TRIBHUVAN UNIVERSITY
Faculty of Management
Shanker Dev Campus

A LAB REPORT

Submitted By:

Name: Ashesh Neupane

TU Exam Roll No: 16295 / 23

Level: Bachelor

Semester: Fourth

Batch: 2080

Subject: Operating System

Submitted To:

Department of BIM

Shanker Dev Campus

Verified By :

Table of Contents

Optimal Page Replacement Algorithm ---------- 1
Round Robin (RR) CPU Scheduling Algorithm ---------- 4
Least Recently Used (LRU) Page Replacement Algorithm ---------- 7
Worst Fit Memory Allocation Strategy ---------- 10
Non-Preemptive Priority Scheduling ---------- 12
Shortest Seek Time First (SSTF) Disk Scheduling Algorithm ---------- 15
FIFO Page Replacement Algorithm ---------- 17
First-Come, First-Served (FCFS) Scheduling Algorithm ---------- 19
C-SCAN (Circular SCAN) Disk Scheduling Algorithm ---------- 22
LOOK Disk Scheduling Algorithm ---------- 25
SCAN Disk Scheduling Algorithm ---------- 27
Best Fit Memory Allocation Algorithm ---------- 30
Shortest Remaining Time First (SRTF) Algorithm ---------- 32

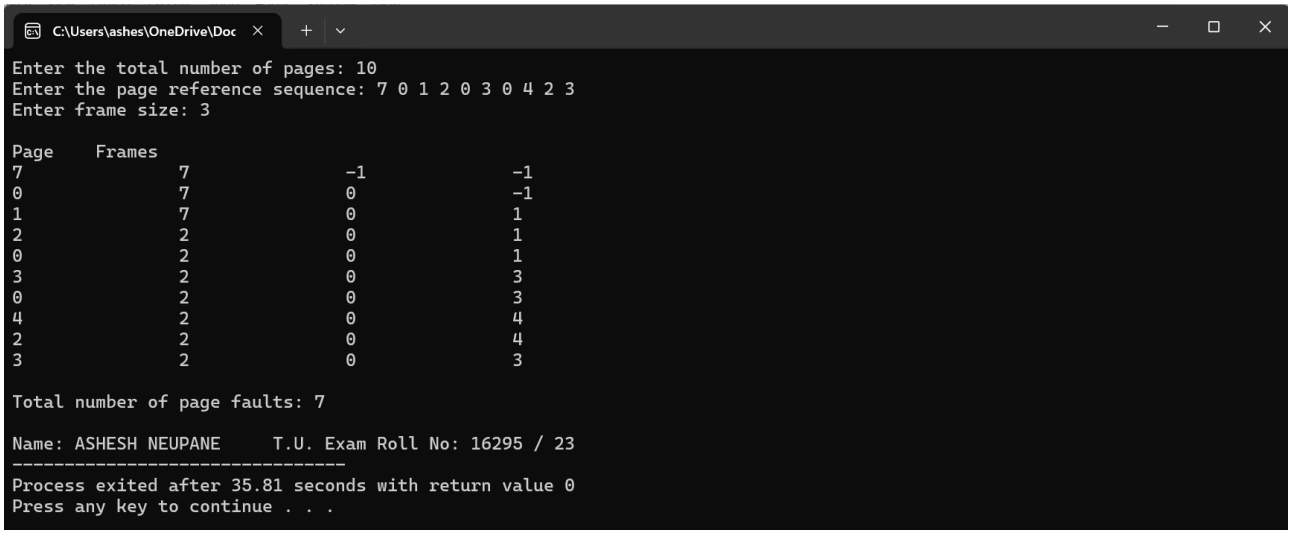
Write a program in C to implement the Optimal Page Replacement Algorithm.

Source Code:

```
#include <stdio.h>
int main() {
   int n, pg[30], fr[10];
   int count[10], i, j, k, fault, f, flag, temp, current, c, dist, max, m, cnt, p, x;
   fault = 0;
   dist = 0;
   k = 0;
   printf("Enter the total number of pages: ");
   scanf("%d", &n);
   printf("Enter the page reference sequence: ");
   for (i = 0; i < n; i++)
      scanf("%d", &pg[i]);
   printf("Enter frame size: ");
   scanf("%d", &f);
   for (i = 0; i < f; i++) {
      count[i] = 0;
      fr[i] = -1;
   }
   printf("\nPage\tFrames\n");
   for (i = 0; i < n; i++) {
      flag = 0;
      temp = pg[i];
      for (j = 0; j < f; j++) {
         if (temp == fr[j]) {
            flag = 1;
            break;
         }
      }
      if ((flag == 0) && (k < f)) {
         fault++;
         fr[k] = temp;
         k++;
```

```
        } else if ((flag == 0) && (k == f)) {
            fault++;
            for (cnt = 0; cnt < f; cnt++) {
                current = fr[cnt];
                for (c = i; c < n; c++) {
                    if (current != pg[c])
                        count[cnt]++;
                    else
                        break;
                }
            }
            max = 0;
            for (m = 0; m < f; m++) {
                if (count[m] > max) {
                    max = count[m];
                    p = m;
                }
            }
            fr[p] = temp;
        }
        printf("%d\t\t", pg[i]);
        for (x = 0; x < f; x++) {
            printf("%d\t\t", fr[x]);
        }
        printf("\n");
    }
    printf("\nTotal number of page faults: %d\n", fault);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0;
}
```

Output Screen:



