
                for (j=0 ; j<m ; j++)
                    if (need [i][j] > avail[j])
                        break;
                if (j == m) {
                    for (int k=0 ; k<m ; k++)
                        avail [k] += alloc [i] [k];
                    safeseq [count++] = i;
                    finish [i] = 1;
                    found = true;
                }
            }
        }
        if (!found) break;
    }

    if (count == n) {
        printf ("System is in safe state\n");
        printf ("Safe sequence is : ");
        for (int i=0 ; i<n ; i++) {
            printf ("P%d", safe seq[i]);
            if (i != n-1) printf ("→ ");
        }
        printf ("\n");
    } else {
        printf ("System is not in safe state");
    }
}
```

Output :

```
Enter number of processes and resources:
5 3
Enter allocation matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter max matrix :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter Available Matrix :
3 3 2
System is in safe state
Safe sequence is : P1 → P3 → P4 → P0 → P2

Execution time : 61.800 s
```

Program-8

Question:

Write a C program to simulate deadlock detection

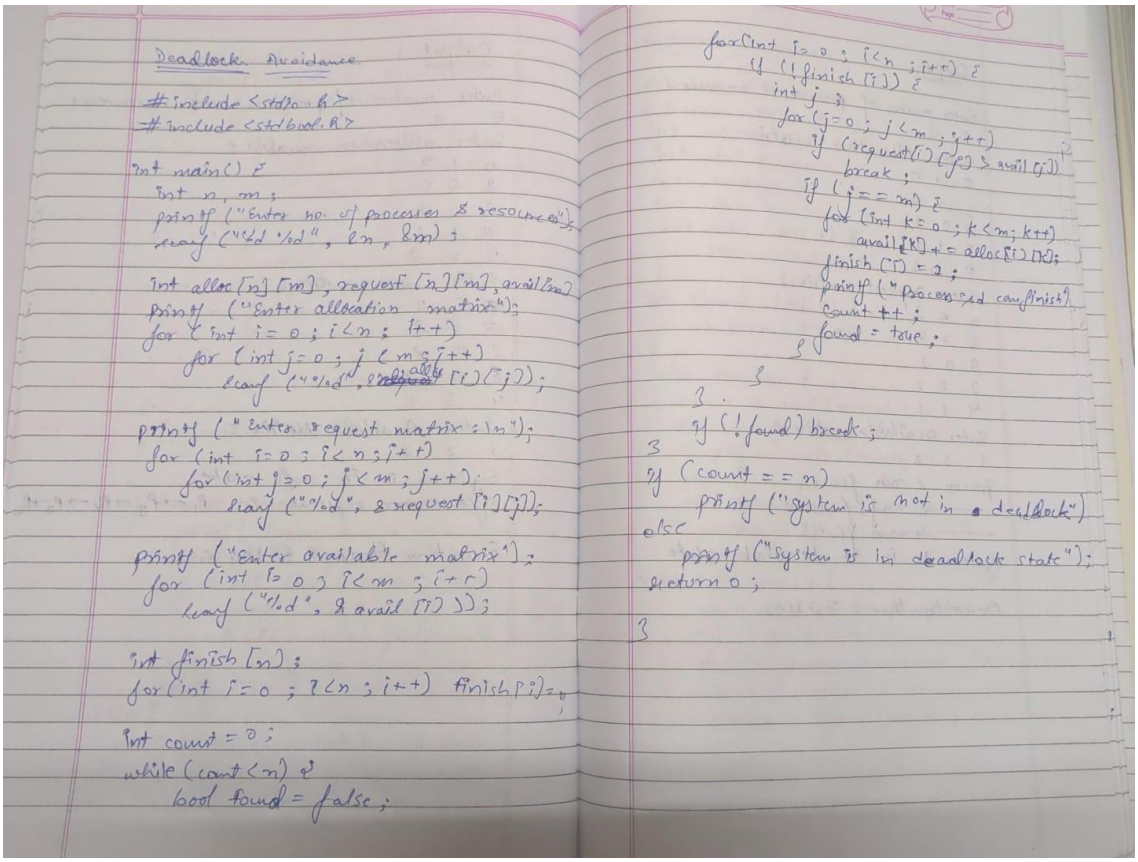
=>DEADLOCK

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

int main() {
    int n, m;
    printf("Enter number of processes and resources:\n");
    scanf("%d %d", &n, &m);
    int alloc[n][m], request[n][m], avail[m];
    printf("Enter allocation matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);
    printf("Enter request matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &request[i][j]);
    printf("Enter available matrix:\n");
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);
    int finish[n];
    for (int i = 0; i < n; i++) finish[i] = 0;
    int count = 0;
    while (count < n) {
```

```
        bool found = false;
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < m; j++)
                    if (request[i][j] > avail[j])
                        break;
                if (j == m) {
                    for (int k = 0; k < m; k++)
                        avail[k] += alloc[i][k];
                    finish[i] = 1;
                    printf("Process %d can finish.\n", i);
                    count++;
                    found = true;
                }
            }
        }
        if (!found) break;
    }
    if (count == n)
        printf("System is not in a deadlock state.\n");
    else
        printf("System is in a deadlock state.\n");
    return 0;
}
```

