PAGE NO :
DATE :

Q) # write a program to implement First come First Serve scheduling: and calculate average turn around time and average waiting time

```
#include<stdio.h>
int main(){
    int n;
    printf(" Enter the number of processes ");
    scanf("%d",&n);
    int arr_time[n];
    int burst_time[n];
    int comp_time[n];
    int TAT[n];
    int WT[n];

    // initialising arrival time and burst time;

    for(int i=0; i<n; i++){
        printf("arrival of p%d:", (i+1));
        scanf("%d", &arr_time[i]);
        printf("burst time of p%d:",(i+1));
        scanf("%d", &burst_time[i]);
    }
        // sorting the processes
    for (int i=0; i<n; i++){
        for(int j= i+1; j<n; j++){
            if (arr_time[j] < arr_time[i]){
                //swaping processes
                int temp= arr_time[j];
                arr_time[j]= arr_time[i];
                arr_time[i]= temp;
                // swaping burst times
                int tmp= burst_time[j];
                burst_time[j]= burst_time[i];
                burst_time[i]= tmp;
```

PAGE NO :
DATE :

```
        }
    }
}
// calculating complete time
compl_time[0]= burst_time[0]+arr_time[0];
for (int i=1; i<n; i++){
    if (arr_time[i] <= compl_time[i-1]){
        compl_time[i]= compl_time[i-1]+burst_time[i]
    }
    else{
        compl_time[i]= arr_time[i]+burst_time[i]
    }
}

// calculating TAT
for (int i=0; i<n; i++){
    TAT[i]= compl_time[i]-arr_time[i];
}
// calculating WT
for (int i=0; i<n; i++){
    WT[i]= TAT[i]-burst_time[i];
}

// calculating avg TAT & WT
double sum_tat=0;
double sum_wt=0;
for (int i=0; i<n; i++){
    sum_tat+= TAT[i];
    sum_wt+= WT[i];
}
double avg_tat= sum_tat/n;
double avg_wt= sum_wt/n;
```

PAGE NO :
DATE :

```
printf("\n avg tat = %.2f", avg_tat);
printf("\n avg wt = %.2f", avg_wt);

return 0;
}
```

Output

```
Enter number of process : 4
arrival of P1 : 0
burst time of P1 : 2
arrival of P2 : 1
burst time of P2 : 2
arrival of P3 : 5
burst time of P3 : 3
arrival of P4 : 6
burst time of P4 : 4

avg tat = 3.5
avg wt = 0.75
```

8/5/24

5) write a program to implement SJF scheduling

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

typedef struct {
    int pid;
    int arr;
    int burst;
    int comp;
    int turn;
    int wait;
    int processed;
} proc;

void sjf_np(proc process[], int n) {
    int curr_time=0;
    int total_~~time~~ comp_time=0;
    int total_wait_time=0;
    int total_turnaround_time=0;

    while(1){
        int shortest_job = -1;
        int shortest_burst= 9999;
        for(int i=0; i<n; i++){
            if(process[i].arr<= curr_time &&
                process[i].processed == 0){
                if(process[i].burst < shortest_burst
                    shortest_burst = process[i].burst
                    shortest_job=i;
                }
            }
        }
```

PAGE NO :
DATE :

```
if (shortest_job==-1){
    break;
}

procs[shortest_job].comp=curr_time+procs[shortest_job].
                                    burst;
procs[shortest_job].turn=procs[shortest_job].comp-
                        procs[shortest_job].burst;
procs[shortest_job].wait= procs[shortest_job].turn
                        - procs[shorter_job].burst;
if(procs[shortest_job].wait<0)
        procs[shortest_job].wait=0;

total_comp_time+=procs[shortest_job].comp;
total_wait_time+=procs[shortest_job].wait;
total_turnaround_time+=procs[shortest_job].turn;

procs[shortest_job].processed=1;
curr_time=procs[shortest_job].comp;
}

double avg_turnaround_time=(double)total_turn
                                around_time/n;
double avg_waiting_time=(double)total_wait_
                                time/n;
printf("process ID \t Arrival Time \t Burst Time \t
        Completion time \t waiting time \t turnarond");
for(int i=0; i<n; i++){
    printf("%d \t \t %d \t \t %d \t \t %d \t \t
                %d \t \t %d",
        procs[i].pid, procs[i].arr, procs[i].burst,
        procs[i].comp, procs[i].wait, procs[i].turn);
}
```

PAGE NO :
DATE :

```
printf("\n Average turnaround Time: %.2f\n",
            avg_turnaround_time);
printf("Average waiting time: %.2f\n",
            avg_waiting_time);
}
int main(){
    int n;
    proc procs[Max_process];
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for (int i=0; i<n; i++){
        procs[i].pid=i+1;
        printf("Enter arrival time for process %d:",
                i+1);
        scanf("%d", &procs[i].arr);
        printf("Enter burst time for process %d:
                i+1);
        scanf("%d", &procs[i].burst);
        procs[i].processed=0;
    }

    SJF_NP(procs, n);
    return 0;
}
```

<u>Output:</u>

Enter the number of process: 5
Enter arrival time & burst time for process: 1 2 2 1 3 4
4 0 5 2

Process	AT	BT	CT	WT	TAT
1	2	1	7	4	5
2	1	5	16	10	15
3	4	1	8	3	4
4	0	6	6	0	6
5	2	3	11	6	9

PAGE NO :
DATE :

Q) C program to implement semaphore

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#define MAX_ITMS 20
#define BUFFER_SIZE 10

int buffer[BUFFER_SIZE];
int head = 0;
int tail = 0;
int cnt = 0;
pthread_mutex_t mtx;
pthread_cond_t cond_full;
pthread_cond_t cond_empty;

void enque(int item){
    buffer[tail] = item;
    tail = (tail+1) % BUFFER_SIZE;
    cnt++;
}
void deque(){
    int item = buffer[head]
    head = (head+1) % BUFFER_SIZE;
    cnt--;
    return item;
}

void* producer(void* param){
    int prod_cnt=0;
    while(1){
        int item = rand() % 100;
        pthread_mutex_lock(&mtx);
```