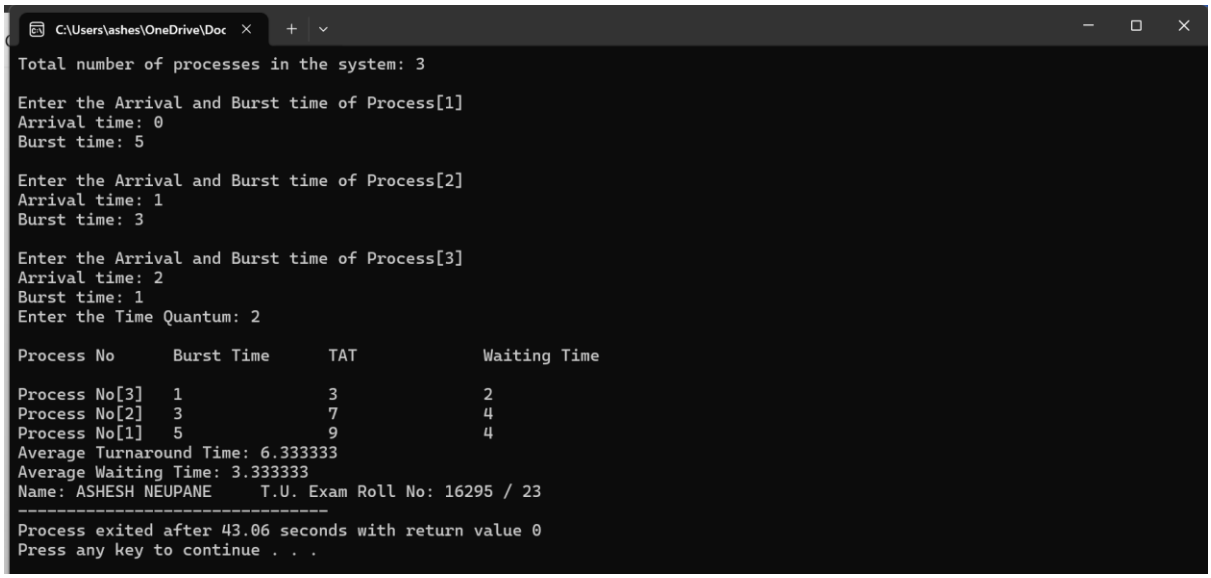
Write a C Program to implement the Round Robin (RR) CPU Scheduling Algorithm with arrival time.

Source Code:

```
#include <stdio.h>
int main() {
   int i, processes, sum = 0, cnt = 0, y, q, wt = 0, tat = 0, at[10], bt[10], temp[10];
   float avg_waitt, avg_turnat;
   // Input the number of processes
   printf("Total number of processes in the system: ");
   scanf("%d", &processes);
   y = processes;  // Assign number of processes to y
   // Input arrival time and burst time for each process
   for(i = 0; i < processes; i++) {
      printf("\nEnter the Arrival and Burst time of Process[%d]\n", i + 1);
      printf("Arrival time: ");
      scanf("%d", &at[i]);
      printf("Burst time: ");
      scanf("%d", &bt[i]);
      temp[i] = bt[i];  // Initialize remaining burst time
   }
   // Input the time quantum
   printf("Enter the Time Quantum: ");
   scanf("%d", &q);
   // Display header for the process info
   printf("\nProcess No \tBurst Time \tTAT \t\tWaiting Time\n");
   // Scheduling loop
   for(sum = 0, i = 0; y != 0;) {
      if(temp[i] <= q && temp[i] > 0) {
         sum = sum + temp[i];
         temp[i] = 0;
         cnt = 1;
      } else if(temp[i] > 0) {
         temp[i] = temp[i] - q;
         sum = sum + q;
```

```
        }
        if(temp[i] == 0 && cnt == 1) {
            y--;  // Decrement remaining processes
            printf("\nProcess No[%d] \t%d \t\t%d \t\t%d", i + 1, bt[i], sum - at[i], sum - at[i] - bt[i]);
            wt = wt + sum - at[i] - bt[i];  // Calculate waiting time
            tat = tat + sum - at[i];  // Calculate turnaround time
            cnt = 0;
        }
        if(i == processes - 1) {
            i = 0;
        } else if(at[i + 1] <= sum) {
            i++;
        } else {
            i = 0;
        }
    }
    // Calculate average waiting time and turnaround time
    avg_waitt = wt * 1.0 / processes;
    avg_turnat = tat * 1.0 / processes;
    printf("\nAverage Turnaround Time: %f", avg_turnat);
    printf("\nAverage Waiting Time: %f", avg_waitt);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
}
```

Output Screen:



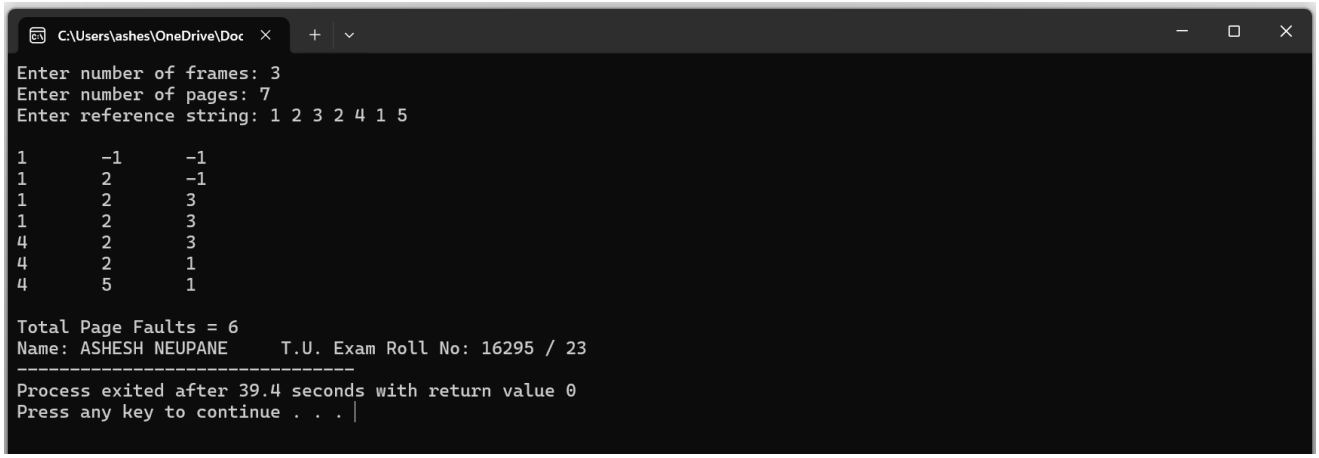
Write a C Program to implement the Least Recently Used (LRU) page replacement algorithm.

