```
#include<stdio.h>
struct process {
      float pid,at,bt,tat,ct,wt,prio;
}pro[20];

void calculate(int n) {
      int i;
      float c=0;
      for(i=0;i<n;i++) {
            pro[i].ct=c+pro[i].bt;
            pro[i].tat=pro[i].ct-pro[i].at;
            pro[i].wt=pro[i].tat-pro[i].bt;
            c=pro[i].ct;
      }
}

void gantt(int n) {
      printf("PRIORITY SHEDULING: \n");
      printf("GANTT CHART: \n");
      printf("|");
      for(int i=0;i<n;i++) {
            printf("P%0.0f |",pro[i].pid);
      }
      printf("\n0");
      for(int i=0;i<n;i++) {
            printf("  %0.0f ",pro[i].ct);
      }
}

void display(int n) {
      float t=0,w=0;
      int i;
      printf("\n");
      printf("Process name\tArival time\t\tBurst time\t\tCompletion
time\t\tTurnaround time\t\tWaiting time\n");
      for (i=0;i<n;i++){


printf("P%0.0f\t\t\t%0.0f\t\t\t%0.0f\t\t\t%0.0f\t\t\t%0.0f\t\t\t%0.0f\n",pro[i].
pid,pro[i].at,pro[i].bt,pro[i].ct,pro[i].tat,pro[i].wt);
            t=t+pro[i].tat;
            w=w+pro[i].wt;
      }
      printf("Average Turnaround time:%0.2f\n",t/n);
      printf("Average Waiting time: %0.2f\n",w/n);
}

int main() {
      int n;
      printf("Enter the number of processes: ");
      scanf("%d",&n);
      int i,j;
      struct process temp;
      for(i=0;i<n;i++) {
            printf("Enter the burst time of process %d: ",i+1);
            scanf("%f",&pro[i].bt);
            printf("Enter the priority of process %d: ",i+1);
            scanf("%f",&pro[i].prio);
            pro[i].at=0;
            pro[i].pid=i+1;
      }
      for(i=0;i<n-1;i++) {
            for(j=0;j<n-i-1;j++) {
                  if(pro[j].prio>pro[j+1].prio || (pro[j].prio==pro[j+1].
```

```
pro[j].pid>pro[j+1].pid)) {
                        temp=pro[j];
                        pro[j]=pro[j+1];
                        pro[j+1]=temp;
                }
            }
        }
        calculate(n);
        gantt(n);
        display(n);
        return 0;
}
```

```
        float pid,at,bt,...
}pro[20];

void calculate(int n) {
        int i;
        float c=0;
        for(i=0;i<n;i++) {
                pro[i].ct=c+pro[i].bt;
                pro[i].tat=pro[i].ct-pro[i].at;
                pro[i].wt=pro[i].tat-pro[i].bt;
                c=pro[i].ct;
        }
}

void gantt(int n) {
        printf("RR: \n");
        printf("GANTT CHART: \n");
        printf("|");
        for(int i=0;i<n;i++) {
                printf("P%0.0f |",pro[i].pid);
        }
        printf("\n0");
        for(int i=0;i<n;i++) {
                printf("  %0.0f ",pro[i].ct);
        }
}

void display(int n) {
        float t=0,w=0;
        int i;
        printf("\n");
        printf("Process name\tArival time\t\tBurst time\t\tCompletion
time\t\tTurnaround time\t\tWaiting time\n");
        for (i=0;i<n;i++){


printf("P%0.0f\t\t\t%0.0f\t\t\t\t%0.0f\t\t\t\t%0.0f\t\t\t\t%0.0f\t\t\t\t%0.0f\n",pro[i].
pid,pro[i].at,pro[i].bt,pro[i].ct,pro[i].tat,pro[i].wt);
                t=t+pro[i].tat;
                w=w+pro[i].wt;
        }
        printf("Average Turnaround time:%0.2f\n",t/n);
        printf("Average Waiting time: %0.2f\n",w/n);
}

int main() {
        int n,ts;
        printf("Enter the number of processes: ");
        scanf("%d",&n);
        int i,j;
        struct process temp;
        for(i=0;i<n;i++) {
                printf("Enter the burst time of process %d: ",i+1);
                scanf("%f",&pro[i].bt);
                pro[i].pid=i+1;
                pro[i].rem_bt=pro[i].bt;
        }
        printf("Enter the time slice: ");
        scanf("%d",&ts);
        int all_done=0;
        float ptm=0;
        while (!all_done) {
```

```
            all_done=1;
            for(i=0;i<n;i++) {
                  if (pro[i].rem_bt>0) {
                        all_done=0;
                        if (pro[i].rem_bt>ts) {
                              ptm+=ts;
                              pro[i].rem_bt-=ts;
                        }else {
                              ptm+=pro[i].rem_bt;
                              pro[i].ct=ptm;
                              pro[i].rem_bt=0;
                        }
                  }
            }
      }
      for(i=0;i<n;i++) {
            pro[i].tat=pro[i].ct-pro[i].at;
            pro[i].wt=pro[i].tat-pro[i].bt;
      }
      gantt(n);
      display(n);
      return 0;
}
```

```c
#include<stdio.h>
#define MAX 20
struct process{
    int pid, at, bt, rt, priority, ct, tat, wt;
} pro[MAX];

void preemptivePriority(struct process p[], int n) {
    int completed = 0, time = 0, minIndex;
    while (completed < n) {
        minIndex = -1;
        int highestPriority = 99999;
        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].rt > 0 && p[i].priority <
highestPriority) {
                highestPriority = p[i].priority;
                minIndex = i;
            }
        }
        if (minIndex == -1) {
            time++;
            continue;
        }
        p[minIndex].rt--;
        time++;
        if (p[minIndex].rt == 0) {
            p[minIndex].ct = time;
            p[minIndex].tat = p[minIndex].ct - p[minIndex].at;
            p[minIndex].wt = p[minIndex].tat - p[minIndex].bt;
            completed++;
        }
    }
    printf("\nPID\tAT\tBT\tPR\tCT\tTAT\tWT\n");
    float totTat = 0, totWt = 0;
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].at,
p[i].bt, p[i].priority, p[i].ct, p[i].tat, p[i].wt);
        totTat += p[i].tat;
        totWt += p[i].wt;
    }
    printf("\nAverage turnaround time: %.2f", (totTat) / n);
    printf("\nAverage waiting time: %.2f\n", (totWt) / n);
}

void gantt(int n) {
    printf("\nGantt Chart\n");
    printf("| ");
    for (int i = 0; i < n; i++)
```

```
        printf("P%d |", pro[i].pid);
    printf("\n0  ");
    for (int i = 0; i < n; i++)
        printf("  %d ", pro[i].ct);
    printf("\n");
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter Arrival time, Burst time and Priority for P%d: ", i
+ 1);
        scanf("%d%d%d", &pro[i].at, &pro[i].bt, &pro[i].priority);
        pro[i].pid = i + 1;
        pro[i].rt = pro[i].bt;
    }
    preemptivePriority(pro, n);
    gantt(n);
    return 0;
}
```

```c
#include <stdio.h>

struct process {
    int pid, at, bt, ct, wt, tat, rt;
} pro[20];

void calculate(int n) {
    int completed = 0, time = 0, minIndex, minBt;
    int isCompleted[n];
    for (int i = 0; i < n; i++) {
        isCompleted[i] = 0;
        pro[i].rt = pro[i].bt;
    }


    int ganttOrder[100], ganttIndex = 0;

    while (completed != n) {
        minIndex = -1;
        minBt = 99999;
        for (int i = 0; i < n; i++) {
            if (pro[i].at <= time && isCompleted[i] == 0 && pro[i].rt
< minBt) {
                minBt = pro[i].rt;
                minIndex = i;
            }
        }
        if (minIndex == -1) {
            time++;
        } else {

            ganttOrder[ganttIndex++] = pro[minIndex].pid;
            pro[minIndex].rt--;
            time++;
            if (pro[minIndex].rt == 0) {
                pro[minIndex].ct = time;
                pro[minIndex].tat = pro[minIndex].ct -
pro[minIndex].at;
                pro[minIndex].wt = pro[minIndex].tat -
pro[minIndex].bt;
                isCompleted[minIndex] = 1;
                completed++;
            }
        }
    }
```

```
    printf("\nGANTT CHART:\n");
    printf("|");
    for (int i = 0; i < ganttIndex; i++) {
        printf(" P%d |", ganttOrder[i]);
    }
    printf("\n0");
    for (int i = 0; i < ganttIndex; i++) {
        printf("    %d", (i == 0 ? pro[ganttOrder[i] - 1].at :
pro[ganttOrder[i - 1] - 1].ct));
    }
    printf("    %d\n", time);
}

void display(int n) {
    float totalTat = 0, totalWt = 0;
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", pro[i].pid, pro[i].at,
pro[i].bt, pro[i].ct, pro[i].tat, pro[i].wt);
        totalTat += pro[i].tat;
        totalWt += pro[i].wt;
    }
    printf("\nAverage Turnaround Time: %.2f ms", totalTat / n);
    printf("\nAverage Waiting Time: %.2f ms\n", totalWt / n);
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter Arrival Time and Burst Time for Process %d: ", i
+ 1);
        scanf("%d%d", &pro[i].at, &pro[i].bt);
        pro[i].pid = i + 1;
    }
    calculate(n);
    display(n);
    return 0;
}
```

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#define SIZE 2
sem_t empty, full, mutex;
int in=0,out = 0;
int buffer[SIZE];
void *Pr_Item(void*);
void *Cr_Item(void*);

void *Pr_Item(void *arg) {
      while(1){
            int item;
            item=rand()%10;
            sem_wait(&empty);
            sem_wait(&mutex);
            buffer[in] = item,
            printf("\nProducer produced item: %d", item);
            sleep(1);
            in=(in+1)%SIZE;
            sem_post(&mutex);
            sem_post(&full);
      }
}

void *Cr_Item(void *arg){
      while(1){
            int item;
            sem_wait(&full);
            sem_wait(&mutex);
            item = buffer[out];
            printf("\nConsumer consumed item. %d", item);
            sleep(1);
            out=(out+1)%SIZE;
            sem_post(&mutex);
            sem_post(&empty);
      }
}

void main(){
      pthread_t P, C;
      sem_init(&empty, 0, SIZE);
      sem_init(&full, 0, 0);
      sem_init(&mutex, 0, 1);
      pthread_create(&P, NULL, (void*) Pr_Item, NULL);
```

```
        pthread_create(&C, NULL, (void*)Cr_Item, NULL);
        pthread_join(P, NULL);
}
```

```
#include<stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t chopstick[5];
void *philos(void *);
void eat(int);

void eat(int ph){
      printf("\nPhilosopher %d begins to eat", ph);
}
void *philos(void *n){
      int ph= *(int*)n;
      printf("\nphilosopher %d wants to eat", ph);
      printf("\nPhilosopher %d tries to pick up left chopstick", ph);
      sem_wait(&chopstick[ph]);
      printf("\nPhilosopher %d picks the left chopstick", ph);
      printf("\nPhilosopher %d tries to pick up right chopstick", ph);
      sem_wait(&chopstick[(ph+1)%5]);
      printf("\nPhilosopher %d picks the right chopstick", ph);
      eat(ph);
      sleep(2);
      printf("\nPhilosopher %d has finished eating", ph);
      sem_post(&chopstick[(ph+1)%5]);
      printf("\nPhilosopher %d leaves the right chopstick", ph);
      sem_post(&chopstick[ph]);
      printf("\nPhilosopher %d leaves the left chopstick", ph);
}

int main(){
      int i, n[5];
      pthread_t T[5];
      for(int i = 0 ; i<5 ;i ++){
            sem_init(&chopstick[i], 0, 1);
      }
      for(int i = 0;i<5 ;i++){
            n[i] = i;
            pthread_create(&T[i], NULL, philos, (void*)&n[i]);
      }
      for(int i=0;i<5;i++){
            pthread_join(T[i],NULL);
      }
}
```

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

#define MAX_PROCESSES 5
#define MAX_RESOURCES 3

Int allocation[MAX_PROCESSES][MAX_RESOURCES],
max[MAX_PROCESSES][MAX_RESOURCES], available[MAX_RESOURCES];

Void calculateNeed(int need[MAX_PROCESSES][MAX_RESOURCES], int
numProcesses) {
   For (int I = 0; I < numProcesses; i++)
      For (int j = 0; j < MAX_RESOURCES; j++)
         Need[i][j] = max[i][j] – allocation[i][j];
}

Bool isSafe(int need[MAX_PROCESSES][MAX_RESOURCES], int numProcesses) {
   Int work[MAX_RESOURCES], finish[MAX_PROCESSES] = {false},
safeSequence[MAX_PROCESSES], count = 0;

   For (int I = 0; I < MAX_RESOURCES; i++) work[i] = available[i];

   While (count < numProcesses) {
      Bool found = false;
      For (int p = 0; p < numProcesses; p++) {
         If (!finish[p]) {
            Bool canAllocate = true;
            For (int i = 0; i < MAX_RESOURCES; i++)
```

```
            If (need[p][j] > work[j]) { canAllocate = false; break; }
        If (canAllocate) {
            For (int j = 0; j < MAX_RESOURCES; j++) work[j] += allocation[p][j];
            safeSequence[count++] = p;
            finish[p] = true;
            found = true;
        }
      }
    }
    If (!found) {
        Printf("System is not in a safe state.\n");
        Return false;
    }
  }

  Printf("System is in a safe state.\nSafe sequence: ");
  For (int I = 0; I < numProcesses; i++) printf("%d ", safeSequence[i]);
  Printf("\n");
  Return true;
}

Bool requestResources(int processID, int request[], int
need[MAX_PROCESSES][MAX_RESOURCES]) {
   For (int j = 0; j < MAX_RESOURCES; j++)
      If (request[j] > need[processID][j] || request[j] > available[j]) {
         Printf("Request cannot be granted.\n");
         Return false;
```

```
        }

    For (int j = 0; j < MAX_RESOURCES; j++) {
        Available[j] -= request[j];
        Allocation[processID][j] += request[j];
        Need[processID][j] -= request[j];
    }

    Return isSafe(need, MAX_PROCESSES);
}

Int main() {
    Int numProcesses, numResources, need[MAX_PROCESSES][MAX_RESOURCES];

    Printf("Enter number of processes: ");
    Scanf("%d", &numProcesses);
    Printf("Enter number of resources: ");
    Scanf("%d", &numResources);

    Printf("Enter maximum resources matrix:\n");
    For (int I = 0; I < numProcesses; i++)
        For (int j = 0; j < numResources; j++)
            Scanf("%d", &max[i][j]);

    Printf("Enter allocation matrix:\n");
    For (int I = 0; I < numProcesses; i++)
        For (int j = 0; j < numResources; j++)
```

```
            Scanf("%d", &allocation[i][j]);

        Printf("Enter available resources:\n");
        For (int j = 0; j < numResources; j++)
            Scanf("%d", &available[j]);

        calculateNeed(need, numProcesses);
        isSafe(need, numProcesses);

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

        requestResources(0, request, need);

        return 0;
    }
```

OUTPUT:-

Enter the no of process : 5
Enter the no of resources : 3
Enter the allocation matrix
0 1 0
2 0 0

```
#include <stdio.h>

int findLRU(int time[],int n) {
    int min=time[0],pos=0;
    for (int i=1;i<n;i++) {
        if (time[i]<min) {
            min=time[i];
            pos=i;
        }
    }
    return pos;
}

void lruPageReplacement(int pages[],int
n,int capacity) {
    int
frame[capacity],time[capacity],pageFaults
=0,counter=0;

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

    for (int i=0;i<n;i++) {
        int found=0;

        for (int j=0;j<capacity;j++) {
            if (frame[j]==pages[i]) {
                found=1;
                time[j]=counter++;
                break;
            }
        }

        if (!found) {
            int replacePos;
            if (i<capacity) {
                replacePos=i;
            } else {

replacePos=findLRU(time,capacity);
            }

            frame[replacePos]=pages[i];
            time[replacePos]=counter++;
            pageFaults++;
        }

        printf("Step %d: ",i + 1);
        for (int j=0;j<capacity;j++) {
            if (frame[j]!=-1)
                printf("%d ",frame[j]);
            else
                printf("- ");
        }
        printf("\n");
    }

    printf("\nTotal Page Faults:
%d\n",pageFaults);
}

int main() {
    int n,capacity;

    printf("Enter the number of pages: ");
    scanf("%d",&n);

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

    printf("Enter the frame capacity: ");
    scanf("%d",&capacity);

    lruPageReplacement(pages,n,capacity);

    return 0;
}
```

#include <stdio.h>

```
int findOptimal(int pages[],int frame[],int
n,int capacity,int index) {
   int farthest=-1, pos=-1;

   for (int i=0;i<capacity;i++) {
      int j;
      for (j=index+1;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 optimalPageReplacement(int
pages[],int n,int capacity) {
   int frame[capacity],pageFaults=0;

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

   for (int i=0;i<n;i++) {
      int found=0;

      for (int j=0;j<capacity;j++) {
         if (frame[j]==pages[i]) {
            found=1;
            break;
         }
      }

      if (!found) {
         int replacePos;
         if (i<capacity) {
            replacePos=i;
         } else {
            replacePos=findOptimal(pages,frame,n,cap
acity,i);
         }

         frame[replacePos]=pages[i];
         pageFaults++;
      }


      printf("Step %d: ",i+1);
      for (int j=0;j<capacity;j++) {
         if (frame[j]!=-1)
            printf("%d ",frame[j]);
         else
            printf("- ");
      }
      printf("\n");
   }

   printf("\nTotal Page Faults:
%d\n",pageFaults);
}

int main() {
   int n, capacity;


   printf("Enter the number of pages: ");
   scanf("%d",&n);

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

   printf("Enter the frame capacity: ");
   scanf("%d",&capacity);


optimalPageReplacement(pages,n,capacity
);

   return 0;
}
```

```c
#include <stdio.h>

void fifoPageReplacement(int pages[], int n,
int capacity) {
   int frame[capacity], front = 0, pageFaults
= 0;

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

   for (int i = 0; i < n; i++) {
      int found = 0;
      for (int j = 0; j < capacity; j++) {
         if (frame[j] == pages[i]) {
            found = 1;
         }
      }

      if (!found) {
         frame[front] = pages[i];
         front = (front + 1) % capacity;
         pageFaults++;
      }


      printf("Step %d: ", i + 1);
      for (int j = 0; j < capacity; j++) {
         if (frame[j] != -1)
            printf("%d ", frame[j]);
         else
            printf("- ");
      }
      printf("\n");
   }

   printf("\nTotal Page Faults: %d\n",
pageFaults);
}

int main() {
   int n, capacity;


   printf("Enter the number of pages: ");
   scanf("%d", &n);

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

   printf("Enter the frame capacity: ");
   scanf("%d", &capacity);


   fifoPageReplacement(pages, n, capacity);

   return 0;
}
```

OUTPUT

Enter the number of frames:

3

Enter the number of pages:

10

Enter the page reference string:

7 8 4 5 7 9 0 6 7 4

Page	Frame
7	7--
8	78-
4	784
5	584
7	574
9	579
0	079
6	069
7	067
4	467

Page Fault: 10

OUTPUT

Enter the number of frames:

3

Enter the number of pages:

20

Enter the page reference string:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

0	201
3	023
2	023
0	012
1	012

Page Fault: 15

OUTPUT

Enter the number of frames:

3

Enter the number of pages:

20

Enter the page reference string:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

0	201
3	023
2	023
0	012
1	012

Page Fault: 15

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

void fcfs(int requests[], int n, int head) {
        int seek_time = 0;
        printf("\nFCFS Disk
Scheduling\nSequence: %d -> ", head);

        for (int i = 0; i < n; i++) {
        seek_time += abs(requests[i] -
head);
        head = requests[i];
        printf("%d", head);
        if (i < n - 1) printf(" -> ");
        }
        printf("\nTotal Seek Time: %d\n",
seek_time);
}

void sstf(int requests[], int n, int head) {
        int completed[n], seek_time = 0,
min_index;
        for (int i = 0; i < n; i++) completed[i] =
0;

        printf("\nSSTF Disk
Scheduling\nSequence: %d -> ", head);
        for (int i = 0; i < n; i++) {
        int min_seek = 9999;
        for (int j = 0; j < n; j++) {
        if (!completed[j]) {
        int distance = abs(requests[j] -
head);
        if (distance < min_seek) {
                min_seek = distance;
                min_index = j;
        }
        }
        }
        seek_time += min_seek;
        head = requests[min_index];
        completed[min_index] = 1;
        printf("%d", head);
        if (i < n - 1) printf(" -> ");
        }
        printf("\nTotal Seek Time: %d\n",
seek_time);
}

void scan(int requests[], int n, int head, int
disk_size) {
        int seek_time = 0, i, j, sorted[n + 1];
        sorted[0] = head;
        for (i = 1; i <= n; i++) sorted[i] =
requests[i - 1];
        for (i = 0; i <= n; i++) {
        for (j = i + 1; j <= n; j++) {
        if (sorted[i] > sorted[j]) {
        int temp = sorted[i];
        sorted[i] = sorted[j];
        sorted[j] = temp;
        }
        }
        }

        printf("\nSCAN Disk
Scheduling\nSequence: ");
        for (i = 0; i <= n; i++) {
        seek_time += abs(sorted[i] - head);
        head = sorted[i];
        printf("%d", head);
        if (i < n) printf(" -> ");
        }
        printf("\nTotal Seek Time: %d\n",
seek_time);
}

void cscan(int requests[], int n, int head, int
disk_size) {
        int seek_time = 0, i, j, sorted[n + 2];
        sorted[0] = head;
        sorted[n + 1] = disk_size - 1;
        for (i = 1; i <= n; i++) sorted[i] =
requests[i - 1];
        for (i = 0; i <= n + 1; i++) {
        for (j = i + 1; j <= n + 1; j++) {
        if (sorted[i] > sorted[j]) {
        int temp = sorted[i];
        sorted[i] = sorted[j];
        sorted[j] = temp;
```

```
            }
            }
            }

            printf("\nC-SCAN Disk
  Scheduling\nSequence: ");
            for (i = 0; i <= n + 1; i++) {
            seek_time += abs(sorted[i] - head);
            head = sorted[i];
            printf("%d", head);
            if (i < n + 1) printf(" -> ");
            }
            printf("\nTotal Seek Time: %d\n",
 seek_time);
}

 void look(int requests[], int n, int head) {
            int seek_time = 0, i, j, sorted[n + 1];
            sorted[0] = head;
            for (i = 1; i <= n; i++) sorted[i] =
  requests[i - 1];
            for (i = 0; i <= n; i++) {
            for (j = i + 1; j <= n; j++) {
            if (sorted[i] > sorted[j]) {
            int temp = sorted[i];
            sorted[i] = sorted[j];
            sorted[j] = temp;
            }
            }
            }

           printf("\nLOOK Disk
  Scheduling\nSequence: ");
           for (i = 0; i <= n; i++) {
           seek_time += abs(sorted[i] - head);
           head = sorted[i];
           printf("%d", head);
           if (i < n) printf(" -> ");
          }
          printf("\nTotal Seek Time: %d\n",
seek_time);
}

int main() {
          int n, choice, head, disk_size;

                printf("Enter number of requests: ");
                scanf("%d", &n);

                int requests[n];
                printf("Enter request sequence:\n");
                for (int i = 0; i < n; i++) {
                scanf("%d", &requests[i]);
                }

                printf("Enter initial head position: ");
                scanf("%d", &head);
                printf("Enter disk size: ");
                scanf("%d", &disk_size);

                while (1) {
                printf("\nDisk Scheduling
 Algorithms:\n");
                printf("1. FCFS\n2. SSTF\n3.
 SCAN\n4. C-SCAN\n5. LOOK\n6. Exit\n");
                printf("Enter choice: ");
                scanf("%d", &choice);

                switch (choice) {
                case 1: fcfs(requests, n, head); break;
                case 2: sstf(requests, n, head); break;
                case 3: scan(requests, n, head,
disk_size); break;
                case 4: cscan(requests, n, head,
disk_size); break;
                case 5: look(requests, n, head);
break;
                case 6: return 0;
                default: printf("Invalid choice! Try
again.\n");
                }
                }
}
```

OUTPUT

```
Enter number of requests: 6
Enter request sequence:
65
98
14
122
187
67
Enter initial head position: 53
Enter disk size: 200

Disk Scheduling Algorithms:
1. FCFS
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK
6. Exit
Enter choice: 1

FCFS Disk Scheduling
Sequence: 53 -> 65 -> 98 -> 14 -> 122 ->
187 -> 67
Total Seek Time: 422

Disk Scheduling Algorithms:
1. FCFS
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK
6. Exit
Enter choice: 2

SSTF Disk Scheduling
Sequence: 53 -> 65 -> 67 -> 98 -> 122 ->
187 -> 14
Total Seek Time: 307

Disk Scheduling Algorithms:
1. FCFS
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK
6. Exit
Enter choice: 3

SCAN Disk Scheduling
Sequence: 14 -> 53 -> 65 -> 67 -> 98 ->
122 -> 187
Total Seek Time: 212

Disk Scheduling Algorithms:
1. FCFS
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK
6. Exit
Enter choice: 4

C-SCAN Disk Scheduling
Sequence: 14 -> 53 -> 65 -> 67 -> 98 ->
122 -> 187 -> 199
Total Seek Time: 224

Disk Scheduling Algorithms:
1. FCFS
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK
6. Exit
Enter choice: 5

LOOK Disk Scheduling
Sequence: 14 -> 53 -> 65 -> 67 -> 98 ->
122 -> 187
Total Seek Time: 212
```