Output

PN	BT	Wt	TAT
1	10	13	23
2	5	10	15
3	8	13	21

Average waiting Time = 12

Average Turn Around Time = 19.6667

Date

Exp. No. 3

Page No.

(3) Write a process scheduling using Round Robin (RR) scheduling Algorithm.

```
using system.
public class CsPh {
    Static void find waiting Time( int[] processes),
        int n, int[] bt, in[] wt, int quantum
{
 int [] rem_bt = new int [n];
 for ( int i=0 ; i<n ; i++)
   rem_btt[i] = bt[i];
 int t = 0;
 while (true)
 {
 bool done = true ;
 for ( int i=0 ; i<n ; i++)
 {
 if ( rem_bt [i] >0)
 {
  done = false
 if( rem_bt[i]) > quantum
 {
   t += quantum
   rem_bt[i] = quantum ;
 }
 else
 {
```

JAI BHARAT

Teacher's Signature

```
            t = t + rem_bt[i];
            wt[i] = t - bt[i];
            rem_bt[i] = 0;
        }
    }
    if (done == true)
        break;
    }
}
static void findTurnAroundTime(int[] processes,
    int n, int[] bt, int[] wt, int[] tat)
{
    for (int i=0; i<n; i++)
        tat[i] = bt[i] + wt[i];
}
    int[] wt = new int[n];
    int[] tat = new int[n];
    int wt = 0, total_tat = 0;

findWaitingTime(processes, n, bt, wt, quantum);

findTurnAroundTime(processes, n, bt, wt, tat);
Console.WriteLine("Processes " + "Burst Time " + "Waiting time" +
                  "Turn around time");
for(int i=0; i<n; i++)
{
```

```
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];

    Console.WriteLine("" " + (i+1) + " \t \t "+ bt[i]
                + "\t" + wt[i] + " \t \t "+ tat[i]);
    }
Console.WriteLine("Average waiting time = " + (float)total_wt/
                (float)n);
Console.Write("Average turn around time = "+
            (float) total_tat /(float)n);

public static void ()}
    int [] processes = {1, 2, 3}
    int n = processes.Length;
    int [] burst_time = {10, 5, 8};
    int quantum = 2;
    findavgTime(processes, n, burst_time, quantum){
    }
}
```

Date

Expt. No. 4 Page No.

(4.) W.a. process Scheduling code using Priority Scheduling Algorithm.

```
Using System;
using System.Collections.Generic;

Class Process
{
  public int pid;  // process ID

  public int bt;
  public int priority;
  public process( int pid, int bt, int priority){
      this.pid = pid;
      this.bt = bt;
      this.priority = priority;
  }
    public int Prior{
      get { return priority; }
    }
  }

Class GFG {
    public void findWaitingTime ( Process[] proc, int n,
          int[] wt)

  {
    for (int i = 1; i<n; i++)
    wt[i] = proc[i-1].bt + wt[i-1];
```

JKI BHARAT

Teacher's Signature

Date

Expt. No. Page No.

```
public void
findturnAroundTime ( Process [] proc , int n, int[] wt,
    int[] tat) {
    for ( int i=0;  i<n ; i++)
      tat[i] =  proc[i].bt + wt [i];
    }
    public void findavgTime (process [] proc, int n) {
        int [] wt = new  int[n];
        int[ ] tat = new int[n];
        int  total_wt  = 0 ;
        int  total_tat = 0;

findWaitingTime ( proc , n, wt);
 findTurnAroundTime(proc, n,  wt, tat);

console.WriteLine(" \n Proceses Burst time
   waiting time & total turn Around time ");

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

Console.WriteLine ("   "+ proc[i].pid +  "\t \t "+
   proc[i].bt  + "\t" + wt[i] + "\t \t " +tat[i]);
  }
Console. WriteLine (" \n Average waiting time = "
      +(float) total_wt / (float) n);
```