Source Code:

```
#include<stdio.h>
int findLRU(int time[], int n){
	int i, minimum = time[0], pos = 0;
	for(i = 1; i < n; ++i){
		if(time[i] < minimum){
			minimum = time[i];
			pos = i;
		}
	}
	return pos;
}
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;
	printf("Enter number of frames: ");
	scanf("%d", &no_of_frames);
	printf("Enter number of pages: ");
	scanf("%d", &no_of_pages);
	printf("Enter reference string: ");
   for(i = 0; i < no_of_pages; ++i){
	scanf("%d", &pages[i]);
   }
	for(i = 0; i < no_of_frames; ++i){
	frames[i] = -1;
   }
   for(i = 0; i < no_of_pages; ++i){
	flag1 = flag2 = 0;
	for(j = 0; j < no_of_frames; ++j){
		if(frames[j] == pages[i]){
			counter++;
			time[j] = counter;
```

```
                                flag1 = flag2 = 1;
                                break;
                        }
        }
        if(flag1 == 0){
                        for(j = 0; j < no_of_frames; ++j){
                        if(frames[j] == -1){
                                counter++;
                                faults++;
                                frames[j] = pages[i];
                                time[j] = counter;
                                flag2 = 1;
                                break;
                        }
                }
        }
        if(flag2 == 0){
                pos = findLRU(time, no_of_frames);
                counter++;
                faults++;
                frames[pos] = pages[i];
                time[pos] = counter;
        }
        printf("\n");
        for(j = 0; j < no_of_frames; ++j){
                printf("%d\t", frames[j]);
        }
        }
        printf("\n\nTotal Page Faults = %d", faults);
        printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
   return 0;
}
```

Output Screen:



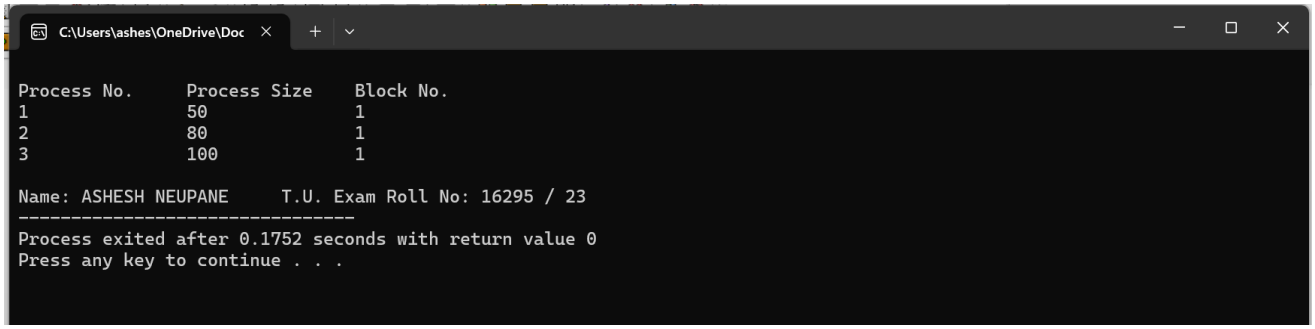
Write a program in C to implement the Worst Fit memory allocation strategy.

Source Code:

```
#include <stdio.h>
void implementWorstFit(int blockSize[],int blocks,int processSize[],int processes)
{
   int allocation[processes];
   for(int i=0;i<processes;i++)allocation[i]=-1;
   for(int i=0;i<processes;i++){
      int indexPlaced=-1;
      for(int j=0;j<blocks;j++){
         if(blockSize[j]>=processSize[i]){
            if(indexPlaced==-1)indexPlaced=j;
            else if(blockSize[indexPlaced]<blockSize[j])indexPlaced=j;
         }
      }
      if(indexPlaced!=-1){
         allocation[i]=indexPlaced;
         blockSize[indexPlaced]-=processSize[i];
      }
   }
   printf("\nProcess No.\tProcess Size\tBlock No.\n");
   for(int i=0;i<processes;i++){
      printf("%d\t\t%d\t\t",i+1,processSize[i]);
      if(allocation[i]!=-1)printf("%d\n",allocation[i]+1);
      else printf("Not Allocated\n");
   }
}
int main()
{
   int blockSize[]={500,100,60};
   int processSize[]={50,80,100};
   int blocks=sizeof(blockSize)/sizeof(blockSize[0]);
   int processes=sizeof(processSize)/sizeof(processSize[0]);
   implementWorstFit(blockSize,blocks,processSize,processes);
```

```
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0;
}
```

Output Screen:



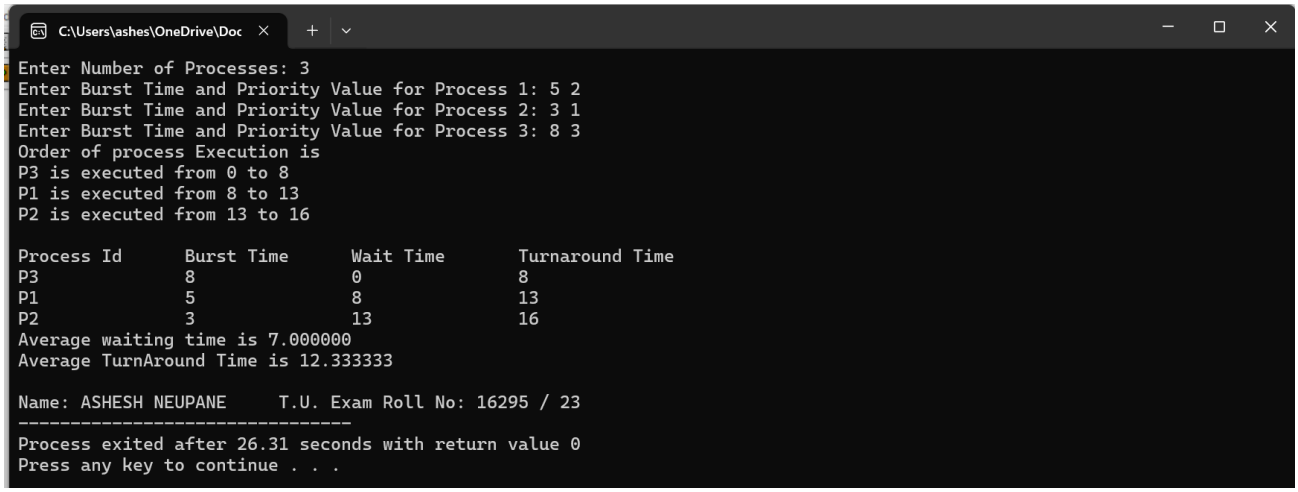
Write a program in C to implement Non-Preemptive Priority Scheduling.

Source Code:

```
#include <stdio.h>
void swap(int *a,int *b)
{
   int temp=*a;
   *a=*b;
   *b=temp;
}
int main()
{
   int n;
   printf("Enter Number of Processes: ");
   scanf("%d",&n);
   int burst[n],priority[n],index[n];
   for(int i=0;i<n;i++)
   {
      printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
      scanf("%d %d",&burst[i],&priority[i]);
      index[i]=i+1;
   }
   for(int i=0;i<n;i++)
   {
      int temp=priority[i],m=i;
      for(int j=i;j<n;j++)
      {
         if(priority[j]>temp)
         {
            temp=priority[j];
            m=j;
         }
      }
      swap(&priority[i],&priority[m]);
      swap(&burst[i],&burst[m]);
```

```
            swap(&index[i],&index[m]);
        }
        int t=0;
        printf("Order of process Execution is\n");
        for(int i=0;i<n;i++)
        {
            printf("P%d is executed from %d to %d\n",index[i],t,t+burst[i]);
            t+=burst[i];
        }
        printf("\n");
        printf("Process Id\tBurst Time\tWait Time\tTurnaround Time\n");
        int wait_time=0;
        int total_wait_time=0;
        int total_turnaround=0;
        for(int i=0;i<n;i++)
        {
            int turnaround=wait_time+burst[i];
            printf("P%d\t\t%d\t\t%d\t\t%d\n",index[i],burst[i],wait_time,turnaround);
            total_wait_time+=wait_time;
            total_turnaround+=turnaround;
            wait_time+=burst[i];
        }
        float avg_wait_time=(float)total_wait_time/n;
        float avg_turnaround=(float)total_turnaround/n;
        printf("Average waiting time is %f\n",avg_wait_time);
        printf("Average TurnAround Time is %f\n",avg_turnaround);
        printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
        return 0;
}
```

Output Screen:



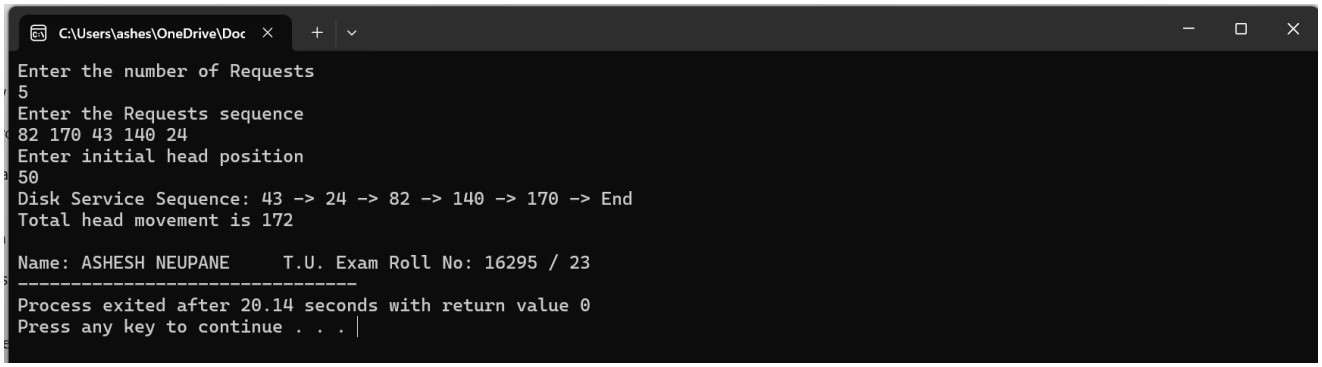
Write a program in C to implement the Shortest Seek Time First (SSTF) disk scheduling algorithm.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int RQ[100], visited[100], n, i, initial, TotalHeadMovement = 0, count = 0;
   printf("Enter the number of Requests\n");
   scanf("%d", &n);
   printf("Enter the Requests sequence\n");
   for(i = 0; i < n; i++) {
      scanf("%d", &RQ[i]);
      visited[i] = 0; // initialize visited array
   }
   printf("Enter initial head position\n");
   scanf("%d", &initial);
   printf("Disk Service Sequence: ");
   while(count != n) {
      int min = 1000000, index = -1;
      for(i = 0; i < n; i++) {
         if(!visited[i]) {
            int distance = abs(RQ[i] - initial);
            if(distance < min) {
               min = distance;
               index = i;
            }
         }
      }
      visited[index] = 1; // mark as visited
      TotalHeadMovement += min;
      printf("%d -> ", RQ[index]);
      initial = RQ[index];
      count++;
   }
```