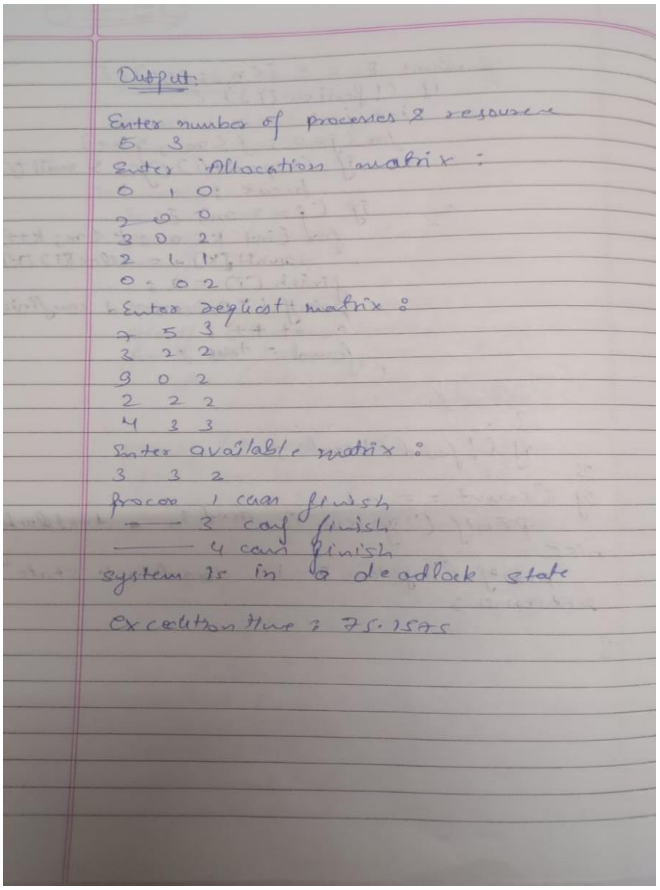
Result:

```
C:\Users\Admin\Desktop\dea
Enter number of processes and resources:
5 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter request matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available matrix:
3 3 2
Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Process returned 0 (0x0)   execution time : 75.157 s
Press any key to continue.
```



Deadlock Avoidance

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

int main() {
    int n, m;
    printf("Enter no. of processes & resources");
    scanf("%d %d", &n, &m);

    int alloc[n][m], request[n][m], avail[m];
    printf("Enter allocation matrix");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);

    printf("Enter request matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &request[i][j]);

    printf("Enter available matrix");
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    int finish[n];
    for (int i = 0; i < n; i++) finish[i] = 0;

    int count = 0;
    while (count < n) {
        bool found = false;
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < m; j++)
                    if (request[i][j] > avail[j])
                        break;
                if (j == m) {
                    for (int k = 0; k < m; k++)
                        avail[k] += alloc[i][k];
                    finish[i] = 1;
                    printf("Process %d can finish", i);
                    count++;
                    found = true;
                }
            }
        }
        if (!found) break;
    }
    if (count == n)
        printf("System is not in a deadlock");
    else
        printf("System is in deadlock state");
    return 0;
}
```



Program-9

Question:

Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit
-Best-fit
-First-fit

=>MEMORY ALLOCATION

```
#include <stdio.h>


struct Block {
    int size;
    int allocated;
};
```

```
struct File {
    int size;
    int block_no;
};

void resetBlocks(struct Block blocks[], int n) {
    for (int i = 0; i < n; i++) {
        blocks[i].allocated = 0;
    }
}

void firstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\n\tMemory Management Scheme – First Fit\n");
    printf("File_no:\tFile_size\tBlock_no:\tBlock_size:\n");

    for (int i = 0; i < n_files; i++) {
        files[i].block_no = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                files[i].block_no = j + 1;
                blocks[j].allocated = 1;
                printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, j + 1, blocks[j].size);
                break;
            }
        }
        if (files[i].block_no == -1) {
            printf("%d\t\t%d\t\t_\t\t_\n", i + 1, files[i].size);
        }
    }
```

```
}

void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\n\tMemory Management Scheme – Best Fit\n");
    printf("File_no:\tFile_size\tBlock_no:\tBlock_size:\n");

    for (int i = 0; i < n_files; i++) {
        int bestIdx = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                if (bestIdx == -1 || blocks[j].size < blocks[bestIdx].size) {
                    bestIdx = j;
                }
            }
        }
        if (bestIdx != -1) {
            blocks[bestIdx].allocated = 1;
            files[i].block_no = bestIdx + 1;
            printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, bestIdx + 1,
blocks[bestIdx].size);
        } else {
            printf("%d\t\t%d\t\t_\t\t_\n", i + 1, files[i].size);
        }
    }
}

