Q.1

a) FCFS

```
#include<stdio.h>

int main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20):");
   scanf("%d",&n);

   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);
   }

   wt[0]=0;    //waiting time for first process is 0

   //calculating waiting time
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
   }

   printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

   //calculating turnaround time
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      avwt+=wt[i];
      avtat+=tat[i];
      printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
   }

   avwt/=i;
   avtat/=i;
   printf("\n\nAverage Waiting Time:%d",avwt);
   printf("\nAverage Turnaround Time:%d",avtat);

   return 0;
}
```

b) SCAN

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

void scan_algorithm(int left[], int right[], int count, int limit)
{
    int arr[20];
    int x = count - 1, y = count + 1, c = 0, d = 0, j;
    while(x > -1)
    {
        printf("\nX:\t%d", x);
        printf("\nLeft[X]:\t%d", left[x]);
        arr[d] = left[x];
        x--;
        d++;
    }
    arr[d] = 0;
    while(y < limit + 1)
    {
        arr[y] = right[c];
        c++;
        y++;
    }
    printf("\nScanning Order:\n");
    for(j = 0; j < limit + 1; j++)
    {
        printf("\n%d", arr[j]);
    }
}

void division(int elements[], int limit, int disk_head)
{
    int count = 0, p, q, m, x;
    int left[20], right[20];
    for(count = 0; count < limit; count++)
    {
        if(elements[count] > disk_head)
        {
            printf("\nBreak Position:\t%d\n", elements[count]);
            break;
        }
    }
    printf("\nValue:\t%d\n", count);
    q = 1;
    p = 0;
    m = limit;
    left[0] = elements[0];
    printf("\nLeft:\t%d", left[0]);
    while(q < count)
```

```
    {
        printf("\nElement[l] value:\t%d" , elements[q]);
        left[q] = elements[q];
        printf("\nLeft:\t%d", left[q]);
        q++;
        printf("\nl:\t%d", q);
    }
    x = count;
    while(x < m)
    {
        right[p] = elements[x];
        printf("\nRight:\t%d", right[p]);
        printf("\nElement:\t%d", elements[x]);
        p++;
        x++;
    }
    scan_algorithm(left, right, count, limit);
}

void sorting(int elements[], int limit)
{
    int location, count, j, temp, small;
    for(count = 0; count < limit - 1; count++)
    {
        small = elements[count];
        location = count;
        for(j = count + 1; j < limit; j++)
        {
            if(small > elements[j])
            {
                small = elements[j];
                location = j;
            }
        }
        temp = elements[location];
        elements[location] = elements[count];
        elements[count] = temp;
    }
}

int main()
{
    int count, disk_head, elements[20], limit;
    printf("Enter total number of locations:\t");
    scanf("%d", &limit);
    printf("\nEnter position of disk head:\t");
    scanf("%d", &disk_head);
    printf("\nEnter elements of disk head queue\n");
    for(count = 0; count < limit; count++)
    {
```

```
        printf("Element[%d]:\t", count + 1);
        scanf("%d", &elements[count]);
    }
    sorting(elements, limit);
    division(elements, limit, disk_head);
    getch();
    return 0;
}
```

c) C-SCAN

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

int main()
{
    int queue1[30], queue2[30], queue3[30];
    int limit, disk_head, count = 0, j, seek_time = 0, range, diff;
    int t1, t2 = 0, t3 = 0;
    float avg_seek_time;
    printf("Maximum Range of Disk:\t");
    scanf("%d", &range);
    printf("Initial Head Position:\t");
    scanf("%d", &disk_head);
    printf("Queue Request Size:\t");
    scanf("%d", &limit);
    printf("Disk Queue Element Positions:\n");
    while(count < limit)
    {
        scanf("%d", &t1);
        if(t1 >= disk_head)
        {
            queue1[t2] = t1;
            t2++;
        }
        else
        {
            queue2[t3] = t1;
            t3++;
        }
        count++;
    }
    count = 0;
    while(count < t2 - 1)
    {
        j = count + 1;
        while(j < t2)
        {
            if(queue1[count] > queue1[j])
```

```
        {
            t1 = queue1[count];
            queue1[count] = queue1[j];
            queue1[j] = t1;
        }
        j++;
    }
    count++;
}
count = 0;
while(count < t3 - 1)
{
    j = count + 1;
    while(j < t3)
    {
        if(queue2[count] > queue2[j])
        {
            t1 = queue2[count];
            queue2[count] = queue2[j];
            queue2[j] = t1;
        }
        j++;
    }
    count++;
}
count = 1;
j = 0;
while(j < t2)
{
    queue3[count] = queue1[j];
    queue3[count] = range;
    queue3[count + 1] = 0;
    count++;
    j++;
}
count = t2 + 3;
j = 0;
while(j < t3)
{
    queue3[count] = queue2[j];
    queue3[0] = disk_head;
    count++;
    j++;
}
for(j = 0; j <= limit + 1; j++)
{
    diff = abs(queue3[j + 1] - queue3[j]);
    seek_time = seek_time + diff;
    printf("\nDisk Head:\t%d -> %d [Seek Time: %d]\n", queue3[j], queue3[j + 1], diff);
}
```

```
    printf("\nTotal Seek Time:\t%d\n", seek_time);
    avg_seek_time = seek_time / (float)limit;
    printf("\nAverage Seek Time:\t%f\n", avg_seek_time);
    return 0;
}
```

Q.2

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

int mutex=1,full=0,empty=3,x=0;

int main()
{
   int n;
   void producer();
   void consumer();
   int wait(int);
   int signal(int);
   printf("\n1.Producer\n2.Consumer\n3.Exit");
   while(1)
   {
      printf("\nEnter your choice:");
      scanf("%d",&n);
      switch(n)
      {
         case 1:    if((mutex==1)&&(empty!=0))
                       producer();
                    else
                       printf("Buffer is full!!");
                    break;
         case 2:    if((mutex==1)&&(full!=0))
                       consumer();
                    else
                       printf("Buffer is empty!!");
                    break;
         case 3:
                    exit(0);
                    break;
      }
   }

   return 0;
}

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

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

void producer()
{
   mutex=wait(mutex);
   full=signal(full);
   empty=wait(empty);
   x++;
   printf("\nProducer produces the item %d",x);
   mutex=signal(mutex);
}

void consumer()
{
   mutex=wait(mutex);
   full=wait(full);
   empty=signal(empty);
   printf("\nConsumer consumes item %d",x);
   x--;
   mutex=signal(mutex);
}
```