```
    printf("End\n");
    printf("Total head movement is %d\n", TotalHeadMovement);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0;
}
```

Output Screen:



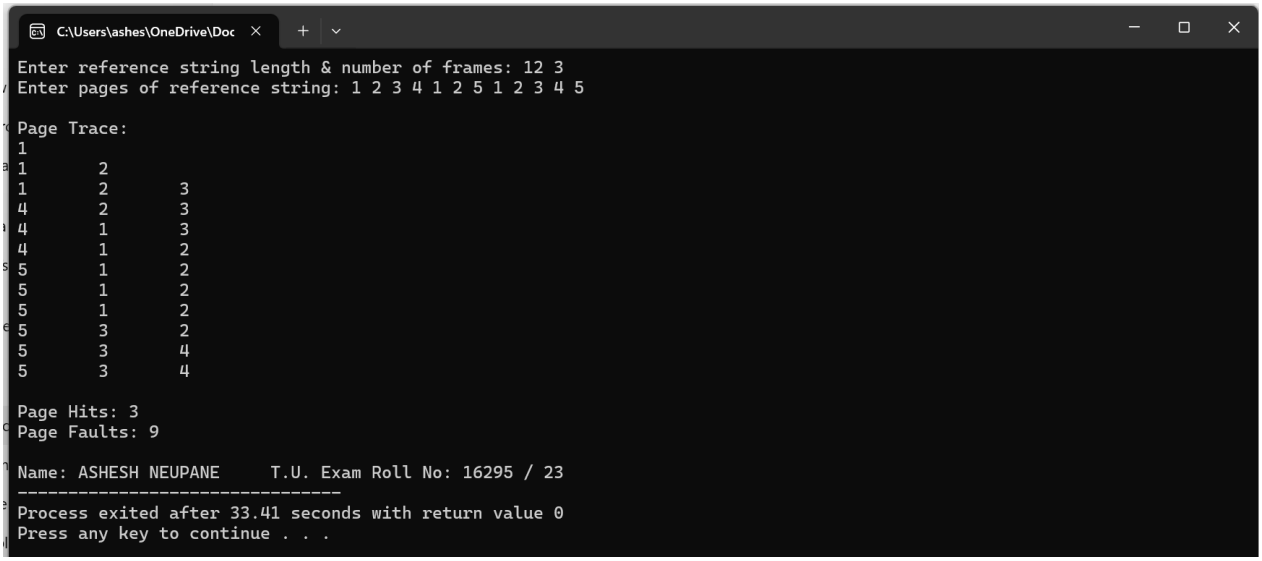
Write a program in C to implement the FIFO Page Replacement Algorithm.

Source Code:

```
#include <stdio.h>
void fifo(int ref[20], int reflen, int fnum) {
   int i, j, k, firstin = 0, found, phit = 0, pfault = 0;
   int frame[10];
   for(j = 0; j < fnum; j++)
      frame[j] = -1;
   printf("\nPage Trace:\n");
   for(i = 0; i < reflen; i++) {
      found = 0;
      for(j = 0; j < fnum; j++) {
         if(ref[i] == frame[j]) {
            found = 1;
            phit++;
            break;
         }
      }
      if(found == 0) {
         pfault++;
         frame[firstin] = ref[i];
         firstin = (firstin + 1) % fnum;
      }
      for(k = 0; k < fnum; k++) {
         if(frame[k] != -1)
            printf("%d\t", frame[k]);
      }
      printf("\n");
   }
   printf("\nPage Hits: %d\nPage Faults: %d\n", phit, pfault);
}
int main() {
   int i, reflen, fnum, refstr[20];
   printf("Enter reference string length & number of frames: ");
```

```
    scanf("%d %d", &reflen, &fnum);
    printf("Enter pages of reference string: ");
    for(i = 0; i < reflen; i++)
        scanf("%d", &refstr[i]);
    fifo(refstr, reflen, fnum);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0;
}
```

Output Screen:



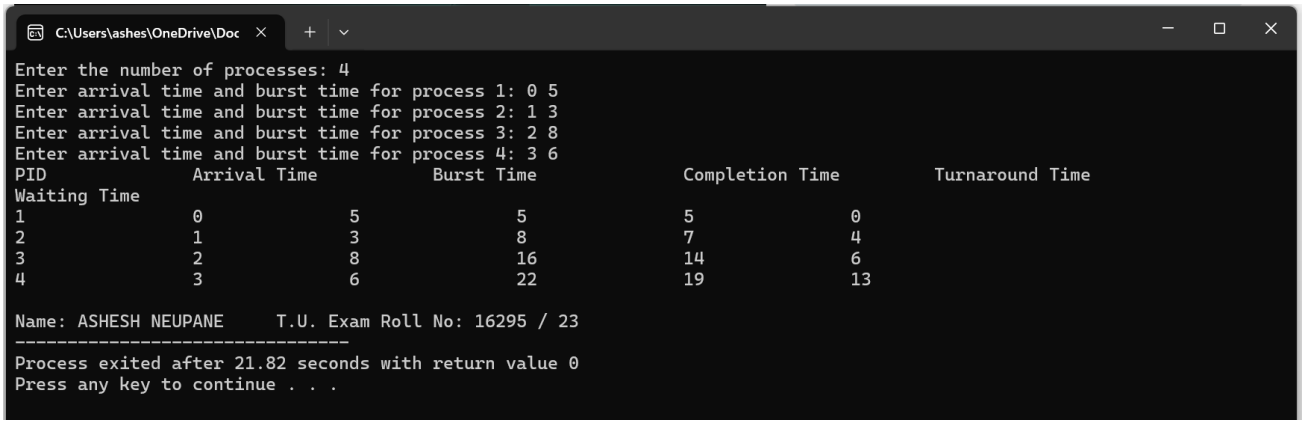
Write a program in C to implement the First-Come, First-Served (FCFS) Scheduling Algorithm.

Source Code:

```
#include <stdio.h>
struct Process {
    int pid;  // Process ID
    int arrival_time;  // Arrival time
    int burst_time;  // Burst time
    int completion_time;  // Completion time
    int waiting_time;  // Waiting time
    int turnaround_time;  // Turnaround time
};
void findCompletionTime(struct Process proc[], int n) {
    int current_time = 0;
    for (int i = 0; i < n; i++) {
        if (current_time < proc[i].arrival_time) {
            current_time = proc[i].arrival_time;
        }
        current_time += proc[i].burst_time;
        proc[i].completion_time = current_time;
    }
}
void findTurnaroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].turnaround_time = proc[i].completion_time - proc[i].arrival_time;
    }
}
void findWaitingTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].waiting_time = proc[i].turnaround_time - proc[i].burst_time;
    }
}
void printProcessTable(struct Process proc[], int n) {
    printf("PID\t\t Arrival Time \t\tBurst Time \t\tCompletion Time\t\tTurnaround Time \t\t Waiting Time \n");
```

```
    for (int i = 0; i < n; i++) {
        printf("%d\t\t %d\t\t%d\t\t%d\t\t%d\t\t%d\n",
                proc[i].pid, proc[i].arrival_time, proc[i].burst_time,
                proc[i].completion_time, proc[i].turnaround_time, proc[i].waiting_time);
    }
}
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process proc[n];
    for (int i = 0; i < n; i++) {
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        proc[i].pid = i + 1;
        scanf("%d%d", &proc[i].arrival_time, &proc[i].burst_time);
    }
    // Sort processes by arrival time
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (proc[j].arrival_time > proc[j + 1].arrival_time) {
                struct Process temp = proc[j];
                proc[j] = proc[j + 1];
                proc[j + 1] = temp;
            }
        }
    }
    findCompletionTime(proc, n);
    findTurnaroundTime(proc, n);
    findWaitingTime(proc, n);
    printProcessTable(proc, n);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0;
}
```

Output Screen:



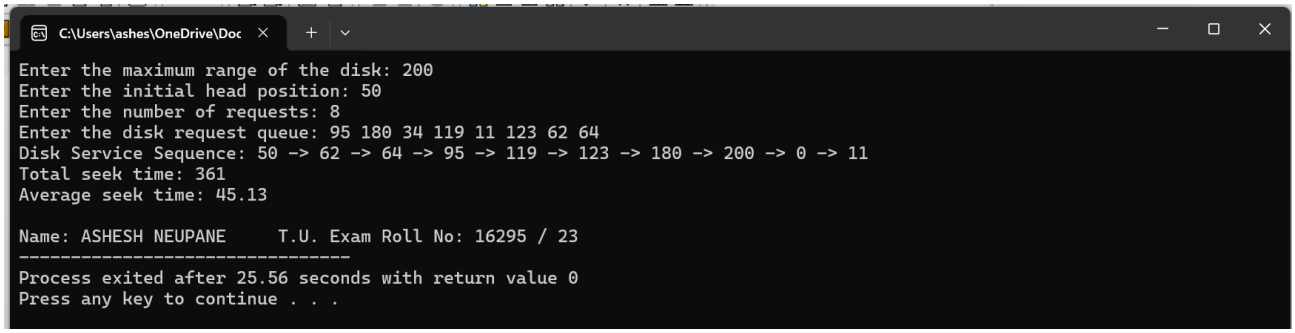
Write a program in C to implement the C-SCAN (Circular SCAN) Disk Scheduling Algorithm.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int n, head, max, seek = 0;
   printf("Enter the maximum range of the disk: ");
   scanf("%d", &max);
   printf("Enter the initial head position: ");
   scanf("%d", &head);
   printf("Enter the number of requests: ");
   scanf("%d", &n);
   int requests[n], queue1[n], queue2[n];
   int temp1 = 0, temp2 = 0;
   printf("Enter the disk request queue: ");
   for (int i = 0; i < n; i++) {
      scanf("%d", &requests[i]);
      if (requests[i] >= head)
         queue1[temp1++] = requests[i];
      else
         queue2[temp2++] = requests[i];
   }
   // Sort queue1 and queue2 in ascending order
   for (int i = 0; i < temp1 - 1; i++) {
      for (int j = i + 1; j < temp1; j++) {
         if (queue1[i] > queue1[j]) {
            int temp = queue1[i];
            queue1[i] = queue1[j];
            queue1[j] = temp;
         }
      }
   }
   for (int i = 0; i < temp2 - 1; i++) {
```

```
        for (int j = i + 1; j < temp2; j++) {
            if (queue2[i] > queue2[j]) {
                int temp = queue2[i];
                queue2[i] = queue2[j];
                queue2[j] = temp;
            }
        }
    }
    int total_requests = temp1 + temp2 + 2; // including max and 0
    int sequence[total_requests];
    int index = 0;
    sequence[index++] = head;
    // Add requests greater than or equal to head
    for (int i = 0; i < temp1; i++)
        sequence[index++] = queue1[i];
    sequence[index++] = max; // move to end
    sequence[index++] = 0;   // jump to start
    // Add remaining requests
    for (int i = 0; i < temp2; i++)
        sequence[index++] = queue2[i];
    printf("Disk Service Sequence: ");
    for (int i = 0; i < total_requests - 1; i++) {
        int diff = abs(sequence[i + 1] - sequence[i]);
        seek += diff;
        printf("%d -> ", sequence[i]);
    }
    printf("%d\n", sequence[total_requests - 1]);
    printf("Total seek time: %d\n", seek);
    printf("Average seek time: %.2f\n", (float)seek / n);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0;
}
```