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\n\tMemory Management Scheme – Worst Fit\n");
    printf("File_no:\tFile_size\tBlock_no:\tBlock_size:\n");
```

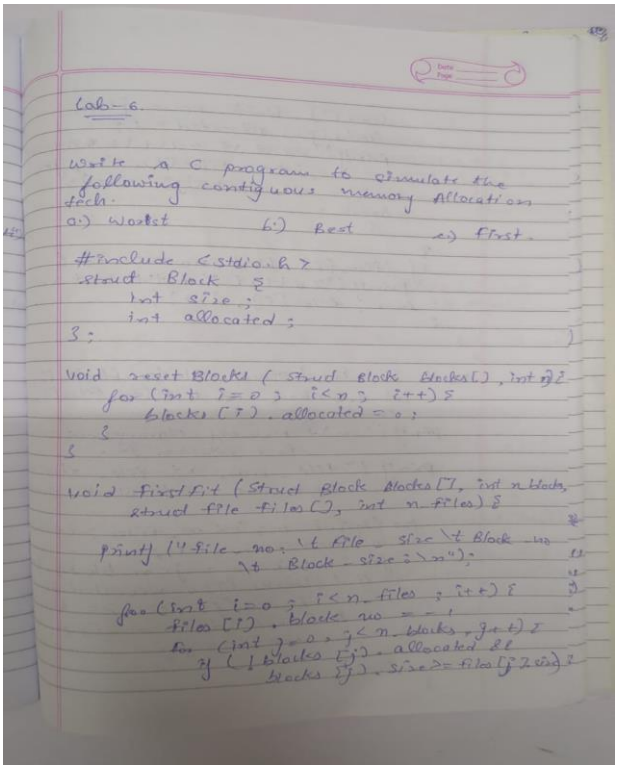
```
    for (int i = 0; i < n_files; i++) {
        int worstIdx = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                if (worstIdx == -1 || blocks[j].size > blocks[worstIdx].size) {
                    worstIdx = j;
                }
            }
        }
        if (worstIdx != -1) {
            blocks[worstIdx].allocated = 1;
            files[i].block_no = worstIdx + 1;
            printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, worstIdx + 1,
blocks[worstIdx].size);
        } else {
            printf("%d\t\t%d\t\t_\t\t_\n", i + 1, files[i].size);
        }
    }
}

int main() {
    int n_blocks, n_files, choice;
    printf("Memory Management Scheme\n");
    printf("Enter the number of blocks: ");
    scanf("%d", &n_blocks);
    printf("Enter the number of files: ");
    scanf("%d", &n_files);
    struct Block blocks[n_blocks];
    struct File files[n_files];
```

```
printf("\nEnter the size of the blocks:\n");
for (int i = 0; i < n_blocks; i++) {
    printf("Block %d: ", i + 1);
    scanf("%d", &blocks[i].size);
    blocks[i].allocated = 0;
}
printf("Enter the size of the files:\n");
for (int i = 0; i < n_files; i++) {
    printf("File %d: ", i + 1);
    scanf("%d", &files[i].size);
}

do {
    printf("\n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    resetBlocks(blocks, n_blocks); // Reset block allocation before each strategy
    switch (choice) {
        case 1:
            firstFit(blocks, n_blocks, files, n_files);
            break;
        case 2:
            bestFit(blocks, n_blocks, files, n_files);
            break;
        case 3:
            worstFit(blocks, n_blocks, files, n_files);
            break;
        case 4:
            printf("\nExiting...\n");
```

```
            break;
        default:
            printf("Invalid choice.\n");
    }
} while (choice != 4);

return 0;
}
```

Result:

```
Memory Management Scheme
Enter the number of blocks: 5
Enter the number of files: 4

Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 420

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1

        Memory Management Scheme û First Fit
File_no:        File_size       Block_no:       Block_size:
1               212             2               500
2               417             5               600
3               112             3               200
4               420             _               _

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2

        Memory Management Scheme û Best Fit
File_no:        File_size       Block_no:       Block_size:
1               212             4               300
2               417             2               500
3               112             3               200
4               420             5               600

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 3

        Memory Management Scheme û Worst Fit
File_no:        File_size       Block_no:       Block_size:
1               212             5               600
2               417             2               500
3               112             4               300
4               420             _               _

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice:
```

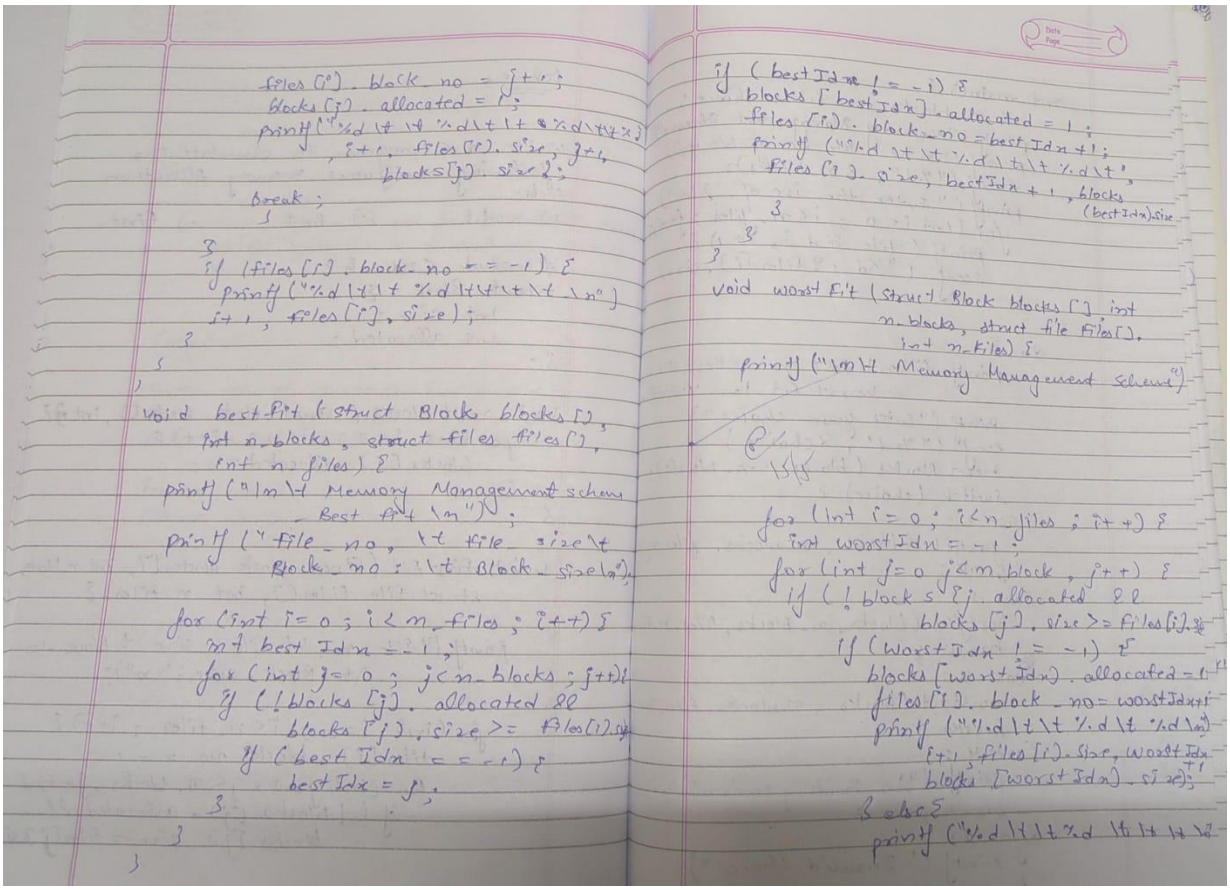


Lab-6.

Write a C program to simulate the following contiguous memory Allocation tech:

a.) Worst b.) Best c.) First.

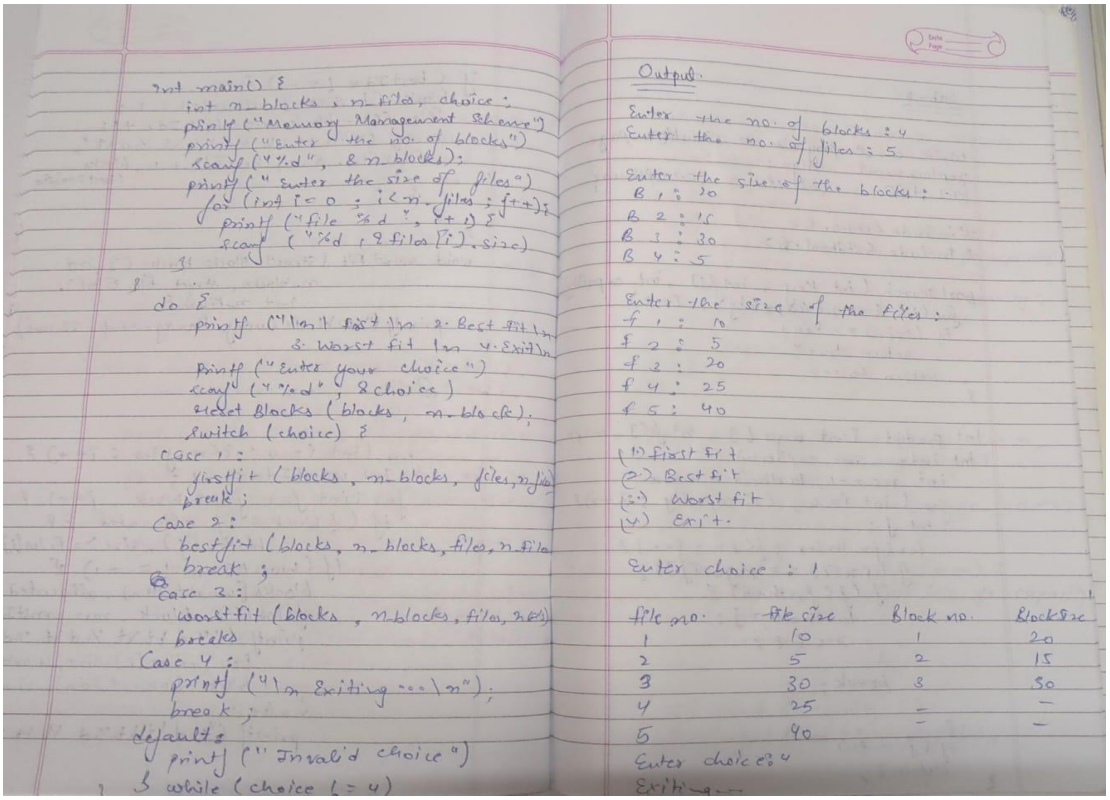
```
#include <stdio.h>
struct Block {
    int size;
    int allocated;
};

void resetBlocks(struct Block blocks[], int n) {
    for (int i = 0; i < n; i++) {
        blocks[i].allocated = 0;
    }
}

void firstFit(struct Block blocks[], int n_blocks,
              struct file files[], int n_files) {

    printf("File_no:\tFile_size\tBlock_no:
            \tBlock_size:\n");

    for (int i = 0; i < n_files; i++) {
        files[i].block_no = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (!blocks[j].allocated &&
                blocks[j].size >= files[i].size) {
```



```
                files[i].block_no = j+1;
                blocks[j].allocated = 1;
                printf("%d\t\t%d\t\t%d\t\t%d\n",
                       i+1, files[i].size, j+1,
                       blocks[j].size);
                break;
            }
        }
        if (files[i].block_no == -1) {
            printf("%d\t\t%d\t\tNot Allocated\n",
                   i+1, files[i].size);
        }
    }
}

void bestFit(struct Block blocks[],
             int n_blocks, struct file files[],
             int n_files) {
    printf("\n\t\tMemory Management scheme
            - Best Fit\n");
    printf("File_no\tFile_size\t
            Block_no:\tBlock_size\n");

    for (int i = 0; i < n_files; i++) {
        int bestIdx = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (!blocks[j].allocated &&
                blocks[j].size >= files[i].size) {
                if (bestIdx == -1) {
                    bestIdx = j;
                }
            }
        }
        if (bestIdx != -1) {
            blocks[bestIdx].allocated = 1;
            files[i].block_no = bestIdx+1;
            printf("%d\t\t%d\t\t%d\t\t%d\n",
                   i+1, files[i].size, bestIdx+1,
                   blocks[bestIdx].size);
        }
    }
}

void worstFit(struct Block blocks[], int
              n_blocks, struct file files[],
              int n_files) {
    printf("\n\t\tMemory Management scheme");
```



```
    for (int i = 0; i < n_files; i++) {
        int worstIdx = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (!blocks[j].allocated &&
                blocks[j].size >= files[i].size) {
                if (worstIdx != -1) {
                    blocks[worstIdx].allocated = 1;
                    files[i].block_no = worstIdx+1;
                    printf("%d\t\t%d\t\t%d\t\t%d\n",
                           i+1, files[i].size, worstIdx+1,
                           blocks[worstIdx].size);
                } else {
                    printf("%d\t\t%d\t\tNot Allocated\n",
```



Program-10

Question:

Write a C program to simulate page replacement algorithms a) FIFO
-LRU
-Optimal

=>LRU

```
#include <stdio.h>

int main() {
    int n, frames, i, j, k, faults = 0;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    int pages[n];
    printf("Enter the reference string: ");
```

```
for(i = 0; i < n; i++)
    scanf("%d", &pages[i]);
printf("Enter number of frames: ");
scanf("%d", &frames);
int frame_arr[frames];
int time[frames];
for(i = 0; i < frames; i++) {
    frame_arr[i] = -1;
    time[i] = 0;
}

int counter = 0;
for(i = 0; i < n; i++) {
    int flag = 0;
    for(j = 0; j < frames; j++) {
        if(frame_arr[j] == pages[i]) {
            flag = 1;
            counter++;
            time[j] = counter;
            break;
        }
    }
    if(flag == 0) {
        faults++;
        int min_time = time[0], min_pos = 0;
        for(k = 1; k < frames; k++) {
            if(time[k] < min_time) {
                min_time = time[k];
                min_pos = k;
```

```
            }
        }
        frame_arr[min_pos] = pages[i];
        counter++;
        time[min_pos] = counter;
    }
    printf("Frames after accessing %d: ", pages[i]);
    for(j = 0; j < frames; j++) {
        if(frame_arr[j] == -1)
            printf("- ");
        else
            printf("%d ", frame_arr[j]);
    }
    printf("\n");
  }
  printf("Total page faults: %d\n", faults);
  int Hits = n-faults;
  printf("Total page Hits: %d\n",Hits);
  return 0;
}
```

Result:

```
Enter number of pages: 20
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames: 4
Frames after accessing 7: 7 - - -
Frames after accessing 0: 7 0 - -
Frames after accessing 1: 7 0 1 -
Frames after accessing 2: 7 0 1 2
Frames after accessing 0: 7 0 1 2
Frames after accessing 3: 3 0 1 2
Frames after accessing 0: 3 0 1 2
Frames after accessing 4: 3 0 4 2
Frames after accessing 2: 3 0 4 2
Frames after accessing 3: 3 0 4 2
Frames after accessing 0: 3 0 4 2
Frames after accessing 3: 3 0 4 2
Frames after accessing 2: 3 0 4 2
Frames after accessing 1: 3 0 1 2
Frames after accessing 2: 3 0 1 2
Frames after accessing 0: 3 0 1 2
Frames after accessing 1: 3 0 1 2
Frames after accessing 7: 7 0 1 2
Frames after accessing 0: 7 0 1 2
Frames after accessing 1: 7 0 1 2
Total page faults: 8
Total page Hits: 12

Process returned 0 (0x0)   execution time : 54.602 s
Press any key to continue.
```

=>OPTIMAL

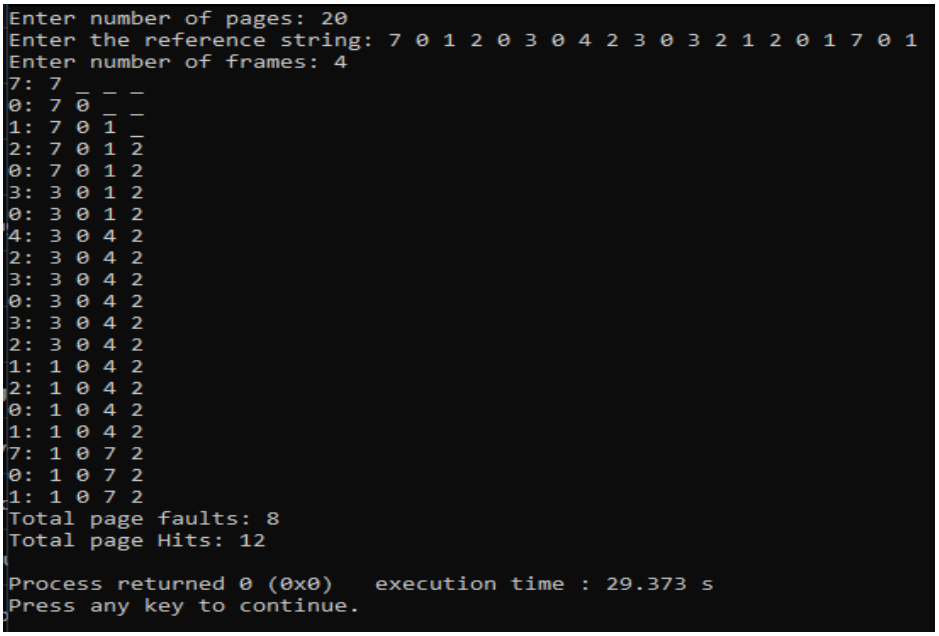
```
#include <stdio.h>

int main() {
    int n, frames, i, j, k, faults = 0;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    int pages[n];
```

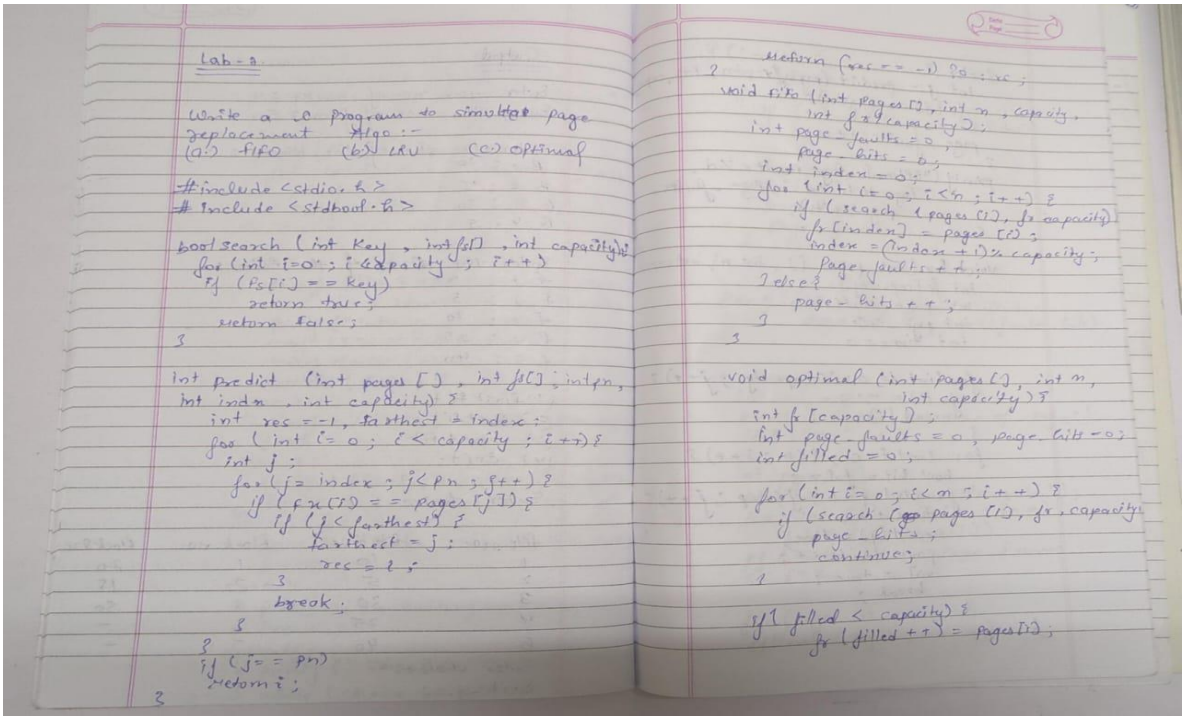
```
printf("Enter the reference string: ");
for(i = 0; i < n; i++)
   scanf("%d", &pages[i]);
printf("Enter number of frames: ");
scanf("%d", &frames);
int frame_arr[frames];
for(i = 0; i < frames; i++)
   frame_arr[i] = -1;
for(i = 0; i < n; i++) {
   int flag = 0;
   for(j = 0; j < frames; j++) {
      if(frame_arr[j] == pages[i]) {
         flag = 1;
         break;
      }
   }
   if(flag == 0) {
      faults++;
      int pos = -1;
      for(j = 0; j < frames; j++) {
         if(frame_arr[j] == -1) {
            pos = j;
            break;
         }
      }
      if(pos == -1) {
         int farthest = i, replace_index = 0;
         for(j = 0; j < frames; j++) {
            int found = 0;
```

```
            for(k = i + 1; k < n; k++) {
                if(frame_arr[j] == pages[k]) {
                    if(k > farthest) {
                        farthest = k;
                        replace_index = j;
                    }
                    found = 1;
                    break;
                }
            }
            if(!found) {
                replace_index = j;
                break;
            }
        }
        pos = replace_index;
    }
    frame_arr[pos] = pages[i];
  }
  printf("%d: ", pages[i]);
  for(j = 0; j < frames; j++) {
    if(frame_arr[j] == -1)
      printf("_ ");
    else
      printf("%d ", frame_arr[j]);
  }
  printf("\n");
}
printf("Total page faults: %d\n", faults);
```

```
    int Hits = n-faults;
    printf("Total page Hits: %d\n",Hits);
    return 0;
}
```

Result:

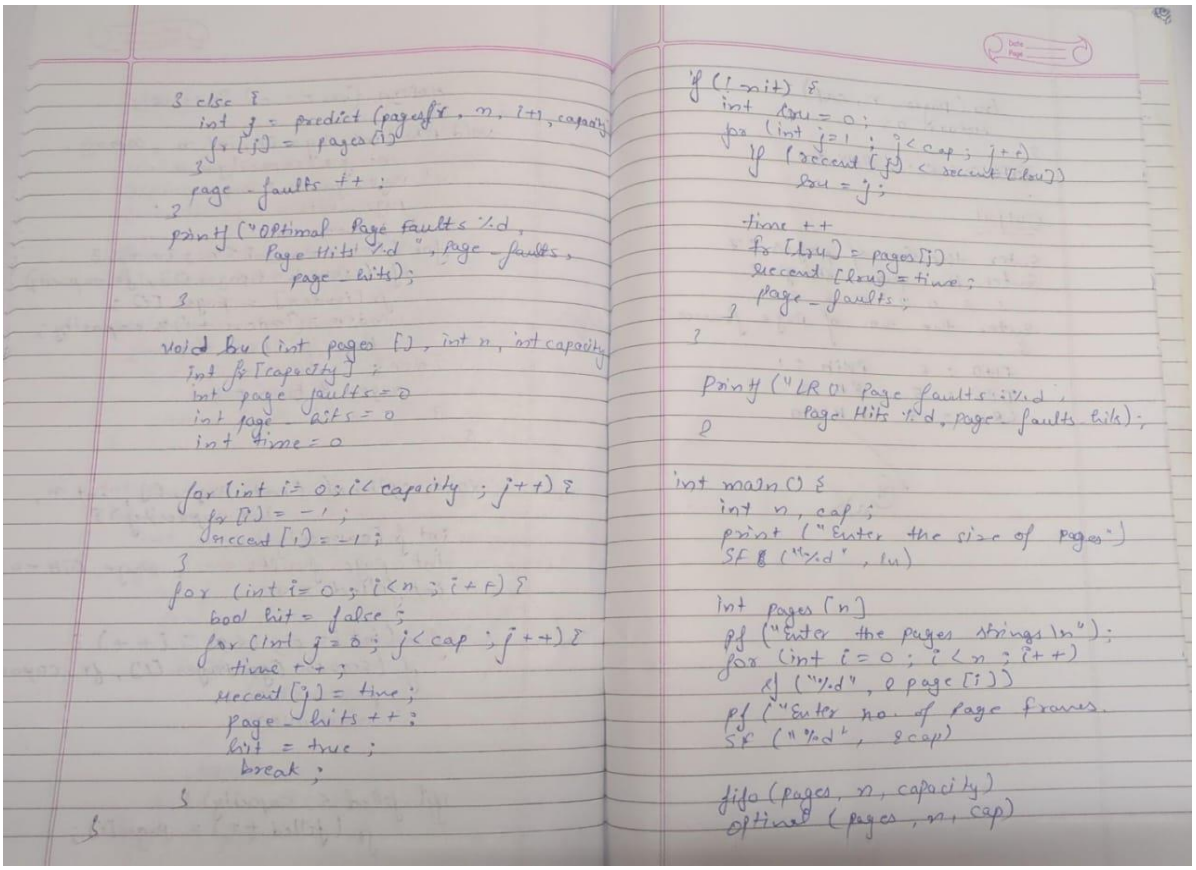




Lab - 2

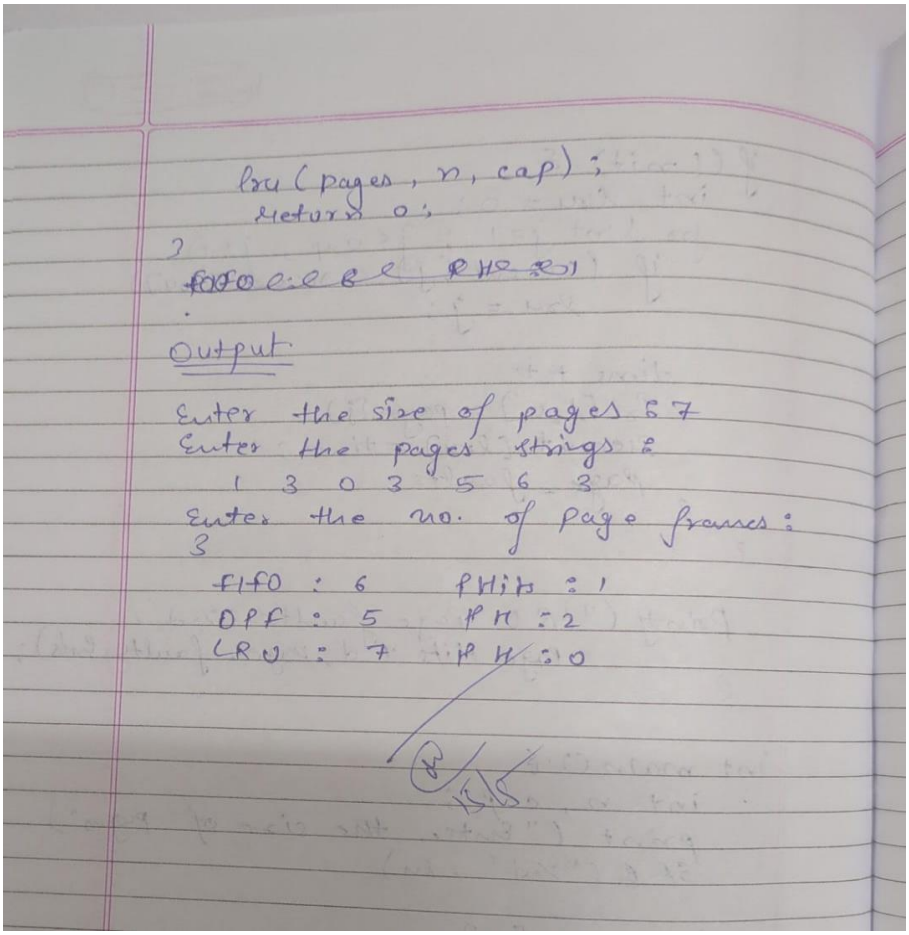
Write a C program to simulate page replacement Algo :-
(a) FIFO (b) LRU (c) Optimal

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

bool search (int key, int fs[], int capacity){
  for (int i=0; i<capacity; i++)
    if (fs[i] == key)
      return true;
  return false;
}

int predict (int pages[], int fs[], int pn,
  int indn, int capacity) {
    int res = -1, farthest = indn;
    for (int i = 0; i < capacity; i++) {
      int j;
      for (j = indn; j<pn; j++) {
        if (fs[i] == pages[j]) {
          if (j > farthest) {
            farthest = j;
            res = i;
          }
          break;
        }
      }
      if (j == pn)
        return i;
    }
    return (res == -1) ? 0 : res;
}
void fifo (int pages[], int n, capacity,
    int fs[capacity]);
  int page_faults = 0,
  page_hits = 0;
  int indexn = 0;
  for (int i=0; i<n; i++) {
    if (search (pages[i], fs, capacity)