PAGE NO :
DATE :

```
while(cnt == BUFFER_SIZE) {
    pthread_cond_wait(&cond_empty, &mtx);
}
if(prod_cnt >= MAX_ITMS) {
    pthread_mutex_unlock(&mtx);
    pthread_cond_signal(&cond_full);
    break;
}

enque(item);
prod_cnt++;
printf("produced: %d\n", item);
pthread_mutex_unlock(&mtx);
pthread_cond_signal(&cond_full);
sleep(rand()%2);
}
return NULL;
}

void consumer(void* param) {
    int cons_cnt=0;
    while(1) {
        pthread_mutex_lock(&mtx);
        while(cnt==0) {
            pthread_cond_wait(&cond_full,
                &mtx);
        }
        if(cons_cnt >= MAX_ITMS) {
            pthread_mutex_unlock(&mtx);
            pthread_cond_signal(&cond_empty);
            break;
        }

        int item=deque();
        cons_cnt++;
        printf("Consumed %d\n", item);
```

```
    pthread_mutex_unlock(&mtx);
    pthread_cond_signal(&cond_empty);
    sleep(rand()%2);
  }

  return NULL;
}

int main(){
    pthread_t tid_prod, tid_cons;
    pthread_mutex_init(&mtx, NULL);
    pthread_cond_init(&cond_full, NULL);
    pthread_cond_init(&cond_empty, NULL);
    pthread_create(&tid_prod, NULL, producer);
    pthread_create(&tid_cons, NULL, consumer, NULL);
    pthread_join(tid_prod, NULL);
    pthread_join(tid_cons, NULL);
    printf(" production & consumption complete
    pthread_mutex_destroy(&mtx);
    pthread_cond_destroy(&cond_full);
    pthread_cond_destroy(&cond_empty);
    return 0;
}
```

Output

```
produced: 91
consumed: 91
produced: 34
produced: 69
produced: 78
produced: 62
produced: 5
consumed: 34
consumed: 69
produced: 81
consumed: 78
consumed: 62
consumed: 5
produced: 61
consumed: 81
consumed: 61
produced: 95
consumed: 95
```

PAGE NO:
DATE:

Round Robin

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

struct process {
    char name[5];
    int AT, BT, ST[20], WT, FT, TAT, pos, CT[20];
};
int quant;

int main() {
    int n, temp;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct process p[n];
    printf("Enter the arrival times & burst times
    for (int i=0; i<n; i++) {
        printf("\n process %d: \n", i+1);
        sprintf(p[i].name, "p%d", i+1);
        printf("Arrival time: ");
        scanf("%d", &p[i].AT);
        printf("Burst time: ");
        scanf("%d", &p[i].BT);
        p[i].pos = i+1;
    }

    printf("Enter the quantum: ");
    scanf("%d", &quant);
```

PAGE NO :
DATE :

```
int c=n, s[n][20];
float time=0; mini=INT_MAX; b[n], a[n];

int index=-1;
for(int i=0; i<n; i++){
    b[i]= p[i].BT;
    a[i]= p[i].AT;
    for(int j=0; j<20; j++){
        s[i][j]= -1;
    }
}

int tat_wt=0, tat_tat=0, tat_rt=0;
bool flag=false;
while (c!=0){
    mini= INT_MAX;
    flag= false;
    for(int i=0; i<n; i++){
        float p = time+0.1;
        if (a[i]<=p && mini>a[i] && b[i]>0){
            index= i;
            mini=a[i];
            flag= true;
        }
    }
    if (!flag){
        time++;
        continue;
    }

    int j=0
    while (s[index][i]!=-1){
        j++;
    }
```

PAGE NO :
DATE :

```
if(s[index][j]==-1){
    s[index][j]= time;
    p[index].ST[j] = time;
}
if (b[index] <= quant){
    time += b[index];
    b[index]=0;
}
else
    time += quant;
    b[index]-=quant;
}
if (b[index]>0){
    a[index] = time + 0.1;
}

if (b[index]==0){
    c--;
    p[index].FT= time;
    p[index].WT=p[index].FT- p[index].AT-
                                p[index].BT;
    tot_wt += p[index].BT + p[index].WT;
    p[index].TAT+= p[index].TAT;
    p[index].CT = time;
    p[index].RT= p[index].ST[0]-p[index]
    tot_rt += p[index].RT;
}
}

printf("\n process \t Name \t arrival \t burst
    \t start \t final \t completion \tot
    \t Responsetime \n");
```

PAGE NO :
DATE :

```
for(int i=0; i<n; i++){
    printf("%d \t %s \t %d \t %d \t \t ",
        p[i].pos; p[i].name, p[i].AT, p[i].BT);
    int j=0;
    while (s[i][j] != -1){
        printf("%d", p[i].ST[j]);
        j++;
    }
    printf("\t %d \t %d \t %d \t %d",
        p[i].FT, p[i].CT, p[i].TAT, p[i].WT,
        p[i].RT);
}
double avg_wt = (double) tot_wt /n;
double avg_tat = (double) tot_tat /n;
double avg_rt = (double) tot_rt /n;

printf("\n the avg WT : %f \n", avg_wt);
printf("\n the avg TAT: %f \n", avg_tat);
printf("\n the avg Response: %f \n", avg_rt);
return 0;
}
```

output