Output Screen:



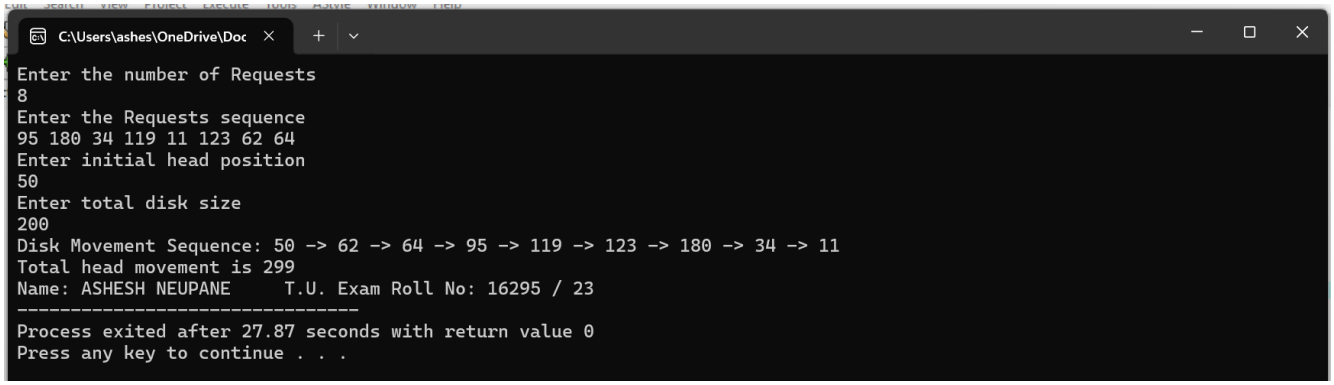
Write a program in C to implement the LOOK Disk Scheduling Algorithm.

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
int main() {
   int RQ[100], i, j, n, TotalHeadMoment = 0, initial, size;
   printf("Enter the number of Requests\n");
   scanf("%d", &n);
   printf("Enter the Requests sequence\n");
   for (i = 0; i < n; i++)
      scanf("%d", &RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d", &initial);
   printf("Enter total disk size\n");
   scanf("%d", &size);
   // Sort the request array
   for (i = 0; i < n; i++)
      for (j = 0; j < n - i - 1; j++)
         if (RQ[j] > RQ[j + 1]) {
            int temp = RQ[j];
            RQ[j] = RQ[j + 1];
            RQ[j + 1] = temp;
         }
   // Find the index of initial head position
   int index;
   for (i = 0; i < n; i++) {
      if (initial < RQ[i]) {
         index = i;
         break;
      }
   }
   printf("Disk Movement Sequence: ");
   // Calculate total head movement and show disk movement sequence
   for (i = index; i < n; i++) {
```

```
        printf("%d -> ", initial);
        TotalHeadMoment += abs(RQ[i] - initial);
        initial = RQ[i];
    }
    for (i = index - 1; i >= 0; i--) {
        printf("%d -> ", initial);
        TotalHeadMoment += abs(RQ[i] - initial);
        initial = RQ[i];
    }
    printf("%d\n", initial); // Print the last disk process
    printf("Total head movement is %d", TotalHeadMoment);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0;
}
```

Output Screen:



Write a program in C to implement the SCAN Disk Scheduling Algorithm.

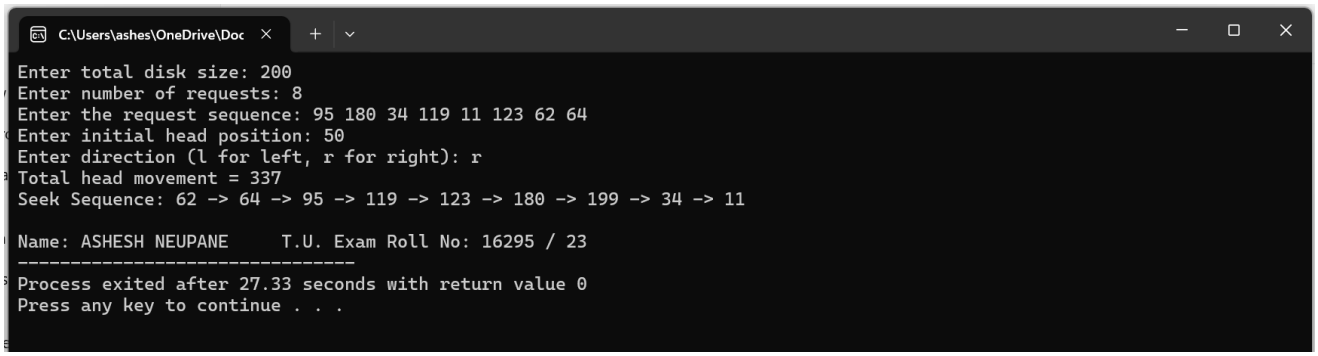
Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int comp(const void *a, const void *b) {
   return (*(int*)a - *(int*)b);
}
void SCAN(int arr[], int n, int head, int disk_size, char direction) {
   int left[100], right[100];
   int seek_sequence[200];
   int seek_count = 0, index = 0;
   int lsize = 0, rsize = 0;
   // add requests to left and right
   for (int i = 0; i < n; i++) {
      if (arr[i] < head)
         left[lsize++] = arr[i];
      else
         right[rsize++] = arr[i];
   }
   // add 0 or disk end depending on direction
   if (direction == 'l')
      left[lsize++] = 0;
   else if (direction == 'r')
      right[rsize++] = disk_size - 1;
   // sort both sides
   qsort(left, lsize, sizeof(int), comp);
   qsort(right, rsize, sizeof(int), comp);
   // move in SCAN direction
   int run = 2; // two runs: initial direction + reverse
   while (run--) {
      if (direction == 'l') {
         for (int i = lsize - 1; i >= 0; i--) {
            seek_sequence[index++] = left[i];
            seek_count += abs(head - left[i]);
```

```
            head = left[i];
        }
        direction = 'r';
    } else if (direction == 'r') {
        for (int i = 0; i < rsize; i++) {
            seek_sequence[index++] = right[i];
            seek_count += abs(head - right[i]);
            head = right[i];
        }
        direction = 'l';
    }
  }
  printf("Total head movement = %d\n", seek_count);
  printf("Seek Sequence: ");
  for (int i = 0; i < index; i++) {
    printf("%d", seek_sequence[i]);
    if (i != index - 1) printf(" -> ");
  }
  printf("\n");
}
int main() {
  int n, head, disk_size;
  char direction;
  printf("Enter total disk size: ");
  scanf("%d", &disk_size);
  printf("Enter number of requests: ");
  scanf("%d", &n);
  int arr[n];
  printf("Enter the request sequence: ");
  for (int i = 0; i < n; i++)
    scanf("%d", &arr[i]);
  printf("Enter initial head position: ");
  scanf("%d", &head);
  printf("Enter direction (l for left, r for right): ");
```

```
    scanf(" %c", &direction);
    SCAN(arr, n, head, disk_size, direction);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0;
}
```