JAI BHARAT

Teacher's Signature

Date

Expt. No. Page No.

```
console.WriteLine("Average turn around time=" +
    (float)totwt/(float)n);
}
public void priorityScheduling(Process[] proc, int n){
    Array.Sort(proc, new Comparison<Process>
    ((a,b) => b.Prior.CompareTo(a.Prior)));

Console.WriteLine("order in which processes gets
        executed");
  for(int i=0; i<n; i++)
  console.Write(proc[i].pid + " ");
findAvgTime(proc,n);
}
  static void Main(String[] args)
{
    GFG ob = new GFG();
    int n=3;
  Process[] proc = new Process[n];
    proc[0] = new Process(1,10,2);
    proc[1] = new Process(2,5,0);
    proc[2] = new Process(3,8,1);

    ob.priorityScheduling(proc,n);
}
}
```

JAI BHARAT

Teacher's Signature

Date

Expt. No. 5

Page No.

(5.) To implement deadlock avoidance & prevention by using Banker's algorithm.

```
# include <iostream>
# include <vector>
using namespace std;
bool isSafe(const vector<vector<int>>& max,
        const vector<vector<int>>& need, const
        vector<int>& allocation, const vector<int>&
        available) {
    int n = max.size();
    for(int i=0; i<n; ++i) {
      bool finished = true;
      for(int j=0; j<max[i].size(); ++j) {
        if(need[i][j] > available[j]) {
          finished = false;
      }
    }
    if(finished) {
      for(int j=0; j<max[i].size(); ++j) {
        available[j] += allocation[i][j];
        need[i][j] = max[i][j];
      }
      return true;
    }
  }
  return false;
}
```

JAI BHARAT

Teacher's Signature

Date

Expt. No.

Page No.

```
int main(){
  int n, m;
  cout << "Number of process:";
  cin >> n;
  cout << "number of resources:";
  cin >> m;
  vector<vector<int>> max(n, vector<int>(m));
  vector<vector<int>> need(n, vector<int>(m));
  vector<int> allocation(n*m);
  vector<int> available(m);

  cout << "Available resources:";
  for(int i=0; i<m; ++i){
    cin >> available[i];
  }
  for(int i=0; i<n; ++i){
    cout << "Process " << i << ":" << endl;
    cout << "Max:";
    for(int j=0; j<m; ++j){
      cin >> max[i][j];
    }
    cout << "Allocation:";
    for(int j=0; j<m; ++j){
      cin >> allocation[i*m+j];
      need[i][j] = max[i][j] - allocation[i*m+j];
    }
  }
```

JAI BHARAT

Teacher's Signature

Date

Expt. No.

Page No.

```
if (isSafe(max, need, allocation, available)){
 cout << "System is safe." <<endl;
} else {
cout << "System is not safe." <<endl;
}
return 0;
}
```

JAI BHARAT

Teacher's Signature

Date

Expt. No. 6

Page No.

(6.) To implement page replacement algorithms FIFO (First In First Out).

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

int capacity;
int pages[MAX_CAPACITY];
int page_count = 0;

void initialize() {
  printf("Enter the capacity of the page replacement algorithm");
  scanf("%d", &capacity);
}
void page_fault(int page) {
  if (page_count < capacity) {
    pages[page_count++] = page;
  } else {
  for (int i=0; i<capacity-1; ++i) {
    pages[i] = pages[i+1];
  }
  pages[capacity-1] = page;
}
}
int main() {
    initialize();
```


Teacher's Signature

Date

Expt. No.

Page No.

```
int page;
printf("Enter the page sequence (enter -1 to
        end);");
while(1){
    scanf("%d", &page);
    if (page == -1)
      break;
    }
    page_fault + (page);
  }
printf("Pages in memory after page faults:");
  for(int i=0; i<page-count; ++i){
    printf("%d"; pages[i]);
  }
   return 0;
}
```

JAI BHARAT

Teacher's Signature

Date

Expt. No. 7

Page No.

7.) To implement page replacement algorithm (LRU based).

```
# include <stdio.h>
# include <stdlib.h>
# define MAX_PAGES 100
# define MAX_FRAMES 3
  int find LRU ( int time [] int n) {
    int minimum = time [0];
        pos = 0;
  for (int i=1 ; i<n ; i++ ){
      if (time [i] < minimum ) {
          minimum = time [i] ;
              pos = i ; }
void LRU page faults ( int pages[] ; int n; int
    capacity ) {
 int frames [MAX_FRAMES];
 int time[ MAX_FRAMES];
    int page faults = 0;
 for ( int i = 0 ; i< MAX_FRAMES ; i++) {
      frames[i] = -1;
        time[i]=0 ; }
   for ( int i=0 ; i<n ; i++){
        int pages = pages[i] ;
        int hit =0
   for ( int j=0 ; j < capacity ; j++) {
      if (frames [j] = page) {
            hit = 1;
```

JAI BHARAT

Teacher's Signature

Date

Expt. No. Page No.

```
        time[j] = i;
        break; }
    }
    if (!hit) {
        int pos = findLRU(time, capacity);
        frames[pos] = page;
        time[pos] = i;
        pageFaults++; } }

printf("Total page faults : %d\n", pageFaults);
    int main() {
    int pages[] = {2,3,1,4,6,2,9,1,3,7};
    int n = sizeof(pages)/sizeof(pages[0]);
        int capacity = 3;
    LRUpageFaults(pages, n, capacity);
    return 0; }
```

JAI BHARAT

Teacher's Signature

Date

Expt. No. 08

Page No.

(8.) To implement page replacement algorithm (optimal based)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_PAGES 100
#define MAX_FRAMES 3
#define INF 9999
int main(){
  int pages[]= {2,3,1,4,6,2,9,1,3,7};
int n = sizeof(pages)/sizeof(Pages[0]);
   int capacity = 3;
optimalPagefaults(page, n, capacity);
     return 0;
int optimalPagefaults(int Pages[], int n, int capacity){
int frames[MAX_FRAMES];
     int Pagefaults = 0;
for(int i=0; i< MAX_FRAMES; i++){
   frames[i] = -1;}
for(int i=0; i<n; i++){
     int page = page[i];
       int hit = 0;
  for(int j=0; j<= capacity; j++){
       if(frameo[j]==pages){
            hit =1
            break;}
       }
```

JAI BHARAT

Teacher's Signature

Output :

Enter virtual address (page number and offset) : 3 1024

Physical address : 14336

Date

Expt. No. 9.

Page No.

(9.) W.a.p. for paging technique (memory management policy)

```
# include <stdio.h>
# define PAGE_SIZE 4096
# define NUM_PAGES 1024
# define MEMORY_SIZE (PAGE_SIZE * NUM_PAGES)

int main () {
    int memory [MEMORY_SIZE];
    int pageNum, offset, physicalAddress;

// Input : Virtual Address.
printf ("Enter virtual address (page number and offset): ");
scanf ("%d %d", &pageNum, &offset);

// Paging Technique : Convert virtual address to physical address.
if (pageNum >= 0 && pageNum < NUM_PAGES &&
    offset >= 0 && offset < PAGE_SIZE) {
physicalAddress = pageNum * PAGE_SIZE + offset;
printf ("Physical address : %d \n", physicalAddress);
} else {
    printf ("Invalid virtual address \n"); }

    return 0;
}
```

JAI BHARAT

Teacher's Signature

Output :

Enter logical address (segment number & offset) :

2: 512

Physical address : 2560

Date

Expt. No. 10

Page No.

(10.) w.a.p. for memory management policy algorithm (segmentation).

```
# include <stdio.h>
# define SEGMENT_SIZE 1024
# define NUM_SEGMENTS 4
# define MEMORY_SIZE (SEGMENT_SIZE * NUM_SEGMENTS)

int main() {
    int memory [MEMORY_SIZE];
    int segmentNum, offset, physicalAddress;

// Input: Logical Address.
    printf ("Enter logical address (segment number & offset): ");
    scanf ("%d %d", &segmentNum, &offset);

// Segmentation: Convert logical address to physical address.
    if (segmentNum >= 0 && segmentNum < NUM_SEGMENTS
        && offset >= 0 && offset < SEGMENT_SIZE) {
        physicalAddress = segmentNum * SEGMENT_SIZE
                          + offset;
        printf ("Physical address: %d \n", physicalAddress);
    } else {
        printf ("Invalid logical address \n");
    }
    return 0;
}
```

JAI BHARAT

Teacher's Signature

Output:

```
Enter words to be written to the file
(enter 'exit' to finish):
Hello
World
exit
File written successfully.
```

Date

Expt. No. 11

Page No.

(11.) Write a C progoom to implement index file allocation.

```
#include <stdio.h>
#include <stlib.h>
#define FILE-SIZE 1624

int main()$
    FILE * file;
    char word[20];

  //open a file for writing
   file = fopen("Sequential-file.txt", "w");

   if (file == NULL)$
     printf("Error openig file \n");
     exit(1);
   }
  // Input: Words to be written to the file
   printf("Enter words to be written to the file
         (enter 'exit' to finish): \n");
  while(1) {
     scanf("%s", word);
     if (strcmp(word, "exit")==0)$
     break;
     }
    fprintf(file, "%s", word);
  }
```


Teacher's Signature

Date

Expt. No. Page No.

```
// Close the file
fclose(file);
printf("File written successfully\n");
return 0;
}
```

JAI BHARAT

Teacher's Signature

Output:

```
Enter the number of records : 3
Enter the index table (key and offset):
1   100
2   200
3   300
Record found at offset 200
```

Date

Expt. No. 12 Page No.

(12.) Write a C program to implement index file allocation.

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

struct IndexEntry {
    int key;
    long offset;
};
int main() {
    FILE *file;
    struct IndexEntry indexTable[10];
    int i, numRecords, searchKey;

// Open a file for writing
    fil = fopen("index_file.txt", "w");

    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }
// Input : Number of records & index table
    printf("Enter the number of records: ");
    scanf("%d", &numRecords);

    for (i=0; i<numRecords; i++) {
        if (indexTable[i].key == searchKey) {
```

JAI BHARAT

Teacher's Signature

```
            printf("Record found at offset %Id \n",
                indexTable[i].offset);
                break;
            }
        }
        if(i == numRecords){
            printf("Record not found \n");
        }
        return 0;
    }
```

Output:

```
Enter values for the Linked List (enter -1 to finish):
1
2
3
4
-1
Linked List written to file successfully.
```

Date

Expt. No. 13 Page No.

(13.) Write a C program to implement link file allocation method.

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

struct Node {
    int data
    struct Node *next; };

int main() {
    FILE *file;
    struct Node *head = NULL, *current, *newNode;
    int value;

    // Open a file for writing
    file = fopen("linked_file.bin", "wb");

    if(file == NULL) {
        printf("Error opening file \n");
    exit(1);
    }
    // Input : Linked list values
    printf("Enter values for the linked list (enter -1
        to finish) : \n");

    while(1) {
        scanf("%d", &value);
```

JAI BHARAT

Teacher's Signature

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

// Create a new node
newNode = (struct Node *)malloc(sizeof(struct Node));
newNode -> data = value;
newNode -> next = NULL;

// Add the node to the linked list
if (head == NULL) {
    head = newNode;
    current = newNode;
} else {
    current -> next = newNode;
    current = newNode;
}
}
// Write the linked list to the file
fwrite(head, sizeof(struct Node), 1, file);

// Close the file
fclose(file);

printf("Linked list written to file successfully \n");
return();
}
```