```
Enter no. of process : 5
Enter time quantum: 2
Enter arival time & burst time : 0 5 1 3 2 1
```

Pid	at	bt	ct	tat	wt	rt
1	0	5	13	13	8	0
2	1	3	12	11	8	1
3	2	1	5	3	2	2
4	3	2	9	6	4	4
5	4	3	14	10	7	5

average turn around time: 8.6
average waiting time: 5.8

write a c program to implement Rate monotonic scheduling:

```
# include <stdio.h>
# include <stdlib.h>
# define MAX_TASKS - 10
typedef struct {
    int Ti;
    int Ci;
    int deadline;
    int RT;
    int id;
} Task;
void Input (Task tasks[], int *n_tasks) {
    printf("Enter no.of tasks");
    scanf("%d", &n_tasks);
    for (int i=0; i<n_tasks; i++) {
        task[i].id= i++;
        printf("Enter Ti of task %d", i+1);
        scanf("%d", &tasks[i].[i]);
        printf("Enter execution time of task %d
                i+1);
        scanf("%d", &tasks[i].Ci);
        tasks[i].deadline= tasks[i].Ti;
        tasks[i].RT= tasks[i].Ci;
    }
}

int compare_by_period (const void *a const void
{
    return ((Task*)a)->Ti - ((Task*))->Ti;
}
void RMS (Task tasks[], int n_tasks, int
        time_frame)
{
```

PAGE NO :
DATE :

```
qsort (tasks, n_tasks, sizeof (Tasks), compare_
    by_Period);
printf ("\n Rate_monotonic scheduling");
for (int time=0; time < time_frame; time++)
{
    int s_tasks = -1;
    for (int i=0; i<n_tasks; i++) {
        if (time % tasks[i].Ti==0 ) {
            tasks[i].RT= tasks[i].Ci ;
        }
    if (tasks[i].RT>0 && (s_task == -1) ||
        tasks[i].Ti < tasks [s-task].Ti)) {
            s_task = i;
        }
    }

if (s_task != -1) {
    printf ("Time %d: task %d \n", time,
        tasks[s_task].id );
    tasks [s_task].RT--;
}
else {
    printf ("Time %d : Idle\n", time);
}
}
}

int main() {
Task tasks[max_tasks];
int n_tasks;
int time_frame;
Input (tasks, &n_tasks);
printf (" Enter time frame for simulation:");
scanf ("%d", &time_frame);
ems( tasks, n_tasks, time_frame);
return 0;
}
```

PAGE NO :
DATE :

Output :
Enter no. of tasks : 3
Enter Ti of task 1 : 20
Enter Execution time of task 1 : 3
Enter Ti of task 2 : 5
Enter Execution time of task 2 : 2
Enter Ti of task 3 : 10
Enter time ~~frame~~ execution of task 3 : 2
Enter time frame for simulation : 20
Rate_monotonic scheduling :
Time 0 : task 2
time 1 : task 2
time 2 : task 3
time 3 : task 3
time 4 : task 1
time 5 : task 2
time 6 : task 2
time 7 : task 1
time 8 : task 1
time 9 : Idle
time 10 : task 2
time 11 : task 2
time 12 : task 3
time 13 : task 3
time 14 : Idle
time 15 : Task 2
time 16 : Idle
time 17 : Idle
time 18 : Idle
time 19 : Idle

PAGE NO :
DATE :

C code to implement proportional scheduling

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

typedef struct {
    char name[5];
    int tickets;
} process;

int main() {
    int n, total_tickets=0;
    float total_T = 0.0;
    printf("Enter no.of processes");
    scanf("%d", &n);
    process P[n];
    srand (time(NULL));
    for (int i=0; i<n; i++) {
        printf("\n process %d:\n", i+1);
        sprintf(P[i].name, "P%d", i+1);
        printf("Tickets");
        scanf("%d", &P[i].tickets);
        total_tickets = p[i].tickets;
        total_tt += P[i].tickets;
    }

printf("\n--proportional share scheduling");
printf("Enter time period for scheduling:");
int m;
scanf("%d", &m);
```

PAGE NO :
DATE :

```
for (int i=0; i<m; i++) {
    int winning_ticket = rand() % total_ticke
    int accumalated_tickets = 0
    int winner_index;
    for (int j=0; j<n; j++) {
        accumalated_ticket += p[j].tickets;
        if (winning_ticket <= accumulated_tic
        {
            winner_index = j;
            break;
        }
    }

    printf("Tickets picked: %d, winner: %
        winning_ticket, p[winning_index].name
    }
    for (int i=0; i<n; i++) {
        printf("\n the process: %s gets %.0.2f%
            of processor Time \n", p[i].name,
            ((p[i].tickets / total_ticket)*100
    }
    return 0;
}
```

output:

Enter number of processes: 3
Process 1:
Tickets: 10
process 2:
Tickets: 20
process 3:
Tickets: 30
proportional share scheduling: 5
Tickets picked: 5 winner: P1
Tickets picked: 25 winner: P2

PAGE NO :
DATE :

Ticket picked : 35, winner : P_3
Ticket picked : 50, winner : P_3
Ticket picked : 15, winner : P_2
The process : P_1 gets 16.67 % of processor time.
The process : P_2 gets 33.33 % of process time
The process : P_3 gets 50.00 % of processor time.