Output Screen:



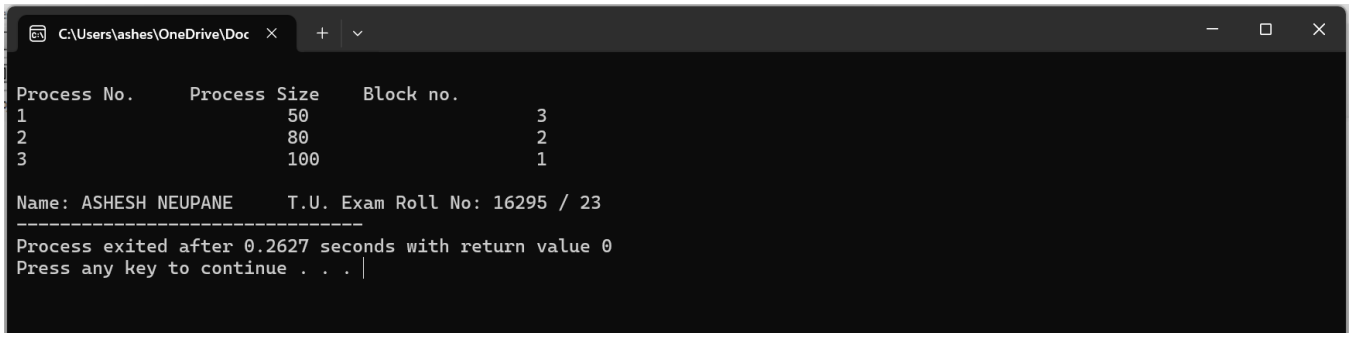
Write a program in C to implement the Best Fit Memory Allocation Algorithm.

Source Code:

```
#include <stdio.h>
void implimentBestFit(int blockSize[], int blocks, int processSize[], int proccesses)
{
   int allocation[proccesses];
   int occupied[blocks];
   for(int i = 0; i < proccesses; i++){
      allocation[i] = -1;
   }
   for(int i = 0; i < blocks; i++){
      occupied[i] = 0;
   }
   for (int i = 0; i < proccesses; i++)
   {
      int indexPlaced = -1;
      for (int j = 0; j < blocks; j++) {
         if (blockSize[j] >= processSize[i] && !occupied[j])
         {
            // place it at the first block fit to accomodate process
            if (indexPlaced == -1)
               indexPlaced = j;
            else if (blockSize[j] < blockSize[indexPlaced])
               indexPlaced = j;
         }
      }
      // If we were successfully able to find block for the process
      if (indexPlaced != -1)
      {
         allocation[i] = indexPlaced;  // allocate block j to process p[i]
         occupied[indexPlaced] = 1; // mark occupied
      }
   }
   printf("\nProcess No.\tProcess Size\tBlock no.\n");
```

```
    for (int i = 0; i < proccesses; i++)
    {
        printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n",allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}
int main()
{
     int blockSize[] = {500, 100, 60};
    int processSize[] = {50, 80, 100};
    int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
    int proccesses = sizeof(processSize)/sizeof(processSize[0]);
    implimentBestFit(blockSize, blocks, processSize, proccesses);
    printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
    return 0 ;
}
```

Output Screen:



Write a C program to implement the Shortest Remaining Time First (SRTF) Algorithm.

Source Code:

```
#include <stdio.h>
int main()
{
 int a[10],b[10],x[10],i,j,smallest,count=0,time,n;
 double avg=0,tt=0,end;
  printf("enter the number of Processes:\n");
  scanf("%d",&n);
 printf("enter arrival time\n");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 printf("enter burst time\n");
 for(i=0;i<n;i++)
 scanf("%d",&b[i]);
 for(i=0;i<n;i++)
 x[i]=b[i];

  b[9]=9999;

 for(time=0;count!=n;time++)
 {
  smallest=9;
  for(i=0;i<n;i++)
  {
  if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
  smallest=i;
  }
  b[smallest]--;
  if(b[smallest]==0)
  {
  count++;
  end=time+1;
  avg=avg+end-a[smallest]-x[smallest];
```

```
  tt= tt+end-a[smallest];
 }
}
printf("\n\nAverage waiting time = %lf\n",avg/n);
   printf("Average Turnaround time = %lf",tt/n);
   printf("\nName: ASHESH NEUPANE \t T.U. Exam Roll No: 16295 / 23");
   return 0;
}
```

Output Screen: