VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

OPERATING SYSTEMS

Submitted by

TARUN S SUNADOLI (1WA23CS015)

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 **TARUN S SUNADOLI (1WA23CS015)**, 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.

Dr. Seema Patil
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

10.	Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal	89-100

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 CT(int processes[], int n, int bt[], int at[], int ct[]) {
    ct[0] = at[0] + bt[0];
    for (int i = 1; i < n; i++) {
        ct[i] = (ct[i - 1] > at[i] ? ct[i - 1] : at[i]) + bt[i];
    }
}

void TAT(int processes[], int n, int bt[], int at[], int ct[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = ct[i] - at[i];
    }
}

void WT(int processes[], int n, int bt[], int at[], int tat[], int wt[]) {
    for (int i = 0; i < n; i++) {
        wt[i] = tat[i] - bt[i];
    }
```

```
}

void AVG(int processes[], int n, int bt[], int at[]) {
    int wt[n], tat[n], ct[n];
    CT(processes, n, bt, at, ct);
    TAT(processes, n, bt, at, ct, tat);
    WT(processes, n, bt, at, tat, wt);

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

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

int main() {
    int n;

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

    int processes[n], bt[n], at[n];

    printf("enter bt and tat: \n");
    for (int i = 0; i < n; i++) {
```

```
        printf("bt %d: ", i + 1);
        scanf("%d", &bt[i]);
        printf("tat %d: ", i + 1);
        scanf("%d", &at[i]);
        processes[i] = i + 1;
    }

    AVG(processes, n, bt, at);

    return 0;
}
```

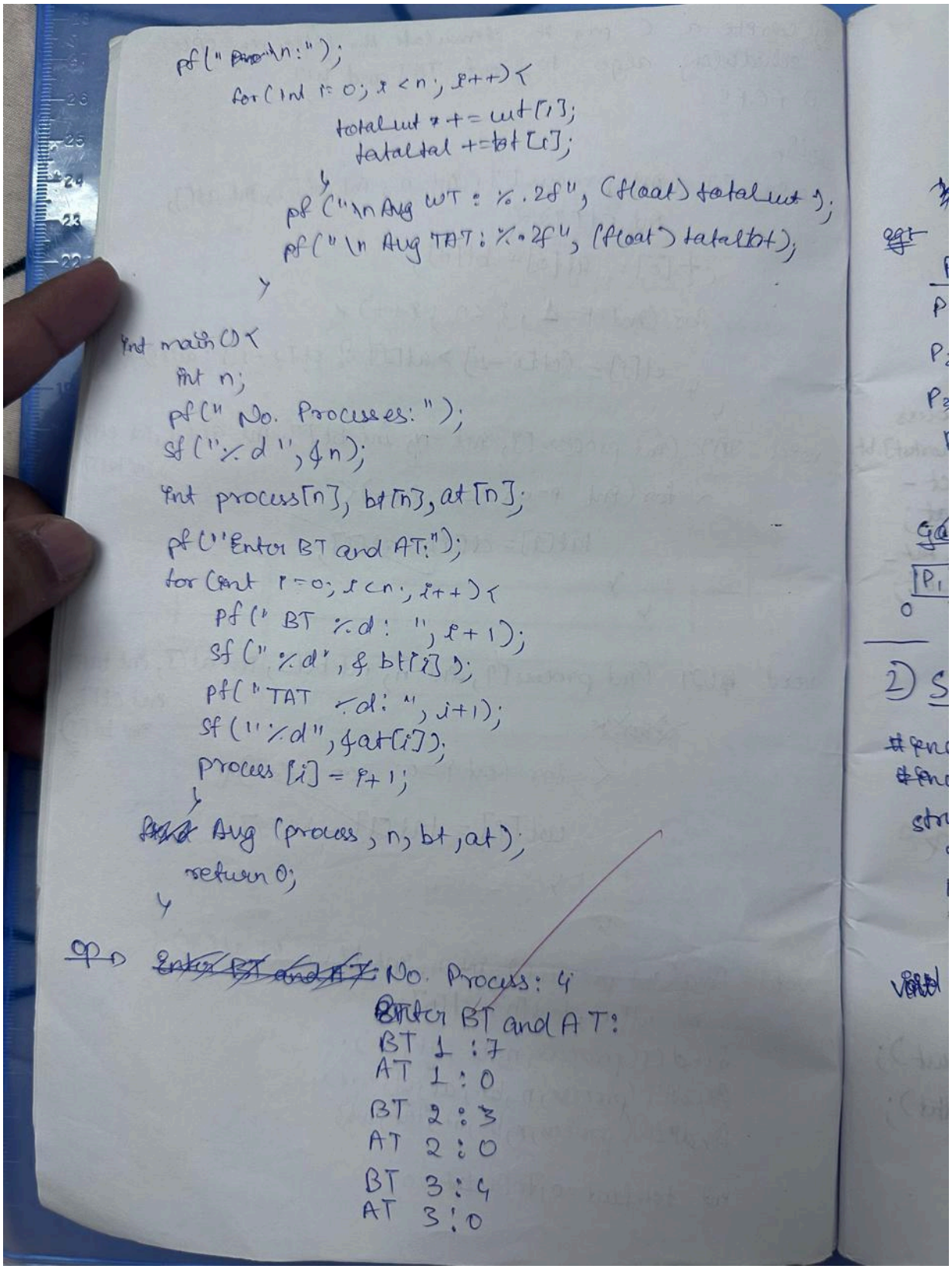
```
no. of processes: 4
enter bt and tat:
bt 1: 7
tat 1: 0
bt 2: 3
tat 2: 0
bt 3: 4
tat 3: 0
bt 4: 6
tat 4: 0

avg bt= 7.75
avg tat = 12.75
```

Q] write a C prg to stimulate the following CPU scheduling algo to find TAT and WT.

1) FCFS

sol^n

```
void CT (int process[], int n, int bt[], int at[],
         int ct[]){
    ct[0] = at[0] + bt[0];
    for (int i=1; i<n; i++){
        ct[i] = (ct[i-1] > at[i] ? ct[i-1] : at[i]) + bt[i];
    }
}
void TAT (int process[], int n, int bt[], int at[], int ct[],
          int tat[]){
    for(int i=0; i<n; i++){
        tat[i] = ct[i] - at[i];
    }
}
void WT (int process[], int n, int bt[], int at[], int tat[],
         int ct[], int bt[]){
    for (int i=0; i<n; i++){
        wt[i] = tat[i] - bt[i];
    }
}
void Avg (int process[], int n, int bt[], int at[]){
    int wt[n], tat[n], ct[n];
    findCT(process, n, bt, at, ct);
    findTAT(process, n, bt, at, ct, tat);
    findWT(process, n, bt, at, tat, wt);
    int totalwt=0, totaltat=0;
```



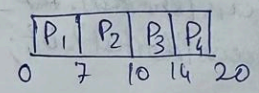
```
pf("Enter n:");
    for(int i=0; i<n; i++){
        total_wt += wt[i];
        total_tat += bt[i];
    }
    pf("\n Avg WT : %.2f", (float) total_wt);
    pf("\n Avg TAT : %.2f", (float) total_bt);
}

int main(){
    int n;
    pf("No. Processes: ");
    sf("%d", &n);
    int process[n], bt[n], at[n];
    pf("Enter BT and AT:");
    for(int i=0; i<n; i++){
        pf("BT %d: ", i+1);
        sf("%d", &bt[i]);
        pf("TAT %d: ", i+1);
        sf("%d", &at[i]);
        process[i] = i+1;
    }
    Avg(process, n, bt, at);
    return 0;
}
```

Op→ Enter BT and AT No. Process: 4
Enter BT and AT:
BT 1 : 7
AT 1 : 0
BT 2 : 3
AT 2 : 0
BT 3 : 4
AT 3 : 0

BT 4 : 6
AT 4 : 0

Avg WT : 7.75
Avg TAT : 12.75

P.	AT	BT	CT	TAT	WT
P_1	0	7	7	7	0
P_2	0	3	10	10	7
P_3	0	4	14	14	10
P_4	0	6	20	20	14
				Avg = 12.75	7.75



=>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++;
```

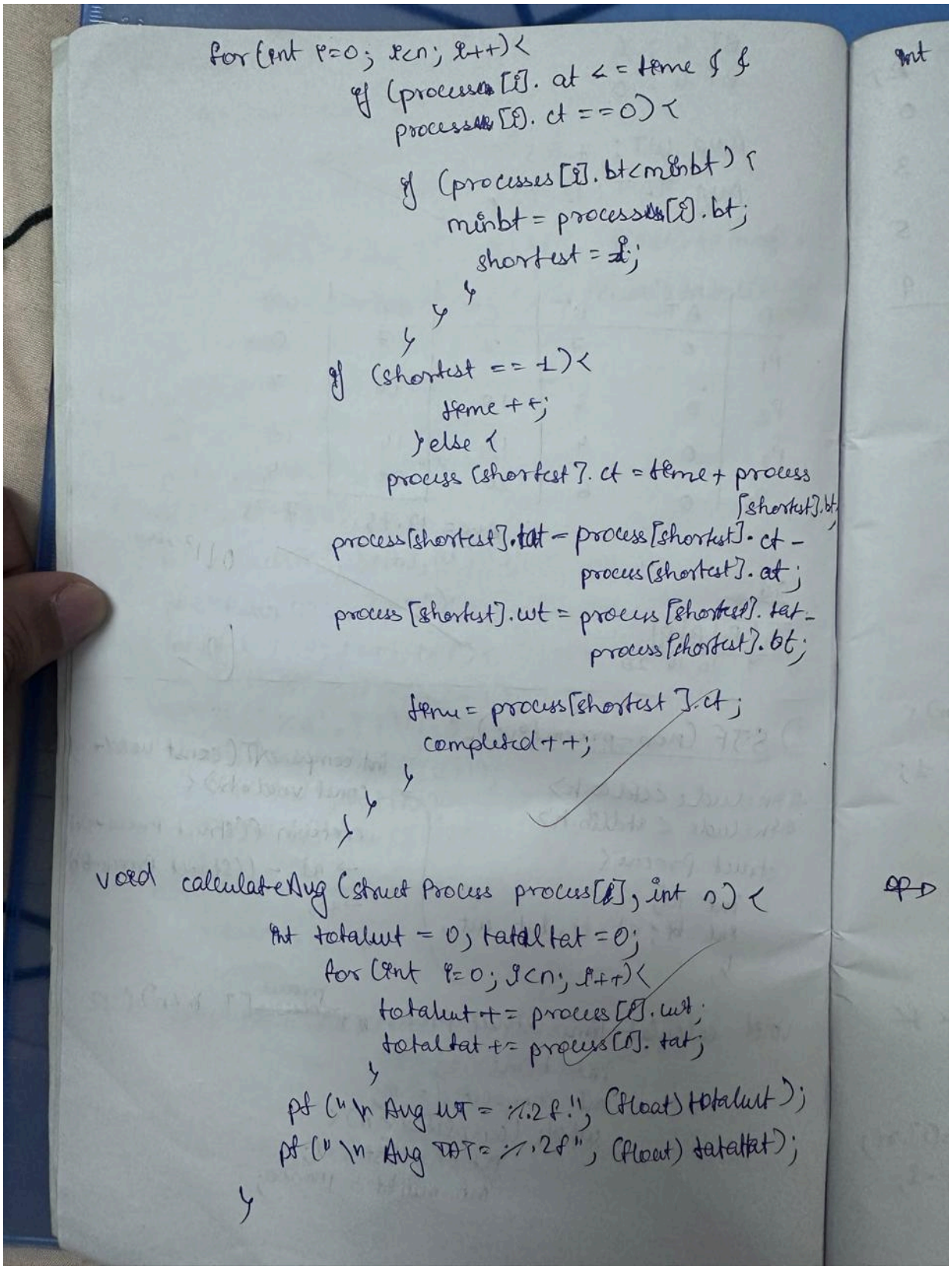
```
        } else {
            processes[shortest].ct = time + processes[shortest].bt;
            processes[shortest].tat = processes[shortest].ct - processes[shortest].at;
            processes[shortest].wt = processes[shortest].tat - processes[shortest].bt;

            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;
}
```

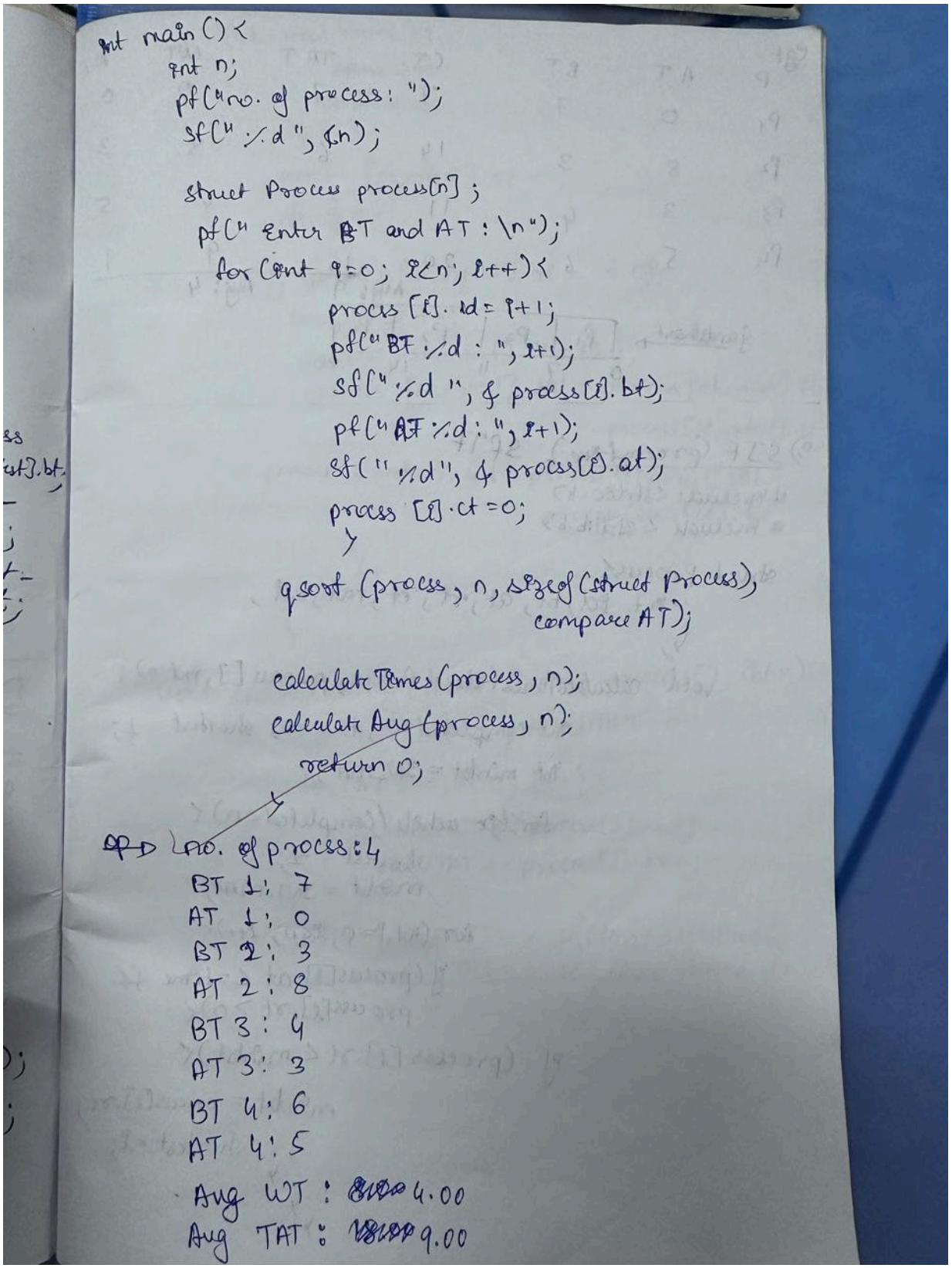
```
no. of processes: 4
enter bt and at:
bt 1: 7
at 1: 0
bt 2: 3
at 2: 8
bt 3: 4
at 3: 3
bt 4: 6
at 4: 5

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

2) SJF (non-preemptive)

```
#include <stdio.h>
#include <stdlib.h>
struct Process {
    int id;
    int bt, at, ct, tat, wt;
};
int compareAT(const void *a,
    const void *b) {
    return ((struct Process *)a)->at
    - ((struct Process *)b)->at;
}
void calculateTimes(struct Process process[], int n) {
    int time = 0;
    int completed = 0;
    while (completed < n) {
        int shortest = -1;
        int minbt = 100000;
```

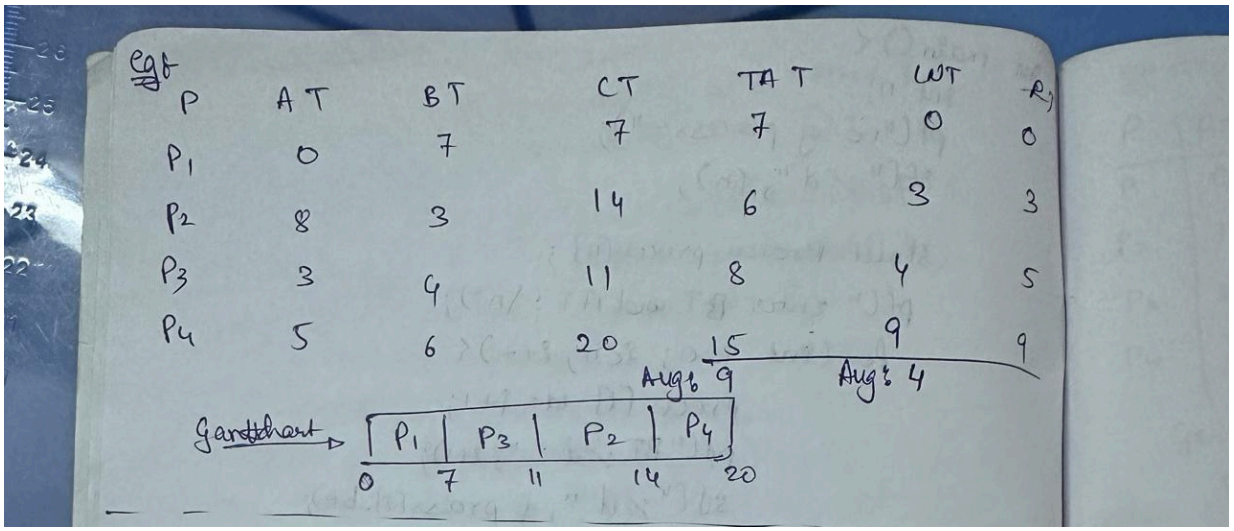


```
for (int i=0; i<n; i++){
    if (process[i].at <= time && process[i].ct == 0){
        if (process[i].bt < minbt){
            minbt = process[i].bt;
            shortest = i;
        }
    }
    if (shortest == -1){
        time++;
    }else{
        process[shortest].ct = time + process[shortest].bt;
        process[shortest].tat = process[shortest].ct - process[shortest].at;
        process[shortest].wt = process[shortest].tat - process[shortest].bt;

        time = process[shortest].ct;
        completed++;
    }
}

void calculateAvg(struct Process process[], int n){
    int totalwt = 0, totaltat = 0;
    for (int i=0; i<n; i++){
        totalwt += process[i].wt;
        totaltat += process[i].tat;
    }
    pf("\n Avg WT = %.2f", (float)totalwt);
    pf("\n Avg TAT = %.2f", (float)totaltat);
}
```



```
int main() {
    int n;
    pf("no. of process: ");
    sf("%d", &n);

    struct Process process[n];
    pf("Enter BT and AT : \n");
    for(int i=0; i<n; i++) {
        process[i].id = i+1;
        pf("BT %d : ", i+1);
        sf("%d", &process[i].bt);
        pf("AT %d : ", i+1);
        sf("%d", &process[i].at);
        process[i].ct = 0;
    }
    qsort(process, n, sizeof(struct Process),
                                compareAT);

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

```
no. of process: 4
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 (pre-emptive):

```
#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;
}
```

```
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 = 6.50
Avg TAT = 13.00%
```

3) SJF (preemptive) → SRTF

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

struct Process{
    int pid, bt, at, rt, ct, tat, wt;
};
void calculateTimes(struct Process process[], int n){
    int completed=0, time=0, shortest=-1;
    int minbt = INT_MAX;
    while(completed<n){
        shortest = -1;
        minbt = INT_MAX;
    for (int i=0; i<n; i++){
        if(process[i].at <=time && process[i].rt >0){
    if (process[i].rt < minbt){
            minbt = process[i].rt;
            shortest =i;
    }}}
```

```
int main(){
    int n;
    pf("No. of process:");
    sf("%d", &n);

    struct Process process[n];

    pf("Enter BT and AT: \n");
    for(int i=0; i<n; i++){
        process[i].id = i+1;
        pf("BT %d: ", i+1);
        sf("%d", &process[i].bt);
        pf("AT %d: ", i+1);
        sf("%d", &process[i].at);
        process[i].rt = process[i].bt;
    }
    calculateTime(process, n);
    calculateAvg(process, n);
    return 0;
}
```

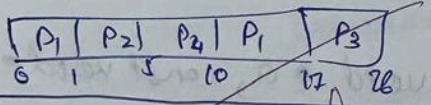
Op→

```
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: 6.50
Avg TAT: 13.00
```

eg:

P	AT	BT	CT	TAT	WT	RT
P_1	0	8	17	17	9	0
P_2	1	4	5	4	0	0
P_3	2	9	26	24	15	15
P_4	3	5	10	7	2	2
				Avg: 13	6.50	

Gantt Chart



2. Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.
→ Priority (Non-pre-emptive):

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

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

int comparePriority(const void *a, const void *b) {
   struct Process *p1 = (struct Process *)a;
   struct Process *p2 = (struct Process *)b;
   if (p1->at == p2->at)
      return p1->priority - p2->priority;
   return p1->at - p2->at;
}

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

   while (completed < n) {
      int highest = -1, highestPriority = 1000000;

      for (int i = 0; i < n; i++) {
         if (processes[i].at <= time && processes[i].ct == 0) {
            if (processes[i].priority < highestPriority) {
               highestPriority = processes[i].priority;
               highest = i;
            }
         }
      }

      if (highest == -1) {
         time++;
      } else {
```

```
            processes[highest].ct = time + processes[highest].bt;
            processes[highest].tat = processes[highest].ct - processes[highest].at;
            processes[highest].wt = processes[highest].tat - processes[highest].bt;

            time = processes[highest].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];

    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);
        printf("Priority %d: ", i + 1);
        scanf("%d", &processes[i].priority);
        processes[i].ct = 0;
    }
```

```
    qsort(processes, n, sizeof(struct Process), comparePriority);

    calculateTimes(processes, n);
    calculateAvg(processes, n);

    return 0;
}
```

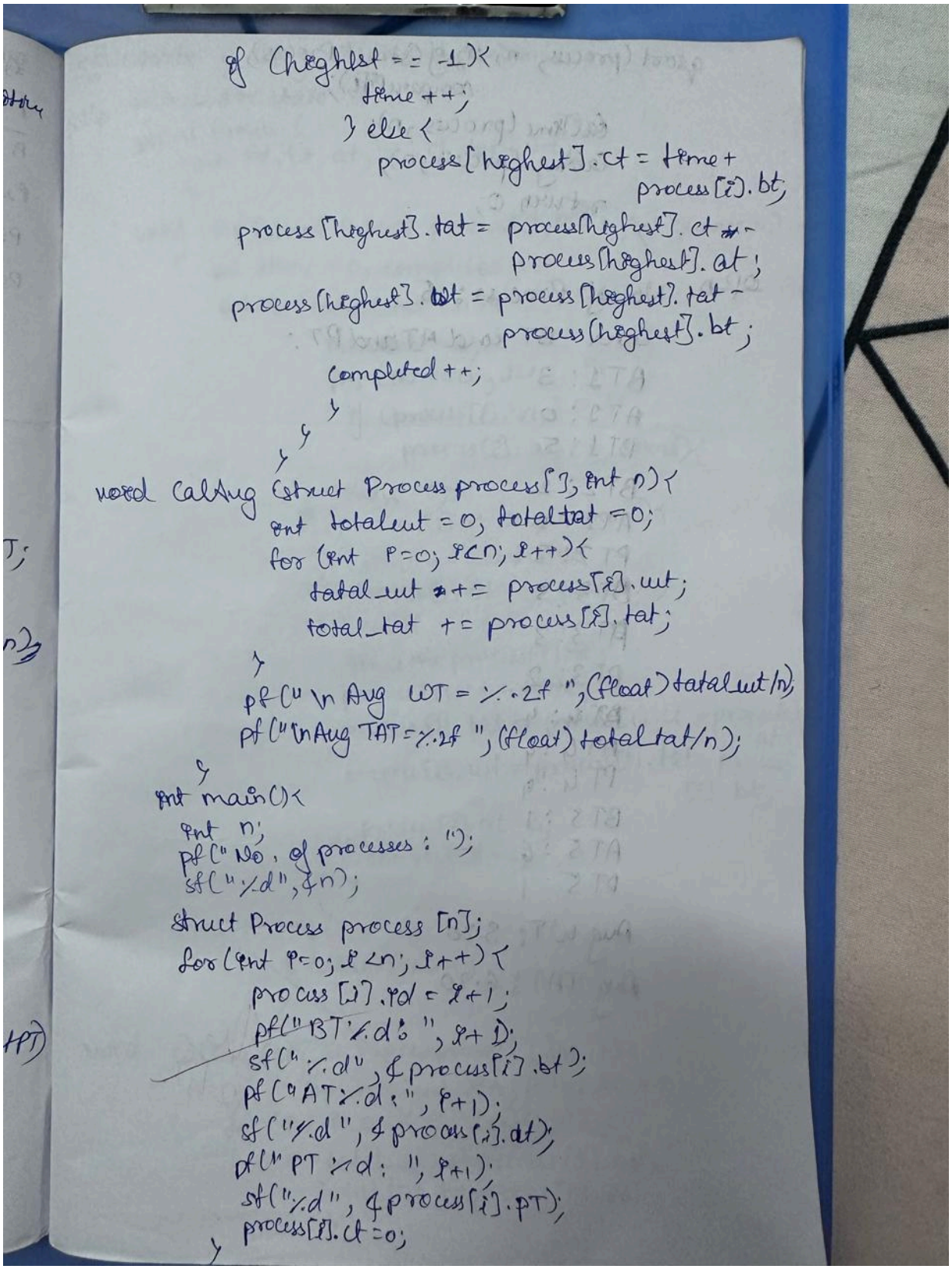
```
No. of processes: 5
BT 1: 3
AT 1: 0
Priority 1: 5
BT 2: 2
AT 2: 2
Priority 2: 3
BT 3: 5
AT 3: 3
Priority 3: 2
BT 4: 4
AT 4: 4
Priority 4: 4
BT 5: 1
AT 5: 6
Priority 5: 1

Avg WT = 3.20
Avg TAT = 6.20
```

LAB-2

4) Write a C prg of Priority Scheduling of non-preemptive

soln

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

int main(){
    int id, bt, at, pt, ct, tat, wt;
};

int comparePT(const void *a, const void *b)
{ struct Process *p1 = (struct Process*) a;
  struct Process *p2 = (struct Process*) b;
  if (p1->at == p2->at)
        return p1->PT - p2->PT;
    return p1->at - p2->at;
}

void calculateTime(struct Process process[], int n)
{
  int time = 0, completed = 0;
  while (completed < n){
        int highest = -1;
        int highestPT = 10000000;
    for (int i=0; i<n; i++){
        if (process[i].at <= time &&
        process[i].ct == 0){
            if (process[i].pt < highestPT)
{ highestPT = process[i].pT;
        highest = i;
    }
  }
}
```



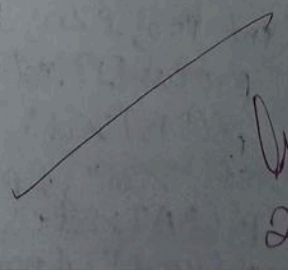
```
if (highest == -1){
        time++;
    } else {
        process[highest].ct = time +
                              process[i].bt;
process[highest].tat = process[highest].ct -
                       process[highest].at;
process[highest].wt = process[highest].tat -
                      process[highest].bt;
        completed++;
    }
  }
}
void calAvg (struct Process process[], int n){
    int totalwt = 0, totaltat = 0;
    for (int i=0; i<n; i++){
        total_wt += process[i].wt;
        total_tat += process[i].tat;
    }
    pf("\n Avg WT = %.2f ", (float) total_wt/n);
    pf("\n Avg TAT = %.2f ", (float) total_tat/n);
}
int main(){
    int n;
    pf("No. of processes : ");
    sf("%d", &n);
    struct Process process[n];
    for (int i=0; i<n; i++){
        process[i].pid = i+1;
        pf("BT %d: ", i+1);
        sf("%d", &process[i].bt);
        pf("AT %d: ", i+1);
        sf("%d", &process[i].at);
        pf("PT %d: ", i+1);
        sf("%d", &process[i].pt);
        process[i].ct = 0;
    }
```

```
qsort(process, n, sizeof(struct Process),
                    comparePT);
    calTime(process, n);
    calAvg(process, n);
    return 0;
}
```

5@

soln

op> No. of Process: 5
Enter BT and AT and PT:
BT1: 3
AT1: 0
PT1: 5
BT2: 2
AT2: 2
PT2: 3
BT3: 5
AT3: 3
PT3: 2
BT4: 4
AT4: 4
PT4: 4
BT5: 1
AT5: 6
PT5: 1
Avg WT: 3.20
Avg TAT: 6.20



→ Priority (pre-emptive):

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

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

int compareArrival(const void *a, const void *b) {
   struct Process *p1 = (struct Process *)a;
   struct Process *p2 = (struct Process *)b;
   return p1->at - p2->at;
}

void calculateTimes(struct Process processes[], int n) {
   int time = 0, completed = 0, min_priority, shortest;
   for (int i = 0; i < n; i++) processes[i].rt = processes[i].bt;

   while (completed < n) {
      shortest = -1;
      min_priority = 1000000;

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

      if (shortest == -1) {
         time++;
      } else {
         processes[shortest].rt--;
         time++;
         if (processes[shortest].rt == 0) {
```

```
            processes[shortest].ct = time;
            processes[shortest].tat = processes[shortest].ct - processes[shortest].at;
            processes[shortest].wt = processes[shortest].tat - processes[shortest].bt;
            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];

  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);
    printf("Priority %d: ", i + 1);
    scanf("%d", &processes[i].priority);
  }

  qsort(processes, n, sizeof(struct Process), compareArrival);
```

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

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

Avg WT = 9.86
Avg TAT = 13.71
```

6 Q) Write a C prg for Priority preemtive scheduling?

Soln →

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

int comparePT (const void * a, const void * b) {
    struct Process * p1 = (struct Process *) a;
    struct Process * p2 = (struct Process *) b;
    if (p1->at == p2->at)
        return p2->pt - p2->pt;
    return p1->at - p2->at;
}

void calculateTime(struct Process process[], int n) {
    int time = 0, completed = 0;
    while (completed < n) {
        int highest = -1, highestPT = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (process[i].at <= time && process[i].rt > 0) {
                if (process[i].priority < highestPT) {
                    highestPT = process[i].pt;
                    highest = i;
                }
            }
        }
        if (highest == -1) {
            time ++;
        } else {
            process[highest].rt --;
            if (process[highest].rt == 0) {
                process[highest].ct = time + 1;
                process[highest].tat = process[highest].ct
                    - process[highest].at;
                process[highest].wt = process[highest].tat -
                    process[highest].bt;
```

```
        }
        time++;
    }
}

void calculateAvg (struct process process[], int n){
    int totalwt=0, totaltat=0;
    for (int i=0; i<n; i++){
        total_wt += process[i].wt;
        total_tat += process[i].tat;
    }
    pf("\n Avg WT = %.2f", (float) totalwt/n);
    pf("\n Avg TAT = %.2f", (float) total_tat/n);
}
int main(){
    int n;
    pf("No. of process: ");
    sf("%d", &n);
    struct process process[n];
    for (int i=0; i<n; i++){
        process[i].pid = i+1;
        pf("BT %d: ", i+1);
        sf("%d", &process[i].bt);
        pf("AT %d: ", i+1);
        sf("%d", &process[i].at);
        pf("Priority %d: ", i+1);
        sf("%d", &process[i].pt);
        process[i].rt = process[i].bt;
    }
    qsort(process, n, sizeof(struct process), comparePT);
    calculateTime(process, n);
    calculateAvg(process, n);
    return 0;
}
```

OP1

No. of process: 7

Enter BT and AT and PT:

BT 1: 8
AT 1: 0
PT 1: 3

BT 2: 2
AT 2: 1
PT 2: 4

BT 3: 4
AT 3: 3
PT 3: 4

BT 4: 1
AT 4: 4
PT 4: 5

BT 5: 6
AT 5: 5
Avg WT:
Avg TAT:
PT 5: 2

BT 6: 5
AT 6: 6
PT 6: 6

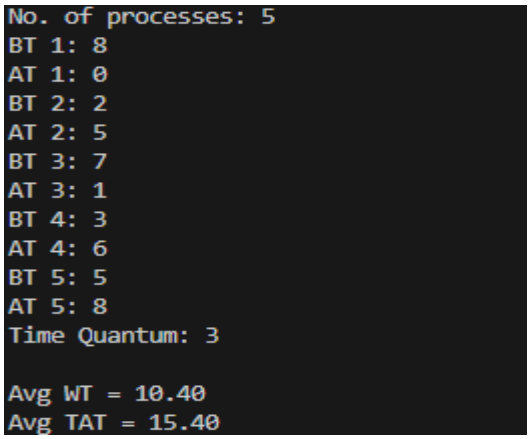
BT 7: 1
AT 7: 7
PT 7: 1

Avg TAT: 13.71
Avg WT: 9.86

→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;
```

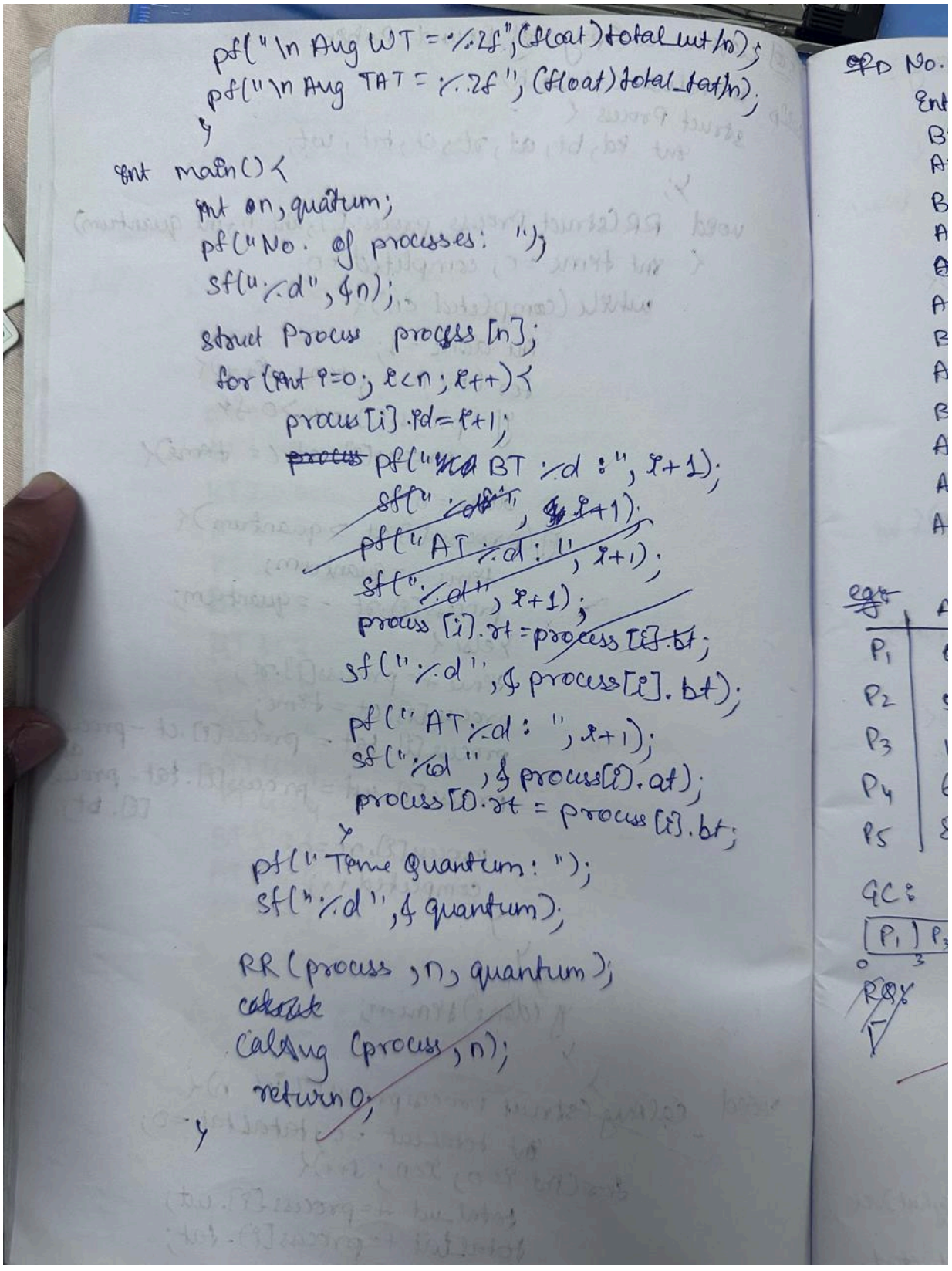
```
    }
    printf("\nAvg WT = %.2f", (float)total_wt / n);
    printf("\nAvg TAT = %.2f", (float)total_tat / n);
}

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;
}
```


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

5] create a C prg of Round-Robin.

soln

```
#include<stdio.h>
struct Process {
    int id, bt, at, rt, ct, tat, wt;
};
void RR(struct Process process[], int n, int quantum)
{ int time=0, completed=0;
  while(completed<n){
      int done=1;
      for(int i=0; i<n; i++){
        if(process[i].rt>0 &&
           process[i].at<=time){
        done=0;
      if(process[i].rt>quantum){
        time+=quantum;
        process[i].rt-=quantum;
      }else{
      time+=process[i].rt;
      process[i].ct=time;
      process[i].tat=process[i].ct-process[i].
                                        at;
      process[i].wt=process[i].tat-process
                                        [i].bt;
       process[i].rt=0;
       completed++;
      }
    }
  if(done) time++;
}
void CalAvg(struct Process process[], int n){
      int totalwt=0, totaltat=0;
  for(int i=0; i<n; i++){
      totalwt+=process[i].wt;
      totaltat+=process[i].tat;
  }
```



```
    pf("\n Avg WT = %.2f", (float)total_wt/n);
    pf("\n Avg TAT = %.2f", (float)total_tat/n);
}
int main(){
    int n, quantum;
    pf("No. of processes: ");
    sf("%d", &n);
    struct Process process[n];
    for(int i=0; i<n; i++){
        process[i].id = i+1;
        pf("BT %d : ", i+1);
        sf("%d", &process[i].bt);
        pf("AT %d : ", i+1);
        sf("%d", &process[i].at);
        process[i].rt = process[i].bt;
    }
    pf("Time Quantum: ");
    sf("%d", &quantum);
    RR(process, n, quantum);
    CalAvg(process, n);
    return 0;
}
```

P_1	
P_2	
P_3	
P_4	
P_5	

GC:

P_1	P_2
0	3

No. of processes: 5

Enter BT and AT:

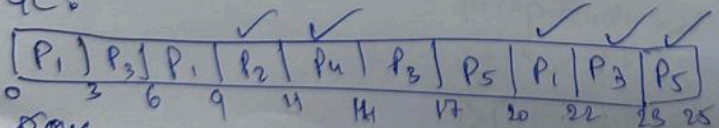
BT1: 8
AT1: 0
BT2: 2
AT2: 5
BT3: 7
AT3: 1
BT4: 3
AT4: 6
BT5: 5
AT5: 8

→ Time Quantum: 3

Avg WT: 10.00
Avg TAT: 15.60

	AT	BT	CT	TAT	WT
P_1	0	8	22	22	14
P_2	5	2	11	6	4
P_3	1	7	23	22	15
P_4	6	3	14	8	5
P_5	8	5	25	17	12

GC:



RQ:

3. 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 scheduling:

```
#include <stdio.h>

#define MAX_PROCESSES 10

typedef struct {
    int pid;
    int bt;
    int at;
    int queue;
} Process;

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

void roundRobin(Process p[], int n, int quantum, int wt[], int tat[], int rt[]) {
    int remaining_bt[MAX_PROCESSES];
    for (int i = 0; i < n; i++)
        remaining_bt[i] = p[i].bt;
```

```
    int t = 0, completed = 0;

    while (completed < n) {
        int executed = 0;
        for (int i = 0; i < n; i++) {
            if (remaining_bt[i] > 0) {
                if (rt[i] == -1) rt[i] = t;

                if (remaining_bt[i] > quantum) {
                    t += quantum;
                    remaining_bt[i] -= quantum;
                } else {
                    t += remaining_bt[i];
                    tat[i] = t - p[i].at;
                    wt[i] = tat[i] - p[i].bt;
                    remaining_bt[i] = 0;
                    completed++;
                }
                executed = 1;
            }
        }
        if (!executed) t++;
    }
}

void fcfs(Process p[], int n, int start_time, int wt[], int tat[], int rt[]) {
    int time = start_time;
    for (int i = 0; i < n; i++) {
        if (time < p[i].at) time = p[i].at;
        rt[i] = time - p[i].at;
        wt[i] = rt[i];
        tat[i] = wt[i] + p[i].bt;
        time += p[i].bt;
    }
}

int main() {
```

```
    int n, quantum;
    Process p[MAX_PROCESSES], sys[MAX_PROCESSES],
usr[MAX_PROCESSES];
    int sys_count = 0, usr_count = 0;
    int wt[MAX_PROCESSES], tat[MAX_PROCESSES], rt[MAX_PROCESSES];

    printf("NO. of process: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter BT,AT and Q (1=sys, 2=user) for P%d: ", i + 1);
        p[i].pid = i + 1;
        scanf("%d %d %d", &p[i].bt, &p[i].at, &p[i].queue);

        if (p[i].queue == 1)
            sys[sys_count++] = p[i];
        else
            usr[usr_count++] = p[i];

        wt[i] = 0;
        tat[i] = 0;
        rt[i] = -1;
    }

    printf("QT for RR: ");
    scanf("%d", &quantum);

    sortByArrival(sys, sys_count);
    sortByArrival(usr, usr_count);

    roundRobin(sys, sys_count, quantum, wt, tat, rt);
    int last_sys_time = (sys_count > 0) ? tat[sys_count - 1] + sys[sys_count - 1].at : 0;
    fcfs(usr, usr_count, last_sys_time, &wt[sys_count], &tat[sys_count],
&rt[sys_count]);

    printf("\nP\tQ\tWT\tTAT\tRT\n");
    for (int i = 0; i < n; i++)
```

```
        printf("P%d\t%d\t%d\t\t%d\t\t\t%d\n", p[i].pid, p[i].queue, wt[i], tat[i], rt[i]);

    float avg_wt = 0, avg_tat = 0, avg_rt = 0;
    for (int i = 0; i < n; i++) {
        avg_wt += wt[i];
        avg_tat += tat[i];
        avg_rt += rt[i];
    }

    printf("\nAVG WT: %.2f", avg_wt / n);
    printf("\nAVG TAT: %.2f", avg_tat / n);
    printf("\nAVG RT: %.2f\n", avg_rt / n);

    return 0;
}
```

```
NO. of process: 4
Enter BT,AT and Q (1=sys, 2=user) for P1: 2 0 1
Enter BT,AT and Q (1=sys, 2=user) for P2: 1 0 2
Enter BT,AT and Q (1=sys, 2=user) for P3: 5 0 1
Enter BT,AT and Q (1=sys, 2=user) for P4: 3 0 2
QT for RR: 2

P       Q       WT      TAT     RT
P1      1       0               2                       0
P2      2       2               7                       2
P3      1       7               8                       7
P4      2       8               11                      8

AVG WT: 4.25
AVG TAT: 7.00
AVG RT: 4.25
```

LAB 3

3/4/2025

8) write a C prog to simulate multi-lvl queue sch algo. All process in sys. are divided into 2 categories - sys processes & user processes. Sys. processes are to be given higher priority than user processes. Use RR and FCFS sch. for the process in each queue.

soln

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

typedef struct {
    int pid;
    int bt;
    int at;
    int queue;
} Process;

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

void RR(Process p[], int n, int QT, int WT[], int TAT[],
        int RT[]) {
    int rBT[MAX_PROCESSES];
    int t=0, completed=0;
```

```
while (completed < n) {
    int exe = 0;
    for (int i=0; i<n; i++) {
        if (rBT[i] > 0) {
            if (RT[i] == -1) RT[i] = t;
            if (rBT[i] > QT) {
                t += QT;
                rBT[i] -= QT;
            } else {
                t += rBT[i];
                TAT[i] = t - p[i].at;
                wt[i] = tat[i] - p[i].bt;
                rBT[i] = 0;
                completed++;
            }
            exe = 1;
        }
    }
    if (!exe) t++;
}

void fcfs (Process p[], int n, int st, int wt[], int tat[],
           int rt[]) {
    int time = st;
    for (int i=0; i<n; i++) {
        if (time < p[i].at) time = p[i].at;
        rt[i] = time - p[i].at;
        wt[i] = rt[i];
        tat[i] = wt[i] + p[i].bt;
        time += p[i].bt;
    }
}
```

```
int main() {
    int n, qT;
    Process p[MAX_PROCESSES], sys[MAX_PROCESSES], usr[MAX_
                                                    PROCESSES];
    int sys_count = 0, usr_count = 0;
    int wt[MAX_PROCESS], tat[MAX_PROCESS], rt[MAX_PROCESS];

    pf("No. of processes: ");
    sf("%d", &n);
    for (int i=0; i<n; i++) {
        pf("Enter BT, AT and Q (1=sys, 2=usr) for
          P%d: ", i+1);
        p[i].pid = i+1;
        sf("%d %d %d", &p[i].bt, &p[i].at, &p[i].queue);

            if (p[i].queue == 1)
                sys[sys_count++] = p[i];

            else
                usr[usr_count++] = p[i];

            wt[i] = 0;
            tat[i] = 0;
            rt[i] = -1;
    }
    pf("QT for RR: ");
    sf("%d", &qT);
    sortAT(sys, sys_count);
    sortAT(usr, sys_count);

    RR(sys, sys_count, qT, wt, tat, rt);
    int last_sys_time = (sys_count > 0) ? tat[sys_count - 1]
    + sys[sys_count - 1].at : 0;
    fcfs(usr, usr_count, last_sys_time, &wt[sys_count],
        &tat[sys_count], &rt[sys_count]);
```

203

```
pf("\n P\t Q\t WT\t TAT\t RT\n");
for(int i=0; i<n; i++)
  pf("P:%d \t:%d \t:%d \t\t:%d \t\t:%d\n",
    P[i].pid, p[i].queue, wt[i], tat[i], rt[i]);

float avg_wt=0, avg_tat=0, avg_rt=0;
for(int i=0; i<n; i++){
    avg_wt += wt[i];
    avg_tat += tat[i];
    avg_rt += rt[i];
}
pf("\n AVG WT: %.2f \n AVG TAT: %.2f \n
    AVG RT: %.2f \n", avg_wt/n,
        avg_tat/n, avg_rt/n);
return 0;
}
```

Op>

No. of process 4

Enter BT, AT and PT Enter BT, AT and Q(1=sys, 2=user)

BT 1

for P1: 2 0 1

Enter BT, AT and Q(1=sys, 2=user) for

P2: 1 0 2

Enter BT, AT and Q(1=sys, 2=user) for

P3: 5 0 1

Enter BT, AT and Q (1=sys, 2=user) for

P4: 3 0 2

for RR: 2

P	Q	WT	TAT	RT
P1	1	0	2	0
2	2	2	7	
3	1	7	8	2
4	2	8	11	8

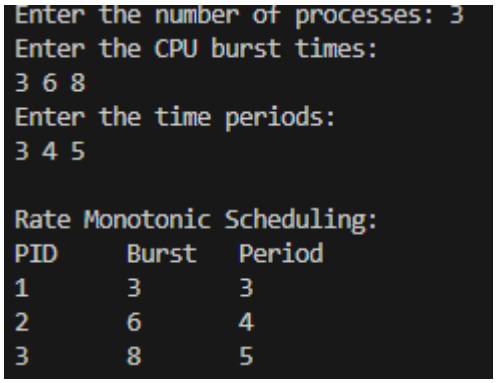
Avg WT: 4.25 Avg RT: 4.25

Avg TAT: 7.00

4. Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic:

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

#define MAX_PROCESSES 10
#define MAX_TIME 50  // Maximum simulation time

typedef struct {
    int pid;
    int burst;
    int period;
    int remaining_time;
    int next_arrival;
} Process;

void rate_monotonic_scheduling(Process p[], int n) {
    int time = 0, executed;
    printf("\nRate Monotonic Scheduling:\n");
    printf("PID\tBurst\tPeriod\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\n", p[i].pid, p[i].burst, p[i].period);

    while (time < MAX_TIME) {
        executed = -1;

        for (int i = 0; i < n; i++) {
            if (p[i].next_arrival <= time && p[i].remaining_time > 0) {
                if (executed == -1 || p[i].period < p[executed].period)
                    executed = i;
            }
        }

        if (executed != -1) {
            printf("%dms : Task %d is running.\n", time, p[executed].pid);
            p[executed].remaining_time--;
```

```
            if (p[executed].remaining_time == 0) {
                p[executed].next_arrival += p[executed].period;
                p[executed].remaining_time = p[executed].burst; // Reset for periodic
execution
            }
        }
        time++;
    }
}

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].pid = i + 1;
        scanf("%d", &processes[i].burst);
        processes[i].remaining_time = processes[i].burst;
    }

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

    rate_monotonic_scheduling(processes, n);

    return 0;
}
```


Enter the number of processes: 3
Enter the CPU burst times:
3 6 8
Enter the time periods:
3 4 5
Rate Monotonic Scheduling:
PID Burst Period
1 3 3
2 6 4
3 8 5

2) Rate-monotonic scheduling 3/4/2025

```
#include <stdio.h>
#define MAX_PROCESS 10
typedef struct {
    int id, bt, period, rt, next_dl;
} process;
void sort_by_period (Process process[], int n) {
    for (int i=0; i<n-1; i++) {
        for (int j=0; j<n; j++) {
            if (process[j].period > process[j+1].
                period)
            { process temp = process[j];
              process[j] = process[j+1];
              process[j+1] = temp;
            }
        }
    }
int gcd (int a, int b) {
    return (a*b)/gcd(a,b);
}
int calculate_lcm (Process process[], int n) {
    int result = process[0].period;
    for (int i=1; i<n; i++) {
        result = lcm (result, process[i].
                 period);
    }
    return result;
}
double utilization_factor (Process process[], int n)
{ double sum=0;
```

```
    return sum;
}
double rms_threshold(int n){
    return n*(pow(2.0,1.0/n)-1);
}
void rate_monotonic_scheduling(Process process[], int n){
int LCM_period = calculate_LCM(process, n);
pf("LCM = %d\n", LCM_period);
pf("Rate monotonic scheduling:\n");
for(int i=0; i<n; i++){
    pf("%d %d %d\n", process[i].id,
    process[i].bt, process[i].period);
}
double utilization = utilization_factor(process, n);
double threshold = rms_threshold(n);
pf("%f <= %f => %s\n", utilization,
    threshold, (utilization <= threshold) ? "true":
    "false");
if(utilization > threshold){
    pf("It sys 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 % process[i].period==0){
    process[i].rt = process[i].bt;
```

```
if (process[i].rt > 0){
        selected = i;
        break;
    }
}
if (selected == -1){
    pf("Time %d : Process %d is running\n",
    timeline, process[selected].bt, process[selected].
    rt--; executed++;}
    else {
    pf("Time %d : CPU is idle\n", timeline);
    }
    timeline++;
}
}
int main(){
    int n;
    Process process[MAX_PROCESS];
    pf("Enter the no. of process: ");
    sf("%d", &n);
    pf("Enter the CPU bt:\n");
    for (int i=0; i<n; i++){
        process[i].id = i+1;
        sf("%d", &process[i].bt);
        process[i].rt = process[i].bt;
    }
    pf("Enter the time period=\n");
    for(int i=0; i<n; i++){
        sf("%d", &process[i].period);
    }
    sort by period(process, n);
    rate_monotonic_scheduling(process
```

Op

Enter the no. of process = 3

Enter the CPU bt :

3 6 8

Enter the time period :

3 4 5

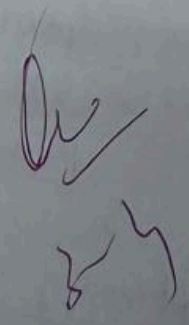
LCM = 60

Rate Monotonic Scheduling:

PID	Burst	Period
1	3	3
2	6	4
3	8	5

4.100000 <= 0.779763 ⇒ false

System may not be schedulable



b) Earliest-deadline:

```
#include <stdio.h>

int gcd(int a, int b) {
   while (b != 0) {
      int temp = b;
      b = a % b;
      a = temp;
   }
   return a;
}

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

struct Process {
   int id, burst_time, deadline, period;
};

void earliest_deadline_first(struct Process p[], int n, int time_limit) {
   int time = 0;
   printf("Earliest Deadline Scheduling:\n");
   printf("PID\tBurst\tDeadline\tPeriod\n");
   for (int i = 0; i < n; i++) {
      printf("%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burst_time, p[i].deadline,
p[i].period);
   }

   printf("\nScheduling occurs for %d ms\n", time_limit);
   while (time < time_limit) {
      int earliest = -1;
      for (int i = 0; i < n; i++) {
         if (p[i].burst_time > 0) {
            if (earliest == -1 || p[i].deadline < p[earliest].deadline) {
               earliest = i;
```

```
            }
        }
    }

    if (earliest == -1) break;

    printf("%dms: Task %d is running.\n", time, p[earliest].id);
    p[earliest].burst_time--;
    time++;
  }
}

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

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

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

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

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

```
    }

    printf("\nSystem will execute for hyperperiod (LCM of periods): %d ms\n",
hyperperiod);

    earliest_deadline_first(processes, n, hyperperiod);

    return 0;
}
```

```
Enter the number of processes: 3
Enter the CPU burst times:
2 3 4
Enter the deadlines:
1 2 3
Enter the time periods:
1 2 3

System will execute for hyperperiod (LCM of periods): 6 ms
Earliest Deadline Scheduling:
PID     Burst   Deadline        Period
1       2               1               1
2       3               2               2
3       4               3               3

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

5] Earliest - Deadline scheduling 3/4/2025

```
#include <stdio.h>
int gcd(int a, int b){
    while (b != 0){
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
int Lcm(int a, int b){
    return (a * b) / gcd(a, b);
}
struct Process {
    int id, bt, deadline, period;
};
void earliest_deadline_first(struct Process p[], int n,
    int time_limit){
    int time = 0;
    printf("Earliest Deadline scheduling :\n");
    printf("PID\t BT\t DL\t Period \n");
    for(int i=0; i<n; i++) {
        printf("%d \t %d \t %d \t %d \n",
        p[i].id, p[i].bt, p[i].deadline, p[i].period);
    }
    printf("\n scheduling occurs for %d ms\n", time_limit);
    while (time < time_limit){
        int earliest = -1;
        for(int i=0; i<n; i++) {
            if (p[i].bt > 0){
                if (earliest == -1 || p[i].deadline
                    < p[earliest].deadline) {
                    earliest = i;
                }
```

```
if (earliest == -1) break;
printf("%d ms = Task %d is running\n",
time, p[earliest].id);
p[earliest].bt--;
time++;
}
}

int main(){
    int n;
printf("Enter the no. of process: ");
scanf("%d", &n);
struct Process p[n];
printf("Enter the CPU BT :\n");
for(int i=0; i<n; i++){
        scanf("%d", &p[i].bt);
        p[i].id = i+1;
    }
    printf("Enter the deadlines :\n");
    for(int i=0; i<n; i++){
            scanf("%d", &p[i].deadline);
        }
printf("Enter the time periods :\n");
for(int i=0; i<n; i++){
    scanf("%d", &p[i].period);
}
int hyperperiod = p[0].period;
for(int i=0; i<n; i++){
        hyperperiod = lcm(hyperperiod,
                          p[i].period);
    }
printf("\n system will exe for hyperperiod.
    (LCM of periods) : %d ms\n", hyperperiod);
```

Of D

Enter the no. of process: 3
Enter the CPU BT;
2 3 4
Enter the Deadlins:
1 2 3
Enter the timeperiods:
1 2 3

System will exe of hyperperiod (LCM of periods): 6ms

Earliest Deadline scheduling:

PID	BT	DL	Period
1	2	1	1
2	3	2	2
3	4	3	3

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

Viva
3-4-25

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

producer-consumer:

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

int mutex = 1;
int full = 0;
int empty=3;
int buffer_item = 0;

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

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

void producer(int id) {
   if ((mutex == 1) && (empty != 0)) {
      int value;
      scanf("%d", &value);
      printf("producer %d produced %d\n", id, value);
      mutex = wait(mutex);
      full = signal(full);
      empty = wait(empty);
      buffer_item = value;
      printf("buffer:%d\n", buffer_item);
      mutex = signal(mutex);
   } else {
      printf("buffer is full!\n");
   }
}
```

```
void consumer(int id) {
   if ((mutex == 1) && (full != 0)) {
      mutex = wait(mutex);
      full = wait(full);
      empty = signal(empty);
      printf("consumer %d consumed %d\n", id + 1, buffer_item);
      printf("current buffer len: 0\n");
      mutex = signal(mutex);
   } else {
      printf("buffer is empty!\n");
   }
}

int main() {
   int np, nc, cap;

   printf("enter no. of producers:");
   scanf("%d", &np);
   printf("enter no. of consumers:");
   scanf("%d", &nc);
   printf("enter buffer capacity:");
   scanf("%d", &cap);

   empty = cap;

   for (int i = 1; i <= np; i++) {
      printf("producer created: %d\n", i);
   }

   for (int i = 1; i <= nc; i++) {
      printf("consumer created: %d\n", i);
   }

   while (1) {
      producer(1);
      consumer(1);
   }
```

```
    return 0;
}
```

```
enter no. of producers:1
enter no. of consumers:1
enter buffer capacity:1
producer created: 1
consumer created: 1
16
producer 1 produced 16
buffer:16
consumer 2 consumed 16
current buffer len: 0
32
producer 1 produced 32
buffer:32
consumer 2 consumed 32
current buffer len: 0
4
producer 1 produced 4
buffer:4
consumer 2 consumed 4
current buffer len: 0
0
producer 1 produced 0
buffer:0
consumer 2 consumed 0
current buffer len: 0
```

16/4/2025

LAB-4

Q] Producer - Consumer

```
#include <stdio.h>
#include <stdlib.h>
int mutex=1, full=0, empty=3, 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:%d\n",x);
        mutex=signal(mutex);
    }
void consumer(){
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        pf("consumer consumes item:%d\n",x);
        x--;
        mutex=signal(mutex);
    }
int main(){
    int choice;
    while(1){
        pf("\n1.Producer\n2.Consumer\n3.Exit");
```

```
sf("%d", &choice);
switch(choice) {
    case 1:
    if((mutex==1) && (empty!=0))
        producer();
    else
        pf("Buffer is full!\n");
    break;

    case 2:
    if((mutex==1) && (full!=0))
        consumer();
    else
        pf("Buffer is empty!\n");
    break;

    case 3:
    exit(0);

    default:
    pf("Invalid choice!\n");
}
return 0;
}
```

OP:

1. Producer
2. Consumer
3. Exit

Enter your choice: 1
Producer produces item 1

Enter your choice: 1
Producer produces item 2

Enter your choice: 1
Producer produces item 3

Enter your choice: 1

Enter your choice: 2
Consumer consumes item 3

Enter your choice: 2
Consumer consumes item 2

Enter your choice: 2
Consumer consumes item 1

Enter your choice: 2
Buffer is empty!

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

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

#define MAX 5
sem_t mutex;
sem_t chopstick[MAX];
int totalPhilosophers;
int hungryCount;
int hungryPhilosophers[MAX];
int choice;

void *philosopher(void *arg) {
   int id = *(int *)arg;
   printf("P %d is waiting\n", id + 1);
   sem_wait(&mutex);
   sem_wait(&chopstick[id]);
   sem_wait(&chopstick[(id + 1) % totalPhilosophers]);
   printf("P %d is granted to eat\n", id + 1);
   sleep(1);
   printf("P %d has finished eating\n", id + 1);
   sem_post(&chopstick[id]);
   sem_post(&chopstick[(id + 1) % totalPhilosophers]);
   sem_post(&mutex);
   pthread_exit(NULL);
}

int main() {
   pthread_t thread[MAX];
   int i;
   printf("Enter the total number of philosophers: ");
   scanf("%d", &totalPhilosophers);
   printf("How many are hungry: ");
```

```
    scanf("%d", &hungryCount);
    for (i = 0; i < hungryCount; i++) {
        printf("Enter philosopher %d position (1 to %d): ", i + 1, totalPhilosophers);
        scanf("%d", &hungryPhilosophers[i]);
        hungryPhilosophers[i]--;
    }
    for (i = 0; i < totalPhilosophers; i++) {
        sem_init(&chopstick[i], 0, 1);
    }
    sem_init(&mutex, 0, 1);
    while (1) {
        printf("\n1. One can eat at a time\n");
        printf("2. Two can eat at a time\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        if (choice == 3) {
            break;
        }
        if (choice == 1) {
            printf("Allow one philosopher to eat at any time\n");
            for (i = 0; i < hungryCount; i++) {
                for (int j = 0; j < hungryCount; j++) {
                    printf("P %d is waiting\n", hungryPhilosophers[j] + 1);
                }
                int *id = malloc(sizeof(int));
                *id = hungryPhilosophers[i];
                pthread_create(&thread[i], NULL, philosopher, id);
                pthread_join(thread[i], NULL);
            }
        }
    }
    return 0;
}
```

```
How many are hungry: 3
Enter philosopher 1 position (1 to 5): 2
Enter philosopher 2 position (1 to 5): 4
Enter philosopher 3 position (1 to 5): 5

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
Allow one philosopher to eat at any time
P 2 is waiting
P 4 is waiting
P 5 is waiting
P 2 is waiting
P 2 is granted to eat
P 2 has finished eating
P 2 is waiting
P 4 is waiting
P 5 is waiting
P 4 is waiting
P 4 is granted to eat
P 4 has finished eating
P 2 is waiting
P 4 is waiting
P 5 is waiting
P 5 is waiting
P 5 is granted to eat
P 5 has finished eating

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
Allow one philosopher to eat at any time
P 2 is waiting
P 4 is waiting
P 5 is waiting
P 2 is waiting
P 2 is granted to eat
```

Q) Dining - Philosopher

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
    int total Philosophers;
    int hungry[MAX];
    int areNeighbors (int a, int b) {
        return (abs(a-b) == 1 || abs(a-b) == total Philosophers - 1);
    }
void opt1(int count){
    pf("\n Allow one philosopher to eat at any time\n");
    for(int i=0; i<count; i++) {
        pf("P %d is granted to eat\n", hungry[i]);
        for(int j=0; j<count; j++) {
            if(j != i){
            pf("P %d is waiting\n",
                hungry[j]);
        }
    }
}
void opt2 (int count) {
    pf("\n Allow two philosophers to eat at same time\n");
    int combination = 1;
    for(int i=0; i<count; i++){
        for(int j=i+1; j<count; j++){
    if(areNeighbors(hungry[i], hungry[j])){
        pf("combination %d\n", combination);
    pf("P %d and P %d are granted to eat\n",
        hungry[i], hungry[j])
```

```
for (int k=0; k < count; k++){
        if (k != i && k != j){
            pf("P %d is waiting \n", hungry[k]);
        }
    }
    pf("\n");
}
if (combination == 1){
    pf("No combination found when
      two non-neighbour philosopher
      can eat \n");
}
}
int main(){
    int hungry count;
    pf("Enter the total no. of philosophers ");
    sf("%d", &totalPhilosophers);
    pf("How many are hungry : ");
    sf("%d ", &hungry count);
    for(int i=0; i<hungryCount; i++){
        pf("Enter philosopher %d position ", i+1);
        sf("%d", &hungry[i]);
    }
    int choice;
do{
    printf("\n1. One can eat at a time\n
        2. Two can eat at a time\n
        3. Exit\n");
    pf("Enter your choice: ");
    sf("%d", &choice);
    switch(choice){
        case 1:
            option1(hungryCount);
            break;
```

```
Case 2:
    option2(hungry(count));
    break;
case 3:
    pf("Exiting\n");
    break;
default:
    pf("Invalid choice\n");
}
}while(choice != 3);
return 0;
}
```

Op▷

```
Enter the total no. of philosophers: 5
How many are hungry : 3
Enter philosopher1 posn : 2
Enter philosopher2 posn : 4
Enter philosopher3 posn : 5

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1

Allow one philosopher to eat at any time
P2 is granted to eat
P4 is waiting
P5 is waiting
P4 is granted to eat
P2 is waiting
P5 is waiting
P5 is granted to eat
P2 is waiting
P4 is waiting
```

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2

Allow Esophelosophers to eat at same time

combination 1
P2 and P4 are granted to eat
P5 is waiting

Combination 2
P2 and P5 are granted to eat
P4 is waiting

16.4.3

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

```
#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], 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("eter 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 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];
    bool finish[n];
    for (int i = 0; i < n; i++)
        finish[i] = false;
    int safeSeq[n];
    int work[m];
    for (int i = 0; i < m; i++)
        work[i] = avail[i];
    int count = 0;
    while (count < n) {
        bool found = false;
        for (int p = 0; p < n; p++) {
            if (!finish[p]) {
```

```
            int j;
            for (j = 0; j < m; j++)
                if (need[p][j] > work[j])
                    break;
            if (j == m) {
                for (int k = 0; k < m; k++)
                    work[k] += alloc[p][k];

                safeSeq[count++] = p;
                finish[p] = true;
                found = true;
            }
        }
    }
    if (!found) {
        printf("sys is not in a safe state.\n");
        return 0;
    }
  }
  printf("sys is in safe state.\nsafe sequence is: ");
  for (int i = 0; i < n; i++)
    printf("P%d%s", safeSeq[i], (i == n - 1) ? "\n" : " -> ");
  return 0;
}
```

```
5 3
enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
eter max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
enter available matrix:
3 3 2
sys is in safe state.
safe sequence is: P1 -> P3 -> P4 -> P0 -> P2
```

LAB-5 ▷ avoidance

Q) Deadlock ~~detection~~ ▷

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

int main(){
    int n,m;
    pf("Enter no. of prouss & resources");
    sf("%d %d", &n, &m);
    int alloc[n][m], request[n][m], avail[m];
    pf("Enter allocation matrix");
    for(int i=0; i<n; i++)
        for(int j=0; j<m; j++)
            sf("%d", &all[i][j]);

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

    pf("Enter avail matrix:");
    for(int i=0; i<n; i++)
        sf("%d", &avail[i]);

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

    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])
```

```
if (p == 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("sys is not in a deadlock");
else
    printf("sys is in deadlock state");
  return 0;
}
```

OP

```
Enter num of processes & resource
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 avail matrix:
3 3 2
```

```
process can finish
" 3 can finish
" 4 "
sys is in a
deadlock state
execution time:
75.1575
```

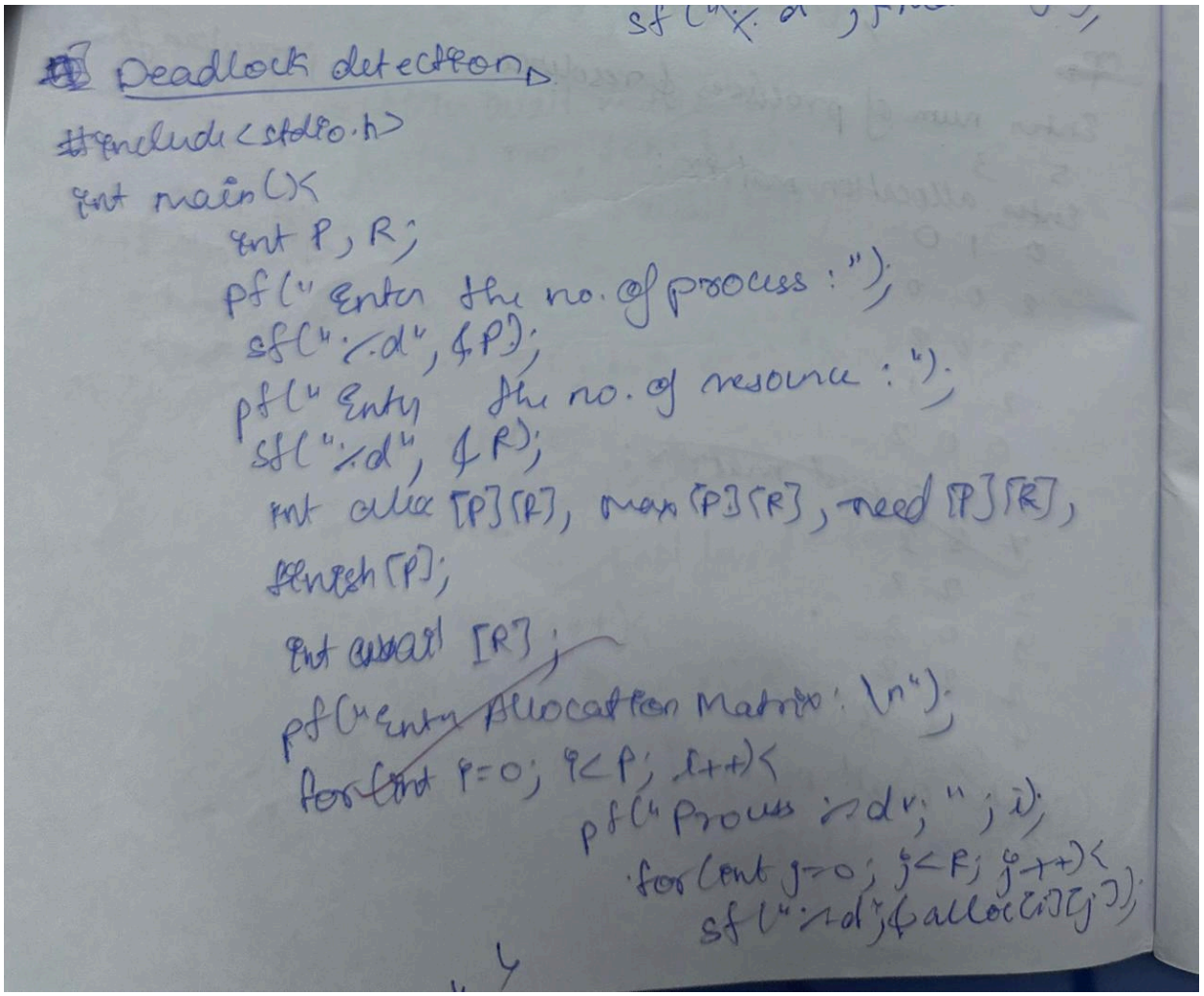
8. Write a C program to simulate deadlock detection

```
#include <stdio.h>
#include <stdbool.h>
int main() {
    int n, m;
    printf("enter num of processes and number of resources:\n");
    scanf("%d %d", &n, &m);
    int alloc[n][m], request[n][m], avail[m];
    printf("enter alloc matrix (%d x %d):\n", n, m);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);
    printf("enter req matrix (%d x %d):\n", n, m);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &request[i][j]);

    printf("enter avail resource (%d values):\n", m);
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);
    int work[m];
    for (int i = 0; i < m; i++)
        work[i] = avail[i];
    bool finish[n];
    for (int i = 0; i < n; i++) {
        bool hasAllocation = false;
        for (int j = 0; j < m; j++) {
            if (alloc[i][j] != 0) {
                hasAllocation = true;
                break;
            }
        }
        finish[i] = hasAllocation ? false : true;
    }
    while (true) {
        bool progress = false;
```

```
    for (int i = 0; i < n; i++) {
        if (!finish[i]) {
            bool canGrant = true;

            for (int j = 0; j < m; j++) {
                if (request[i][j] > work[j]) {
                    canGrant = false;
                    break;
                }
            }

            if (canGrant) {
                for (int j = 0; j < m; j++)
                    work[j] += alloc[i][j];
                finish[i] = true;
                progress = true;
            }
        }
    }

    if (!progress)
        break;
}
printf("\nDLD result:\n");
bool deadlock = false;

for (int i = 0; i < n; i++) {
    if (!finish[i]) {
        printf("process P%d is deadlocked\n", i);
        deadlock = true;
    } else {
        printf("process P%d is not deadlocked\n", i);
    }
}
if (!deadlock)
    printf("\nno deadlock detected in the system.\n");
```

```
    return 0;
}
```

```
enter num of processes and number of resources:
5 3
enter alloc matrix (5 x 3):
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
enter req matrix (5 x 3):
0 0 0
2 0 2
0 0 1
1 0 0
0 0 2
enter avail resource (3 values):
0 0 0

DLD result:
process P0 is not deadlocked
process P1 is deadlocked
process P2 is deadlocked
process P3 is deadlocked
process P4 is deadlocked
```



Deadlock detection

```
#include<stdio.h>
int main()
{
    int P, R;
    pf("Enter the no. of process : ");
    sf("%d", &P);
    pf("Enter the no. of resource : ");
    sf("%d", &R);
    int alloc[P][R], max[P][R], need[P][R],
    finish[P];
    int avail[R];
    pf("Enter Allocation Matrix: \n");
    for(int i=0; i<P; i++)
        pf("Process %d: ", i);
        for(int j=0; j<R; j++)
            sf("%d", &alloc[i][j]);
}
```

```
pf("Enter max", matrix : ");
for (int i=0; i<P; i++){
        pf("Process %d", i);
        for (int j=0; j<R; j++){
            sf("%d", &max[i][j]);
        }
}
pf("Enter Avail Resources: \n");
for (int i=0; i<R; i++);
{
for (int i=0; i<P; i++){
        finish[i]=0;
        for (int j=0; j<P; j++){
            need[i][j] = max[i][j] -
                        alloc[i][j];
        }
}
int count = 0;
bool deadlock = false;
while (count<P){
        bool found = false;
        for (int i=0; i<P; i++){
            if (!finish[i]){
                int j=j;
                for (j=0; j<R; j++){
                    if (need[i][j] > avail[i])
                        break;
                }
                if (j==R){
                    for (int k=0; k<R; k++)
                        avail[k] += alloc[i][j];
                    finish[i]=1;
                    found = true;
                    count++;
```

```
        pf("Process %d can finish: \n", i);
    }
  }
 }

 if(!found)
    deadlock = true;
    break;
  }
 }

 if(!deadlock)
    pf("sys is not in a deadlock
      state: \n");

 else
    pf("sys is in a deadlock state \n");
 return 0;
}
```

Op →

Enter num of process & resources:
5 3

Enter num of Alloc matrix
Process 0 : 0 1 0
Process 1 : 2 0 0
Process 3 : 3 0 2
Process 4 : 0 0 2

Enter Max matrix:
Process 0 : 7 5 3
Process 1 : 3 2 2
Process 2 : 9 0 2
Process 3 : 2 2 2
Process 4 : 4 3 3

Enter Avail Resource:
3 2 2
process 1 can finish
process 3 can finish
4 4 4
4 0 "
4 2 "
Sys is not in deadlock state.

14-5-25

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

```
#include <stdio.h>

struct Block {
   int size;
   int is_free;
};

struct File {
   int size;
};

void firstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
   printf("\nfirst fit alloc\n");
   printf("file no\tfile size\tblock no\tblock size\tfrag\n");
   for (int i = 0; i < n_files; i++) {
      int allocated = 0;
      for (int j = 0; j < n_blocks; j++) {
         if (blocks[j].is_free && blocks[j].size >= files[i].size) {
            blocks[j].is_free = 0;
            printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, j + 1,
blocks[j].size, blocks[j].size - files[i].size);
            allocated = 1;
            break;
         }
      }
      if (!allocated) {
         printf("file %d cannot be allocated\n", i + 1);
      }
   }
}
```

```
void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
   printf("\best fit alloc\n");
   printf("file no\tfile size\tblock no\tblock size\tfrag\n");
   for (int i = 0; i < n_files; i++) {
      int best_block = -1, min_fragment = 10000;
      for (int j = 0; j < n_blocks; j++) {
         if (blocks[j].is_free && blocks[j].size >= files[i].size) {
            int fragment = blocks[j].size - files[i].size;
            if (fragment < min_fragment) {
               min_fragment = fragment;
               best_block = j;
            }
         }
      }
      if (best_block != -1) {
         blocks[best_block].is_free = 0;
         printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, best_block + 1,
blocks[best_block].size, min_fragment);
      } else {
         printf("File %d cannot be allocated\n", i + 1);
      }
   }
}

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
   printf("\nworst fit alloc\n");
    printf("file no\tfile size\tblock no\tblock size\tfrag\n");
   for (int i = 0; i < n_files; i++) {
      int worst_block = -1, max_fragment = -1;
      for (int j = 0; j < n_blocks; j++) {
         if (blocks[j].is_free && blocks[j].size >= files[i].size) {
            int fragment = blocks[j].size - files[i].size;
            if (fragment > max_fragment) {
               max_fragment = fragment;
               worst_block = j;
            }
         }
```

```
        }
        if (worst_block != -1) {
            blocks[worst_block].is_free = 0;
            printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, worst_block + 1,
blocks[worst_block].size, max_fragment);
        } else {
            printf("file %d cannot be allocated\n", i + 1);
        }
    }
}

int main() {
    int n_blocks, n_files;

    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];

    for (int i = 0; i < n_blocks; i++) {
        printf("enter the size of block %d: ", i + 1);
        scanf("%d", &blocks[i].size);
        blocks[i].is_free = 1;
    }

    for (int i = 0; i < n_files; i++) {
        printf("enter the size of file %d: ", i + 1);
        scanf("%d", &files[i].size);
    }

    firstFit(blocks, n_blocks, files, n_files);

    for (int i = 0; i < n_blocks; i++) blocks[i].is_free = 1;
```

```
    bestFit(blocks, n_blocks, files, n_files);

    for (int i = 0; i < n_blocks; i++) blocks[i].is_free = 1;

    worstFit(blocks, n_blocks, files, n_files);

    return 0;
}
```

```
enter the number of blocks: 5
enter the number of files: 4
enter the size of block 1: 100
enter the size of block 2: 500
enter the size of block 3: 200
enter the size of block 4: 300
enter the size of block 5: 600
enter the size of file 1: 212
enter the size of file 2: 417
enter the size of file 3: 112
enter the size of file 4: 420

first fit alloc
file no file size       block no        block size      frag
1       212             2               500             288
2       417             5               600             183
3       112             3               200             88
file 4 cannot be allocated
est fit alloc
file no file size       block no        block size      frag
1       212             4               300             88
2       417             2               500             83
3       112             3               200             88
4       420             5               600             180

worst fit alloc
file no file size       block no        block size      frag
1       212             5               600             388
2       417             2               500             83
3       112             4               300             188
file 4 cannot be allocated
```

LAB-6

Q) First fit, best fit, worst fit

```
#include<stdio.h>
#define MAX 10
void allocate(int blocks[], int nBlocks, int process[],
              int nProcess, int method)
{
    int alloc[MAX], i, j;
    for(i=0; i<nProcess; i++) alloc[i] = -1;
    for(i=0; i<nProcess; i++){
        int idx = -1;
        for(j=0; j<nBlocks; j++){
            if(method == 1)
                idx = j; break;
            if(method == 2){
                if(idx == -1 || blocks[j]<
                   blocks[idx])
                    idx = j;
            }
            if(method == 3){
                if(idx == -1 || blocks[j]>
                   blocks[idx])
                    idx = j;
            }
        }
    }
    printf("\n process no. \t process size \t Block no.\n");
    for(i=0; i<nProcess; i++){
        printf("%d\t\t%d\t\t", i+1, process[i]);
        if(alloc[i] != -1) printf("%d\n", alloc[i]);
        else printf("Not Allocated\n");
    }
}
```

```
int main() {
    int blocks[MAX] = {100, 500, 200, 300, 600};
    int process[MAX] = {212, 417, 112, 426};
    int nBlocks = 5; nProcess = 4;
    pf("First fit :");
    int ff[MAX];
    for (int i=0; i<nBlocks; i++)
        ff[i] = blocks[i];
    allocate(ff, nBlocks, process, nProcess, 1);
    pf("\n Best fit :");
    int bf[MAX];
    for (int i=0; i< nBlocks; i++)
        bf[i] = blocks[i];
    allocate(bf, nBlocks, process, nProcess, 2);
    pf("\n Worst fit :");
    int wf[MAX];
    for (int i=0; i<nBlocks; i++)
        wf[MAX] = blocks[i];
    allocate(wf, nBlocks, process, nProcess, 3);
    return 0;
}
```

OP:

First fit:

Process no.	Process size	Block no.
1	212	2
2	417	5
3	112	2
4	426	Not allocated

Best fit:

Process no.	Process size	Block no.
1	212	4
2	417	2
3	112	3
4	426	5

worst fit :

process no.	process size	Block no.
1	212	5
2	417	2
3	112	5
4	426	Not Allocated

LAB-7

Q) FIFO

```
#include <stdio.h>
int search (int key, int frame[], int size)
{ for (int i=0; i< size; i++) {
    if (frame[i] == key)
      return 1;
  }
  return 0;
}
void simulate FIFO (int pages[], int n, int framesize)
{ int frame[framesize], front = 0, faults = 0, hits = 0;
  for (int i=0; i< framesize; i++)
    frame[i] = -1;
  for (int i=0; i<n; i++){
    if (!search (pages[i], frame, framesize))
    {
      frame[front] = pages[i];
      front = (front+1) % framesize;
      faults++;
    }
    else
    { hits++;
    }
  }
  pf("FIFO page faults: %d, pagehits: %d\n",
                    faults, hits);
}
```

10. Write a C program to simulate page replacement algorithms
a)FIFO

```
#include <stdio.h>

int search(int key, int frame[], int size) {
   for (int i = 0; i < size; i++) {
      if (frame[i] == key)
         return 1;
   }
   return 0;
}

void simulateFIFO(int pages[], int n, int frameSize) {
   int frame[frameSize], front = 0, faults = 0, hits = 0;

   for (int i = 0; i < frameSize; i++)
      frame[i] = -1;

   for (int i = 0; i < n; i++) {
      if (!search(pages[i], frame, frameSize)) {
         frame[front] = pages[i];
         front = (front + 1) % frameSize;
         faults++;
      } else {
         hits++;
      }
   }

   printf("FIFO page fault: %d, page hits: %d\n", faults, hits);
}

int main() {
   int n, frameSize;

   printf("enter the size of the pages: ");
   scanf("%d", &n);
```

```
    int pages[n];
    printf("enter the page strings: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &pages[i]);
    }

    printf("enter the no of page frames: ");
    scanf("%d", &frameSize);

    simulateFIFO(pages, n, frameSize);

    return 0;
}
```

```
enter the size of the pages: 7
enter the page strings: 1 3 0 3 5 6 3
enter the no of page frames: 3
FIFO page fault: 6, page hits: 1
PS C:\Users\Admin>
```

worst fit :

process no.	process size	Block no.
1	212	5
2	417	2
3	112	5
4	426	Not Allocated

LAB-7

Q) FIFO

```
#include <stdio.h>
int search (int key, int frame[], int size)
{ for (int i=0; i< size; i++) {
    if (frame[i] == key)
      return 1;
  }
  return 0;
}
void stimulate FIFO (int pages[], int n, int framesize)
{ int frame[framesize], front = 0, faults = 0, hits = 0;
  for (int i=0; i< framesize; i++)
    frame[i] = -1;
  for (int i=0; i<n; i++){
    if (!search (pages[i], frame, framesize))
    {
      frame[front] = pages[i];
      front = (front+1) % framesize;
      faults++;
    }
    else
    { hits++;
    }
  }
  pf("FIFO page faults: %d, pagehits: %d\n",
      faults, hits);
}
```

```
int main(){
    int n, framesize;
    pf("Enter the size of the pages: ");
    sf("%d", &n);
    int pages[n];
    pf("Enter the page strings: ");
    for(int i=0; i<n; i++)
            sf("%d", &pages[i]);
    pf("Enter the no. of page frames: ");
    sf("%d", &framesize);
    simulate FIFO(pages, n, framesize);
    return 0;
}
```

```
Enter the size of pages: 7
Enter the page strings: 1 3 0 3 5 6 3
FIFO page faults: 6
Page Hits: 1
```

b)LRU and Optimal:

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

int search(int key, int frame[], int frameSize) {
   for (int i = 0; i < frameSize; i++) {
      if (frame[i] == key)
         return i;
   }
   return -1;
}

int findOptimal(int pages[], int frame[], int n, int index, int frameSize) {
   int farthest = index, pos = -1;
   for (int i = 0; i < frameSize; i++) {
      int j;
      for (j = index; j < n; j++) {
         if (frame[i] == pages[j]) {
            if (j > farthest) {
               farthest = j;
               pos = i;
            }
            break;
         }
      }
      if (j == n)
         return i;
   }
   return (pos == -1) ? 0 : pos;
}

void simulateFIFO(int pages[], int n, int frameSize) {
   int frame[frameSize], front = 0, count = 0, hits = 0;

   for (int i = 0; i < frameSize; i++)
      frame[i] = -1;
```

```
    for (int i = 0; i < n; i++) {
        if (search(pages[i], frame, frameSize) == -1) {
            frame[front] = pages[i];
            front = (front + 1) % frameSize;
            count++;
        } else {
            hits++;
        }
    }
    printf("FIFOpage faults: %d,page hits: %d\n", count, hits);
}

void simulateLRU(int pages[], int n, int frameSize) {
    int frame[frameSize], time[frameSize], count = 0, hits = 0;

    for (int i = 0; i < frameSize; i++) {
        frame[i] = -1;
        time[i] = 0;
    }

    for (int i = 0; i < n; i++) {
        int pos = search(pages[i], frame, frameSize);
        if (pos == -1) {
            int least = 0;
            for (int j = 1; j < frameSize; j++) {
                if (time[j] < time[least])
                    least = j;
            }
            frame[least] = pages[i];
            time[least] = i;
            count++;
        } else {
            hits++;
            time[pos] = i;
        }
    }
```

```
    printf("LRU page faults: %d, page hits: %d\n", count, hits);
}

void simulateOptimal(int pages[], int n, int frameSize) {
    int frame[frameSize], count = 0, hits = 0;

    for (int i = 0; i < frameSize; i++)
        frame[i] = -1;

    for (int i = 0; i < n; i++) {
        if (search(pages[i], frame, frameSize) == -1) {
            int index = -1;
            for (int j = 0; j < frameSize; j++) {
                if (frame[j] == -1) {
                    index = j;
                    break;
                }
            }
            if (index != -1) {
                frame[index] = pages[i];
            } else {
                int replaceIndex = findOptimal(pages, frame, n, i + 1, frameSize);
                frame[replaceIndex] = pages[i];
            }
            count++;
        } else {
            hits++;
        }
    }
    printf("optimal page faulkt: %d, page hits: %d\n", count, hits);
}

int main() {
    int n, frameSize;
    printf("enter the size of the pages: ");
    scanf("%d", &n);
```

```
    int pages[n];
    printf("enter the page strings: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &pages[i]);

    printf("enter the no of page frames: ");
    scanf("%d", &frameSize);

    simulateFIFO(pages, n, frameSize);
    simulateOptimal(pages, n, frameSize);
    simulateLRU(pages, n, frameSize);

    return 0;
}
```

```
enter the size of the pages: 12
enter the page strings: 7 0 1 2 0 3 0 4 2 3 0 3
enter the no of page frames: 3
FIFOpage faults: 10,page hits: 2
optimal page faulkt: 7, page hits: 5
LRU page faults: 9, page hits: 3
PS C:\Users\Admin>
```

LRU and optimal

```
#include <stdio.h>
#include <stdlib.h>
int search (int key, int frame[], int framesize)
    { for (int i=0; i< framesize ; i++)
            { if (frame [i] == key)
                return;
            }
          return -1;
    }
int findOptimal (int pages [], int frame[], int n,
                 int index, int framesize)
    {int farthat = index, pos = -1;
     for (int i=0; i< framesize; i++)
        { int j;
          for (j= index; j<n; j++)
            { if (frame [i] == pages [j])
              { if (j> farthat)
                  { farthat = j;
                    pos= i;
                  }
                break;
              }
            }
          if (j==n)
            return i;
        }
      return (pos == -1) ? 0 : pos;
    }
```

```
void simulateLRU(int pages[], int n, int framesize)
{int frame[framesize], time[framesize],
 count=0, hits=0;
 for(int i=0; i<framesize; i++)
    {frame[i]=-1;
     time[i]=0;
    }
 for(int i=0; i<n; i++)
  {int pos = search(pages[i], frame, framesize);
   if(pos==-1)
   {int last=0;
    for(int j=1; j<framesize; j++)
     {
      if(time[j]<time[least])
        least=j;
      }
     frame[least]=pages[i];
     time[least]=1;
     count++;
    }
  else {
     hits++;
     time[pos]=i;
  }}
 pf("LRU Page faults: %d, pages Hits: %d\n",
                    count, hits);
}
void simulateOptimal(int pages, int n, int framesize)
{int frame[framesize], count=0, hits=0;
 for(int i=0; i<framesize; i++)
     frame[i]=-1;
  for(int i=0; i<n; i++)
  {
```

```
if (search(pages[i], frame, frameSize) == -1)
    { int index = -1;
      for (int j=0; j<frameSize; j++)
        { if (frame[j] == -1)
            { index = j;
              break;
            }
        }
      if (index != -1)
        { frame[index] = pages[i];
        }
else { int replaceIndex = findOptimal(pages, frame,
                                      n, i+1, frameSize);
       frame[replacement index] = pages[i];
       }
       count ++
    }
else
    { hits ++;
    }
}
pf("Optimal page faults: %d, page hits: %d \n",
                                  count, hits);
}
int n, frameSize;
pf("Enter the size of the pages: ");
sf("%d", &n);
int pages[n];
pf("Enter the page strings: \n");
for (int i=0; i<n; i++)
```

```
simulateOptimal(pages, n, framesize);
simulateLRU(pages, n, framesize);
    return 0;
}
```

Op:

Enter the size of pages: 12
Enter the page strings: 7 0 1 2 0 3 0 4 2 3 5

Enter the no. of frames: 3
Optimal page faults: 7, page Hits = 5
LRU Page faults: 9, pageHits: 3