6/6/24

PAGE NO :
DATE :

Priority scheduling (Non-premptive)

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

void swap(int *a, int *b){
    *a = *a + *b;
    *b = *a - *b;
    *a = *a - *b;
}

void sort(int *pid, int *at, int *bt, int *prior,
          int n){
  for(int i=0; i<n; i++){
      for(int j=0; j<n; j++){
          if( at[i]< at[j]){
              swap(&at[i], &at[j]);
              swap(&bt[i], &bt[j]);
              swap(&pid[i], &pid[j]);
              swap(&prior[i], &prior[j]);
          }
      }
  }
}

int highest_priority(int *prior, int s, int e){
    int x = prior[s];
    int j = s;
    for (int i = s; i<e; i++){
        if (prior[i]>x){
            x= prior[i];
            j=i;
        }
    }
```

DATE :

```
    return j;
}
int main(){
    int n, t, x;
    printf("Enter the number of processes:");
    scanf("%d", &n);
    int pid[n], at[n], bt[n], ct[n], tat[n], wt[n],
        bt2[n], rt[n], prior[n];
    for (int i=0; i<n; i++){
        printf("Enter arrival time & burst time &
                priority : ");
        scanf("%d %d %d", &at[i], &bt[i], &prior[i]);
        pid[i]=i+1;
    }
    sort(pid, at, bt, prior, n);
    for (int i=0; i<n; i++){
        bt2[i]= bt[i];
        rt[i]=-1;
    }
    int arvc=0;
    int count=0;
    int ctvar=at[0];
    int curi=0;
    while (count!=n){
        if (rt[curi]==-1){
            rt[curi]= ctvar - ar[curi];
        }
        if (arvc==n){
            ctvar+ = bt2[curi];
            bt2[curi]=0;
        }
        else{
            ctvar+ =1;
```

PAGE NO :
DATE :

```
    bt2[curi]--;
  }
  for(int i=0; at[i]<=ctvar; i++){
      arvct=1;
      x = i;
    }
  if (bt2[curi]==0){
      countt=1;
      ct[curi]=ctvar;
      prior[curi]=-1;
    }
    curi= highest_priority(prior, 0, x+1);
  }
  for (int i=0; i<n; i++){
    tat[i]= ct[i]-at[i];
    wt[i]= tat[i]-bt1[i];
  }
  float avg_tat=0;
  float avg_wt=0;
  for (int i=0; i<n; i++){
    avg_tat+= tat[i];
    avg_wt+=wt[i];
  }
  printf("pid \t at \t bt \n ct \n tat \t wt \t rt
  for (int i=0; i<n; i++){
    printf("%d \t %d \t %d \t %d \t %d \t %d \t",
        pid[i], at[i], bt1[i], ct[i], tat[i], wt[i],
    print ("\n");
  }
  printf("\n average TAT: %f", avg_tat/n);
  printf("\n average WT: %f", avg_wt/n);
  return 0;
}
```

PAGE NO :
DATE :

~~#include~~ write a c program to solve philoso-phers problem

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

sem_t* forks;
sem_t mutex;

void* philosopher_one(void* num){
    int id = *(int*) num;
    printf("%d is waiting\n", id);
    sem_wait(&mutex);
    printf("%d is granted to eat\n", id);
    sem_wait(&forks[id]);
    sem_wait(&forks[(id+1)%5]);
    sem_post(&mutex);
    sleep(1);

    sem_post(&forks[id]);
    sem_post(&forks[(id+1)%5]);
    return NULL;
}
void* philosopher_two(void* num){
    int id = *(int*)num;
    printf("%d is waiting\n", id);
    sem_wait(&forks[id]);
    sem_wait(&forks[(id+1)%5]);
    printf("%d is granted to eat\n", id);
    sleep(1);
```

```
    sem_post(&forks[id]);
    sem_post(&forks[(id+1)%5]);
    return NULL;
}
int main(){
    int num_philosophers, num_hungry;
    printf("Dining philosophers problem\n
    printf("Enter the total no.of philosophers
    scanf("%d", &numphilosophers: ");
    int hungry_positions[num_hungry];
    for (int i=0; i<num_hungry; i++){
        printf("Enter philosopher %d position
                i+1);
        scanf("%d", &num_hungry);
        hungry_positions[i]--;
    }

    forks = (sem_t*)malloc(num_philosophers
                sizeof(sem_t));
    pthread_t philosophers[num_hungry];
    int ids[num_hungry];

    for (int i=0; i<num_philosophers; i++){
        sem_init(&forks[i], 0, 1);
    }

    sem_init(&mutex, 0, 1);
    int choice;
    printf("\n 1. One can eat at a time \t
        2. Two can eat at a time \t 3. Exit \n
    printf("Enter your choice: ");
        scanf("%d", &choice);
    if(choice==1){
        printf("\nAllow one philosopher to
                eat at any time");
```

PAGE NO :
DATE :

```
for(int i=0; i<num_hungry; i++){
    ids[i]= hungry_position[i];
    pthread_create(&philosophers[i], NULL,
        Philosopher_one, &ids[i]);
  }
} else if (choice==2){
  printf("\n Allow two philosophers at
    a time \n");
  for (int i=0; i<num_hungry; i++){
    ids[i]= hungry_positions[i];
    pthread_create(&philosophers[i], NULL,
        philosopher_two, &ids[i]);
  }
} else {
  printf(" Exiting program \n");
  free(forks);
  return 0;
}
for (int i=0; i<num_hungry; i++){
  pthread_join(philosophers[i], NULL);
}
for (int i=0; i<num_philosophers; i++){
  sem_destroy(&forks[i]);
}
sem_destroy(&mutex);
free(forks);
return 0;
}
```

PAGE NO :
DATE :

Output

Dining philosopher problem.
Enter the total no. of philosophers: 5
How many are hungry: 5
Enter philosopher 1 position: 1
Enter philosopher 2 position: 2
Enter philosopher 3 position: 3
Enter philosopher 4 position: 4
Enter philosopher 5 position: 5

1. One can eat at a time, 2. two can eat at a time 3. exit
Enter your choice: 1
Allow one philosophers to eat at any time.
P0 is waiting
P0 is granted to eat
P2 is waiting
P2 is granted to eat
P3 is waiting
P3 is granted to eat
P4 is waiting
P1 is waiting
P4 is granted to eat
P1 is granted to eat.

3/7/24

PAGE NO :
DATE :

Q) write a c program to stimulate page replacement algorithms
a) FIFO
b) LRU
c) Optimal

```
#include <stdio.h>
#include <stdbool.h>
void FifoPageReplacement(int pages, int incomingStream[]
    int frames){
  int pageFaults=0;
  int temp[ ~~faultFram~~ frames];
  int front=0;
  for(int m=0; m<frames; m++){
      temp[m]=-1;
  }
  printf("\n Fifo Page Replacement Algorithm\n");
  printf("\n Incoming \t Frame 1\t Frame 2\t Frame
      3\t Frame \n");
  for (int m=0; m<pages; m++){
      bool pageFault= true;
      for(int n=0; n< Frames; n++){
        if( temp[n] == incomingStream[m]){
            pageFault= false;
            break;
        }
      }
      if (pageFault){
          temp[front]= incomingStream[m];
          front = (front+1) % frames;
          pageFaults++;
      }
```

PAGE NO :
DATE :

```
printf("%d \t \t", incomingStream[m]);
for(int n=0; n< frames; n++){
    if (temp[n] != -1)
        printf("%d \t \t", temp[n]);
    else
        printf ("- \t \t");
    }
    printf(" \n");
}
printf(" \n Total page Faults: \t %d \n",
                pageFaults);
}
void lruPageReplacement(int pages, int incomingSt
ream[], int frames){
int pageFaults = 0;
int temp[frames];
int used[frames];
for(int m=0; m< frames; m++){
    temp[m] = -1;
    used[m] = 0;
}
printf(" \n LRU pageReplacement Algorithm \n");
printf(" \n Incoming \t frame 1 \t frame 2 \t
    frame 3 \t frame ");
for(int m=0; m<pages; m++){
    bool pagefault= true;
    int leastUsedIdx = 0;
    for(int n=0; n<frames; n++){
        if (temp[n] == incomingStream[m]){
            pagefault= false;
            used[n]= m+1;
            break;
        }
```

```
        if(used[n] < used[leastUsedIdx]){
            leastUsedIdx = n;
        }
    }
    if (pageFault){
        temp[leastUsedIdx] = incomingStream[m];
        used[leastUsedIdx] = m+1;
        pageFaults++;
    }

    printf("%d \t \t", incomingStream[m]);
    for(int n=0; n < frames; n++){
        if (temp[n] != -1)
            printf("%d \t \t", temp[n]);
        else
            printf("- \t \t");
    }

    printf("\n");
}

printf("\n Total pageFaults: \t %d\n",
        pageFaults);
}

void optimalPageReplacement (int pages, int
    incomingStream[], int frames){
    int pageFaults = 0;
    int temp[frames];
    for(int m=0; m < frames; m++){
        temp[m] = -1;
    }

    printf("\n Optimal page Replacement \n");
    printf("\n Incoming \t Frame 1 \t Frame 2 \t
        Frame 3 \t Frame");
```

PAGE NO :
DATE :

```
for(int m=0; m<pages; m++){
    bool pageFault = true;
    int replaceIdx = -1;
    int farthest=m;
    for(int n=0; n<frames; n++){
        if (temp[n] == incomingStream[m]){
            pageFaults = false;
            break;
        }
    }

    if (pageFault){
        for(int n=0; n<frames; n++){
            int j;
            for(j=m+1; j<pages; j++){
                if(temp[n] == incomingStream[j]){
                    if (j> farthest) {
                        farthest=j;
                        replaceIdx = n;
                    }
                    break;
                }
            }

            if (j==pages){
                replaceIdx=n;
                break;
            }
        }

        if (replaceIdx == -1){
            replaceIdx=0;
        }
        temp[replaceIdx]= incomingStream[m];
        pageFaults++;
    }
```

```
printf("%d\t\t", incomingStream[m]);
for (int n=0; n< frames; n++){
    if (temp[n] != -1){
        printf("%d\t\t", temp[n]);
    else
        printf("-\t\t");
    }
    printf("\n");
}
printf("\n Total page Faults: \t %d\n",
                pagefaults);
}
int main(){
    int pages;
    printf("Enter number of pages in incoming
                stream: ");
    scanf("%d", &pages);
    int incomingStream[pages];
    printf("Enter the sequence of pages:\n");
    for (int i=0; i<pages; i++){
        scanf("%d", &incomingStream[i]);
    }
    int frames;
    printf("Enter no.of frames: ");
    scanf("%d", &frames);
    fifopageReplacement(pages, incomingStream, fram
    lrupageReplacement(pages, incomingStream, fram
    Optimal PageReplacement(pages, incomingStream, fra
    return 0;
}
```

Question:

Enter order of pages in incoming sequence

Enter the sequence of pages

0 9 0 1 8 1 8 7 8 7 1 8 0 8 4 7 8

2 3 8 2

Enter number of frames: 4

FIFO Page Replacement Algorithm

Incoming	Frame 1	Frame 2	Frame 3	Frame 4
0	0			
9		0	9	
0	0	0	9	
1	0	0	9	1
8	8	0	9	1
1	8	8	9	1
8	8	8	9	1
7	8	8	7	1
8	8	8	7	1
7	8	8	7	1
1	8	8	7	1
8	8	8	7	2
8	8	8	7	2
2	8	8	7	2
7	8	8	7	2
8	8	8	7	2
2	8	8	7	2
3	8	8	7	2
8	3	8	7	2
3	3	8	7	2

Total page faults: 8

PAGE NO :
DATE :

LRU page Replacement algorithm

Incoming	Frame1	Frame2	Frame3
0	0	–	–
9	0	9	–
0	0	9	–
1	0	9	1
8	0	8	1
1	0	8	1
8	0	8	1
7	7	8	1
8	7	8	1
7	7	8	1
1	7	8	1
2	7	2	1
8	8	2	1
2	8	2	1
7	8	2	7
8	8	2	7
2	8	2	7
3	8	2	3
8	8	2	3
3	8	2	3