      fs[indexn] = pages[i];
      indexn = (indexn + 1) % capacity;
      page_faults++;
    } else {
      page_hits++;
    }
  }

void optimal (int pages[], int n,
      int capacity) {
  int fs[capacity];
  int page_faults = 0, page_hits = 0;
  int filled = 0;

  for (int i=0; i<n; i++) {
    if (search (pages[i], fs, capacity)
      page_hits++;
      continue;
    }

  if (filled < capacity) {
      fs[filled++] = pages[i];
```



```
  } else {
    int j = predict (pages, fs, n, i+1, capacity
    fs[j] = pages[i];
  }
  page_faults++;
}
printf ("Optimal Page faults %d,
    Page Hits %d ", page_faults,
      page_hits);
}

void lru (int pages[], int n, int capacity
  int fs[capacity];
  int page_faults = 0
  int page_hits = 0
  int time = 0

  for (int i=0; i<capacity; i++) {
    fs[i] = -1;
    recent[i] = -1;
  }
  for (int i=0; i<n; i++) {
    bool hit = false;
    for (int j=0; j<cap; j++) {
      time++;
      recent[j] = time;
      page_hits++;
      hit = true;
      break;
    }
  }
```

```
if (!hit) {
  int lru = 0;
  for (int j=1; j<cap; j++)
    if (recent[j] < recent[lru])
      lru = j;

  time++
  fs[lru] = pages[i]
  recent[lru] = time;
  page_faults;
}
}

printf ("LRU Page faults : %d,
    Page Hits %d, page_faults, hits);
}

int main() {
  int n, cap;
  print ("Enter the size of pages")
  sf ("%d", &n)

  int pages[n]
  pf ("Enter the pages strings \n");
  for (int i = 0; i < n; i++)
    sf ("%d", &page[i])
  pf ("Enter no. of page frames.
  sf ("%d", &cap)

  fifo (pages, n, capacity)
  optimal (pages, n, cap)
```



```
lru(pages, n, cap);
return 0;
}
```

Output

```
Enter the size of pages 8 7
Enter the pages strings :
1 3 0 3 5 6 3
Enter the no. of page frames :
3
FIFO : 6    PHit : 1
OPT : 5     PH : 2
LRU : 7     PH : 0
```