Page faults: 9

PAGE NO :
DATE :

Optimal page Replacement algorithm

incoming	Frame 1	Frame 2	Frame 3
0	0	—	—
9	0	9	—
0	0	9	—
1	1	9	—
8	1	8	—
1	1	8	—
8	1	8	—
7	1	8	7
8	1	8	7
7	1	8	7
1	1	8	7
2	2	8	7
8	2	8	7
2	2	8	7
7	2	8	7
8	2	8	7
2	2	8	7
3	3	8	7
8	3	8	7
3	3	8	7

Total page faults: 7

10/7/24

PAGE NO :
DATE :

Q) C code for implementing multi level queue process scheduling

```
#include <stdio.h>
#include <stdlib.h>
# define NUM_QUEUES 3
# define QUANTUM 3

typedef struct {
    int id;
    int priority;
    int burst_time;
    int remaining_time;
} process;

typedef struct {
    process *queue[100];
    int front;
    int rear;
} Queue;
Queue *queues[NUM_QUEUES];

void init_queues() {
    for(int i=0; i<NUM_QUEUES; i++){
        queues[i]= (Queue*)malloc(sizeof(Queue
        queues[i]-> front = -1;
        queues[i]-> rear = -1;
    }
void enqueue(process *p) {
    int priority = p-> priority;
    queues[priority]-> rear++;
    queues[priority]-> queue[queues[priority]-re
```

PAGE NO :
DATE :

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

struct proc{
    int id;
    int bt;
    int at;
    int qt;
    int wt;
    int tat;
    int rt;
    int st;
    int et;
    int vis;
};
void main(){
    int n, currTime=0;
    float avgWT=0, avgTAT=0, avgRT=0, tp;
    printf("Queue 1 is system process\n Queue
        2 is user process \n");
    printf("Enter number of processes: ");
    scanf("%d",&n);

    struct proc procs[n], tmp;
    for (int i=0; i<n; i++){
        procs[i].id= i+1;
        procs[i].vis=0;
        printf("Enter Burst time, Arrival Time and
            Queue of P%d: ", i+1);
        scanf("%d %d %d", &procs[i].bt, &procs[i].at,
            &procs[i].qt);
    }
```

```
for (int i=0; i<n; i++){
    for (int j= i+1; j<n; j++){
        if (procs[i].at > procs[j].at ||
            (procs[i].at == procs[j].at &&
                procs[i].qt)){
                tmp = procs[i];
                procs[i]= procs[j];
                procs[j]= tmp;
            }
        }
    }

int compTime= procs[0].bt, minQT= INT_MAX
    minQI, procCount= 0;
procs[0].vis= 1;
while (procCount <n-1){
    for(int i=0; i<n; i++){
        if(procs[i].at<= compTime && procs[i].
            vis==0){
            if(procs[i].qt < minQT){
                minQT = procs[i].qt;
                minQI= i;
            }
        }
    }

procs[minQI].vis= 1;
compTime += procs[minQI].bt;
minQT = INT_MAX;
tmp= procs[procCount+1];
procs[procCount+1]= procs[minQI];
procs[minQI]= tmp;
procCount++;
}
```

PAGE NO :
DATE :

```
printf("\n process \t WT \t TAT \t RT \n");
for(int i=0; i<n; i++) {
    if (procs[i].at < curTime) {
        procs[i].st = curTime;
    } else {
        procs[i].st = procs[i].at;
    }

    procs[i].et = procs[i].st + procs[i].bt;
    curTime += procs[i].bt;
    procs[i].tat = procs[i].et - procs[i].at;
    procs[i].wt = procs[i].tat - procs[i].bt;
    procs[i].rt = procs[i].st - procs[i].at;
    printf("%d \t %d \t %d \t ", procs[i].id,
        procs[i].wt, procs[i].tat, procs[i].rt);
    avgWT += procs[i].wt;
    avgTAT += procs[i].tat;
    avgRT += procs[i].rt;
}

tp = (float) procs[n-1].et / n;
printf("Avg WT: %.2f \n", avgWT/n);
printf("Avg TAT: %.2f \n", avgTAT/n);
printf("Avg RT: %.2f \n", avgRT/n);
printf("Throughput: %.2f", tp);
}
```

output

```
Enter no. of process : 4
Enter Burst Time, Arrival Time & Queue :
4   0   1
3   0   1
5   0   2
5   10  1
```

Avg WT: 3.25
Avg TAT: 7.5
Avg RT: 3.25
Throughput: 7.25

e)

5/6/24

PAGE NO.
DATE

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

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

typedef struct {
    int size;
    int isAllocated;
} partition;

typedef struct {
    int size;
    int isAllocated;
    int partitionIndex;
} process;

void allocate(Process processes[], int process
-Count, Partition partitions[], int partition
-ct, int (*findIndex)(Partition [], int, int
int totalFragmentation=0;
printf("\n File NO. \t process size \t
        Block NO. \t Block size \t Fragment
for (int i=0; i<ProcessCount; i++) {
    int index= findIndex(partitions, partition
        processes[i].size);
    if (index != -1) {
```

PAGE NO :
DATE :

```
            process[i].isAllocated = 1;
            process[i].partitionIndex = index;
            partions[index].isAllocated = 1;
            int fragmentation = partions[index].size -
                process[i].size;
            totalFragmentation += fragmentation;
            printf("%d \t %d \t %d \t %d \t %d \n",
                i, process[i].size, index, partitions[index].
                size, fragmentation);
        } else {
            printf("%d \t %d \t - \t Process could
                not be allocated \n", i, ~~process[i]~~
                processes[i].size);
        }
    }

    printf("\n Total Fragmentation: %d \n",
        totalFragmentation);
}

int firstFitIndex(partition partions[], int
    partitionCount, int processSize) {
    for (int i=0; i < partitionCount; i++) {
        if (!partitions[i].isAllocated &&
            partitions[i].size >= processSize) {
                return i;
        }
    }

    return -1;
}

int ~~best~~ worst FitIndex(partition partitions[], int
    partitionCount, int processSize) {
    int worstIndex = -1;
    for (int i=0; i < partitionCount; i++) {
        if (!partitions[i].isAllocated &&
```

```
partitions[i].size >= processSize) {
        if (worstIndex == -1 || partition[i].size
        > partition[worstIndex].size) {
            worstIndex = i;
        }
    }
}
return worstIndex;
}

int bestFitIndex(partition partitions[], int
    partitionCount, int processSize) {
    int bestIndex = -1;
    ~~for(!partitions[i].~~
    for( int i=0; i<partitionCount;
        if (!partition[i].isAllocated &&
            partitions[i].size >= processSize) {
            if (bestIndex == -1 || partitions[i].size
                < partitions[bestIndex].size) {
                bestIndex = i;
            }
        }
    }
    return bestIndex;
}
void reset(partition partitions[], int partition
    Count, process processes[], int processCount)
    for(int i=0; i< partitionCount; i++)
        partitions[i].isAllocated=0;
    for(int i=0; i< processCount; i++) {
        process[i].isAllocated = 0;
        process[i].partitionIndex = -1;
    }
}
```

PAGE NO:
DATE:

```
int main() {
    int partitionCount, processCount, choice,
    alreadyAllocated;
    partition partitions[MAX_PARTITIONS];
    process processes[MAX_PROCESSES];
    printf("Enter the number of partitions:
    scanf("%d", &partitionCount);
    printf("Enter size of each partition:\n");
    for(int i=0; i< partitionCount; i++) {
        printf("partition %d: ", i);
        scanf("%d", &partitions[i].size);
        partitions[i].isAllocated = 0;
    }

    printf("Enter no. of Processes: ");
    scanf("%d", &processCount);
    printf("Enter the size of each process:\n
        for(int i=0; i< processCount; i++) {
            printf("Process %d ", i);
            scanf("%d", &processes[i].size);
            processes[i].isAllocated = 0;
            processes[i].partitionIndex = -1;
        }

    printf("Choose memory allocation
            statergy: \n 1. FirstFit \n2.
    Beast-Fit \n3. worst_fit \n");
    scanf("%d", &choice);
    int (*allocationStatergy)(partition[], int,
        int) = NULL;
    switch(choice) {
        case 1: allocationStatergy= firstFitIndex;
                break;
        case 2: allocationStatergy = bestFitIndex;
                break;
```

```
case 3: allocationStrategy = WORSTFIT;
        break;
default: printf("Invalid Choice\n");
        return;
}
printf("Memory Management scheme");
printf("File no\t File size\t Block no\t Block size\n");
allocate(processes, process count, partitions, partition count, allocationStrategy);
}
```

output

```
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of blocks:
Block1: 5
Block2: 2
Block3: 7
Enter the size of files:
File 1: 1
File 2: 4
```

Memory Management scheme - First Fit

File no:	File size	Block no:	Block size	Fragment
1	1	1	5	4
2	4	3	7	3

memory management scheme - worst fit

File no	File size	Block no	Block size	Fragment
1	1	3	7	6
2	4	1	5	1

memory management Scheme - Best fit

File no	File size	Block no	Block size	Fragment
1	1	9	9	1
2	4	1	5	1

PAGE NO :
DATE :

8) C code for Earliest deadline first;

```
# include <stdlib.h>
# include <stdio.h>
# define MAX_tasks 10
typedef struct {
    int p;
    int c;
    int d;
    int rt;
    int nd;
    int id;
} Task;

void Input (TASK tsks[], int *n_tsk) {
    printf("Enter no.of tasks (Max %d);", MAX_Ts
                                          ks);
    scanf("%d", n_tsk);
    if (*n_tsk > MAX_TSKS){
        printf("no. of tasks exceeds the max limit
            of %d. \n", MAX_TSKS);
            exit (EXIT_FAILURE);
    }
    for (int i=0; i<*n_tsk; i++) {
        tsks [i].id = i+1;
        printf(" Enter period of task %d:", i+1);
        scanf("%d", &tsks [i].p);
        printf(" Enter Execution time of tasks %d:
                i+1);
        scanf("%d", &tsks[i].c);
        printf("Enter deadline of task %d:", i+1);
        scanf("%d", &tasks[i].d);
        tasks [i].rt = tasks[i].c;
        tasks [i].nd = tsks[i].d;
    }
```

PAGE NO :
DATE :

```
void EDF (Task tsks[], int n_tsk, int tf){
    printf("\n Earliest_deadline first \n");
    for (int t=0; t<tf; t++){
        int s_tsk = -1;
        for (int i=0; i<n_tsk; i++){
            if ( t % tsks[i].P == 0){
                tasks[i].rt = tsks[i].c;
                tsks[i].nd = t + tsks[i].d;
                }
        }

        for (int i=0; i<n_tsks; i++){
            if (tsks[i].rt > 0 && (s_tsk==-1 ||
                tsks[i].nd < tsks[s_tsk].nd)){
                    s_tsk = i;
                }
        }

        if (s_tsk != -1){
            printf(" Time %d; Task %d \n", t,
                    tasks [s_tsk].id);
            tsks [s_tsk].rt--;
        } else{
        ~~if ts~~  printf(" Time %d; Idle \n");
        }
    }
}

int main(){
    Task tsks[MAX_TSKS];
    int n_tsk;
    int tf;
    Input( tsks, &n_tsk);
    Printf(" Enter time frame for simulation
    scanf("%d", &tf);
    EDF(
```

PAGE NO :
DATE :

Enter no. of tasks: 3
Enter T of tasks 1: 20
Enter Execution time of task : 3
Enter T of task 2: 5
Enter execution time of task 2: 2
Enter T: of task 3: 10
Enter execution time of task 3: 2

time 0 : task 2
time 1 : task 2
time 2 : task 3
tim 3 : task 3
time 4 : task 1
time 5 : task 2

5/6/24

PAGE NO :
DATE :

Q) Write C code to implement deadlock detection program.

```
# include <stdio.h>
# include <stdbool.h>
# define NUM_PROCESSES 5
# define NUM_RESOURCES 3

int available[NUM_Resources];
int allocation[NUM_Resources][Num_Resources];
int request[Num_PROCESSES][NUM_RESOURCES];

bool deadlockDetection(int *safeSequence){
    int work[NUM_RESOURCES];
    bool finish[NUM_PROCESSES];
    for(int i=0; i<NUM_RESOURCES; i++){
        work[i]= available[i];
    }

    int count=0;
    while(count<NUM_processes){
        bool found=false;
        for(int i=0; i<NUM_processes; i++){
            if(!finish[i]){
                bool canProceed=true;
            for(int j=0; j<NUM_RESOURCES;
                j++){
            if(request[i][j]>work[j]){
                canProceed=false;
                break;
            }
        }

    if(canProceed){
        for(int j=0; j<NUM_RESOURCES; j++){
            work[j]+= allocation[i][j];
        }
```

PAGE NO:
DATE:

```
            safeSequence[count++] = i;
            finish[i] = true;
            found = false;
        }
    }
}
if(!found){
    break;
}
}
for (int i=0; i<NUM_PROCESSES; i++){
    if (!finish[i]){
        printf("Deadlock detected. Process
            P%d is in deadlock. \n", i);
        return false;
    }
}
printf("No deadlock detected. The System
        is in a safe state. \n");
printf("\n");
return true;
}
int main(){
    int i, j;
    printf("Enter the available Resources
    for (i=0; i<NUM_RESOURCES; i++){
        scanf("%d", &available[i]);
    }
    printf("Enter Allocation Matrix: \n");
    for (i=0; i<NUM_Processes; i++){
        for (j=0; j<NUM_RESOURCES; j++){
            scanf("%d", &allocation[i][j]
        }
    }
```

PAGE NO :
DATE :

```
printf("Enter the Request matrix:\n");
for(i=0; i<NUM_processes; i++){
    for(j=0; j<NUM_RESOURCES; j++){
        scanf("%d", &request[i][j]);
    }
}

int safeSequence[NUM_processes];
deadlockDetection(safeSequence);
return 0;
}
```

Output

```
Enter the available Resource vector:
3 3 2
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the Request matrix:
7 4 3
1 2 2
6 0 0
0 1 1
0 0 2
No Deadlock detected: The system is in a safe state
Safe sequence: P1 P3 P4 P0 P2
```

PAGE NO :
DATE :

Q1) C code to implement Bankers algorithm

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

void calculatedNeed(int need [MAX_PROCESSES]
        [MAX_RESOURCES], int max [MAXPROCESSES]
        [MAX_RESOURCES], int allot [MAX PROCESSES]
        [MAX_RESOURCES], int np, int nr) {
    for (int i=0; i<np; i++) {
        for (int j=0; j<nr; j++) {
            need [i][j] = max [i][j] - allot[i][j];
        }
    }
}

bool isSafe(int processes[], int avail[], int
    max[][MAX_RESOURCES], int allot[][MAX_
    RESOURCES], int np, int nr);
    bool finish [MAX_PROCESSES] = {0};
    int safeSeq [MAX_PROCESSES];
    int work [MAX_RESOURCES];
    for (int i=0; i<nr; i++)
        work[i] = avail [i];
    int count = 0;
    while (count < np) {
        bool found = false;
        for (int p=0; p<np; p++) {
            if (finish [p] == 0) {
                int j;
                for (j=0; j<nr; j++)
                    if (need [p][j] > work[j])
                        break;
```

PAGE NO :
DATE :

```
        if(j==nr) {
            for (int k=0; k<nr; k++)
                work[k] += allot[p][k];

            safeseq[count++]=p;
            finish[p]=1;
            found = true;
        }
    }
}
if (found== false) {
    printf("System is not in safe state\n"
    return false;
}
}
printf("System is in safe state. \n Safe sequen
 is :");
for (int i=0; i<np; i++)
    printf("%d", safeSeq[i]);
printf("\n");
return true;
}.
int main() {
    int np, nr;
    printf("Enter number of processes:");
    scanf("%d", &np);
    printf("Enter number of resource types:")
    scanf("%d", &nr);
    int processes[MAX_PROCESS];
    for (int i=0; i<np; i++)
        processes[i] = i;
    int avail[MAX_RESOURCES];
```

```
printf("Enter available resources:");
for(int i=0; i<nr; i++)
    scanf("%d", &avail[i]);
int max[max_PROCESSES][MAX_RESOURCES];
printf("Enter maximum resource matrix:
        \n");
for(int i=0; i<np; i++) {
    printf("Process %d:", i);
    for(int j=0; j<nr; j++)
        scanf("%d", &allot[i][j]);
}
isSafe(processes, avail, max, allot,
        np, nr);
return 0;
}
```

Output:

```
Enter the number of process: 6
Enter number of resource: 3
Enter available resource matrix: 3 3 2
Enter maximum resource 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 allocation resource matrix:
Process 0: 0 1 0
Process 1 : 2 0 0
Process 2 : 3 0 2
Process 3 : 2 1 1
Process 4 : 0 0 2
```

DATE :

system is in safe state.
safe ~~stack~~ sequence is: 1 3 4 0 2.

8)

3/7/24