**Practical 4**

1. Queue Demonstration with Operations

Code:

#include<stdio.h>

int queue[100], front, rear, c, n, x;

void enqueue(int);

void dequeue(void);

void display(void);

int main()

{

rear = -1;

front = 0;

printf("\n Enter the size of Queue max is 100 :");

scanf("%d",&n);

printf("\n\t Queue operations using array");

printf("\n\t 1. Enqueue.\n\t 2. Dequeue.\n\t 3. Display.\n\t 4. Exit");

do

{

printf("\n Enter your choice :\n");

scanf("%d",&c);

switch(c)

{

case 1:

{

printf("\n Enter an element to enqueue\n");

scanf("%d",&x);

enqueue(x);

break;

}

case 2:

{

printf("\n Dequeuing the element\n");

dequeue();

break;

}

case 3:

{

printf("\n Queue Array\n");

display();

break;

}

case 4:

{

printf("\n Exit ...\n");

break;

}

default:

{

printf("\n Invalid Option\n");

break;

}

}

}while(c != 4);

return 0;

}

void enqueue(int p)

{

if(rear >= n-1)

{

printf(" Queue overflow.\n");

}

else

{

rear++;

queue[rear] = p;

}

}

void dequeue()

{

if(rear>=front)

{

int ele = queue[front];

queue[front] = '\0';

printf(" Deleted Element - %d \n",ele);

front++;

}

else

{

printf(" Queue is underflow\n");

}

}

void display()

{

if(rear<0)

{

printf(" Queue is empty\n");

}

else

{ if(front <= rear)

{

for(int i=front; i<=rear; i++)

{

printf(" Queue Elements :- %d \n",queue[i]);

}

}

else

{

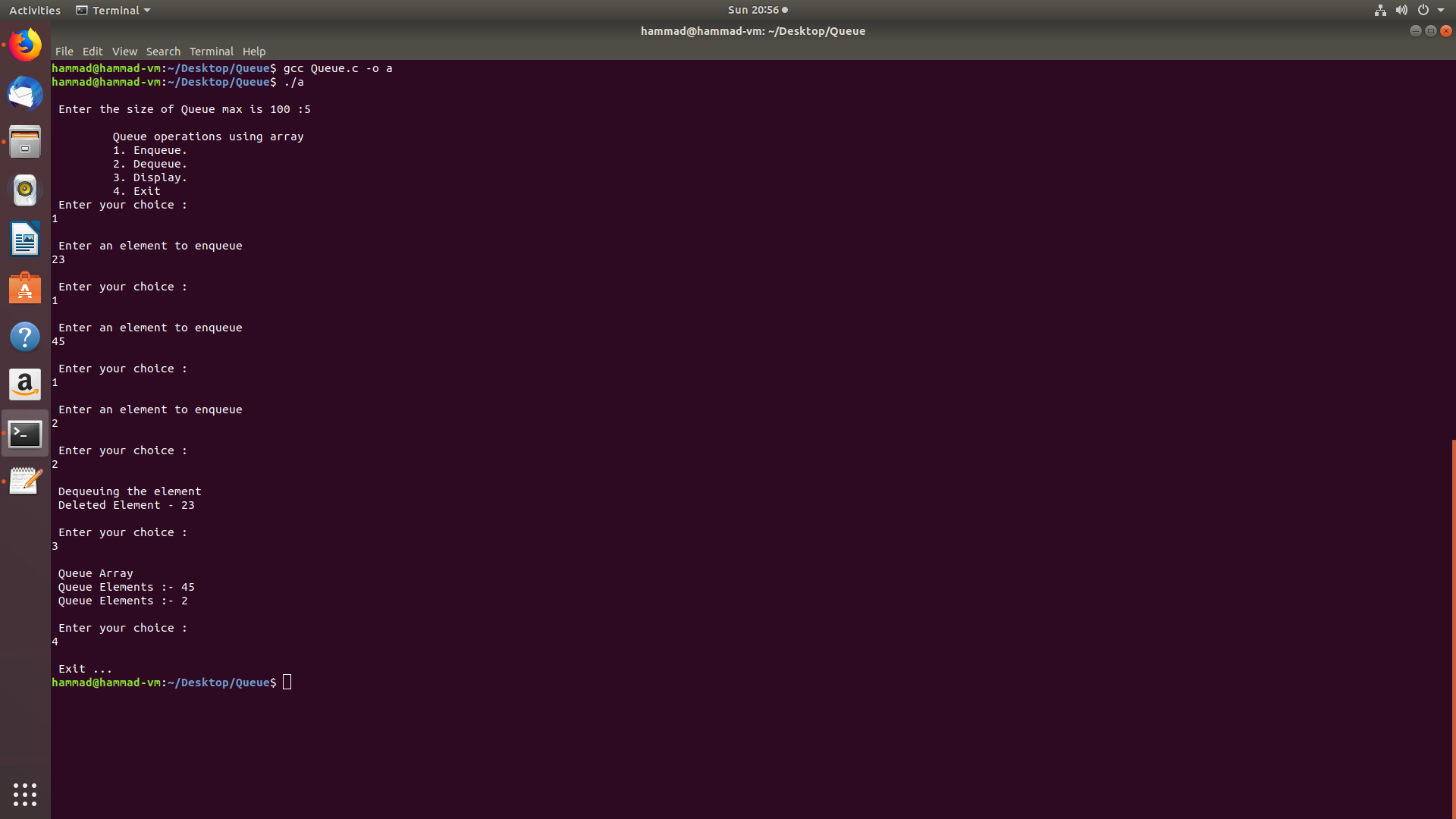
printf(" All elements has been deleted \n");

}

}

}

Screenshot:



1. Doubly Ended Queue

Code:

#include<stdio.h>

int queue[100], front, rear, c, n, x, front2, rear2, i;

void enqueue(int);

void enqueuer(int);

void dequeuer(void);

void dequeue(void);

void display(void);

int main()

{

rear = -1;

front = 0;

printf("\n Enter the size of Queue max is 100 :");

scanf("%d",&n);

rear2 = n;

front2 = n-1;

printf("\n\t Queue operations using array");

printf("\n\t 1. Enqueue.\n\t 11. Right Enqueue.\n\t 2. Dequeue.\n\t 22. Right Dequeue.\n\t 3. Display.\n\t 4. Exit");

do

{

printf("\n Enter your choice :\n");

scanf("%d",&c);

switch(c)

{

case 1:

{

printf("\n Enter an element to enqueue\n");

scanf("%d",&x);

enqueue(x);

break;

}

case 2:

{

printf("\n Dequeuing the element\n");

dequeue();

break;

}

case 3:

{

printf("\n Queue Array\n");

display();

break;

}

case 4:

{

printf("\n Exit ...\n");

break;

}

case 11:

{

printf("\n Enqueuing from right\n");

printf("\n Enter an element to enqueue\n");

scanf("%d",&x);

enqueuer(x);

break;

}

case 22:

{

printf("\n Dequeuing from right\n");

dequeuer();

break;

}

default:

{

printf("\n Invalid Option\n");

break;

}

}

}while(c != 4);

return 0;

}

void enqueue(int p)

{

if(rear >= rear2-1)

{

printf(" Queue overflow.\n");

}

else

{

rear++;

queue[rear] = p;

}

}

void enqueuer(int p)

{

if(rear2 <= rear+1)

{

printf(" Queue Overflow\n");

}

else

{

rear2--;

queue[rear2] = p;

}

}

void dequeue()

{

if(rear>=front)

{

int ele = queue[front];

queue[front] = '\0';

printf(" Deleted Element - %d \n",ele);

front++;

}

else

{

printf(" Queue is underflow\n");

}

}

void dequeuer()

{

if(rear2<=front2)

{

int ele = queue[front2];

queue[front2] = '\0';

printf(" Deleted Element - %d \n",ele);

front2--;

}

else

{

printf(" Queue is underflow\n");

}

}

void display()

{

if(rear<0 && rear2>n)

{

printf(" Queue is empty\n");

}

else

{ if(front <= rear)

{

for(i=front; i<=rear; i++)

{

printf(" Queue Elements :- %d \n",queue[i]);

}

}

else

{

printf(" All elements has been deleted from left side\n");

}

if(front2 >= rear2)

{

for(i=rear2; i<=front2; i++)

{

printf(" Queue Elements R :- %d \n",queue[i]);

}

}

else

{

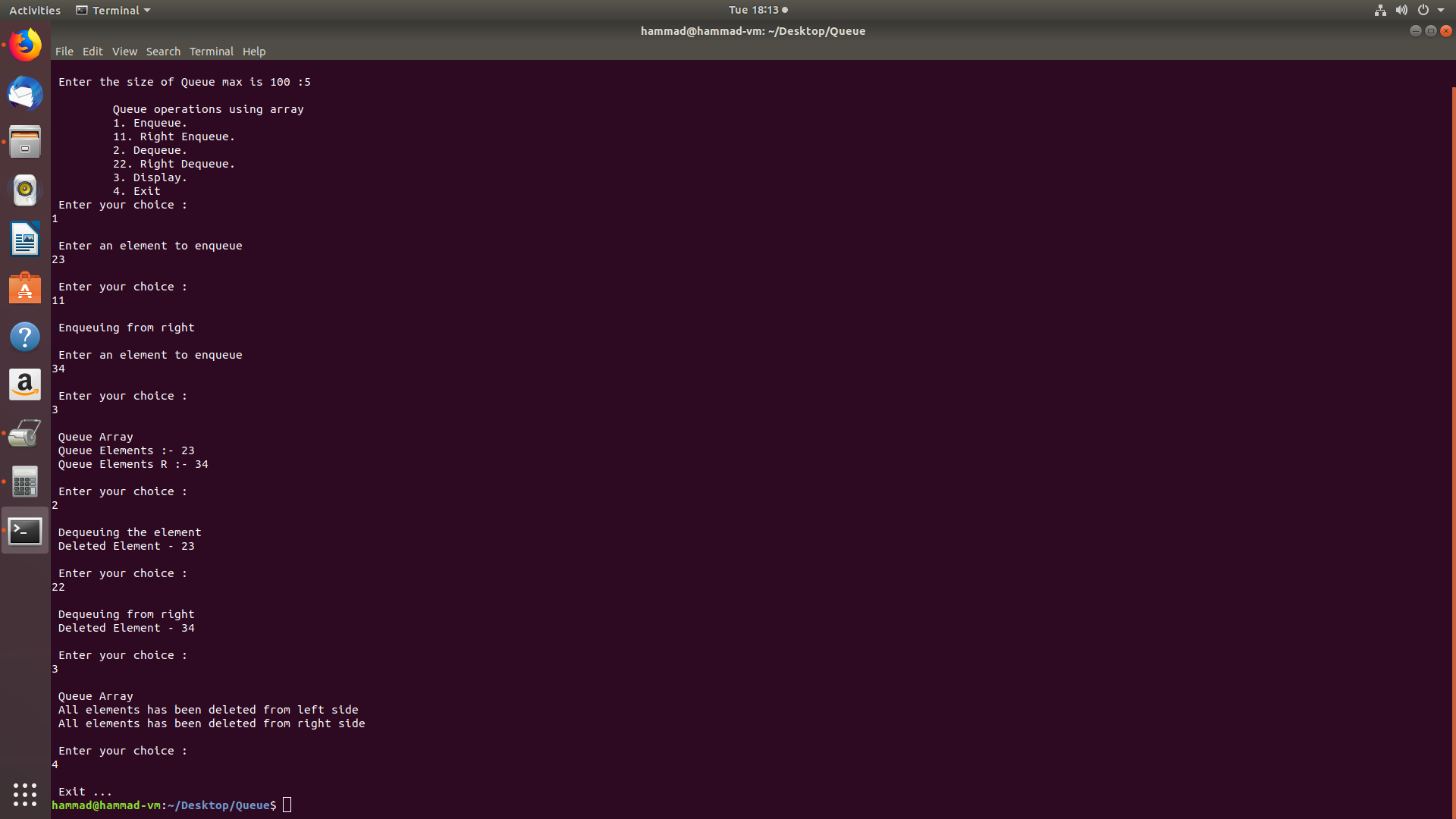
printf(" All elements has been deleted from right side \n");

}

}

}

Screenshot:



1. Circular Queue

Code:

#include<stdio.h>

int queue[100], front, rear, c, n, x, i;

int isEmpty();

void enqueue(int);

void dequeue(void);

void display(void);

int main()

{

rear = -1;

front = -1;

printf("\n Enter the size of Queue max is 100 :");

scanf("%d",&n);

for(i=0;i<=n;i++)

{

queue[i]='\0';

}

printf("\n\t Queue operations using array");

printf("\n\t 1. Enqueue.\n\t 2. Dequeue.\n\t 3. Display.\n\t 4. Exit");

do

{

printf("\n Enter your choice :\n");

scanf("%d",&c);

switch(c)

{

case 1:

{

printf("\n Enter an element to enqueue\n");

scanf("%d",&x);

enqueue(x);

break;

}

case 2:

{

printf("\n Dequeuing the element\n");

dequeue();

break;

}

case 3:

{

printf("\n Queue Array\n");

display();

break;

}

case 4:

{

printf("\n Exit ...\n");

break;

}

default:

{

printf("\n Invalid Option\n");

break;

}

}

}while(c != 4);

return 0;

}

void enqueue(int p)

{

if(isFull())

{

printf("\n Queue is full!! \n");

}

else

{

if(front == -1) front = 0;

rear = (rear + 1) % n;

queue[rear] = p;

printf("\n Inserted -> %d", p);

}

}

void dequeue()

{

int ele;

if(isEmpty()) {

printf("\n Queue is empty !! \n");

return(-1);

}

else

{

ele = queue[front];

if (front == rear){

front = -1;

rear = -1;

}

else {

front = (front + 1) % n;

}

printf("\n Deleted element -> %d \n", ele);

}

}

void display()

{

int i;

if(isEmpty())

{

printf("\n Empty Queue\n");

}

else

{

for( i = front; i!=rear; i=(i+1)%n) {

printf("%d \t",queue[i]);

}

printf("%d ",queue[i]);

}

}

int isFull()

{

if( (front == rear + 1) || (front == 0 && rear == n-1))

{

return 1;

}

return 0;

}

int isEmpty()

{

if(front == -1)

{

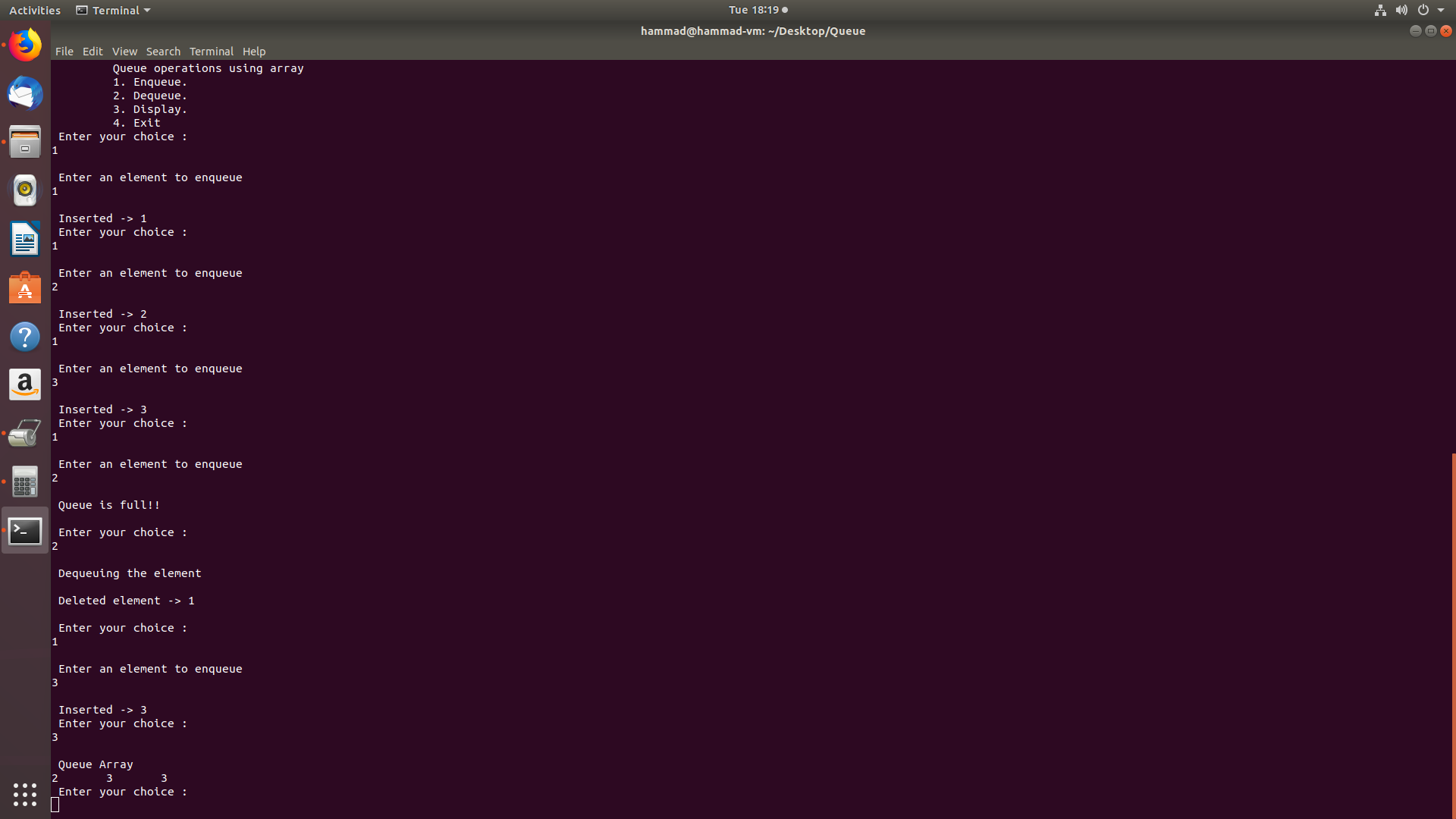
return 1;

}

return 0;

}

Screenshot:



4) Round Robin :-

Code :

#include<stdio.h>

int main()

{

int counter,j,n,time,remain,flag=0,tq;

int wt=0,tat=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

for(counter=0;counter<n;counter++)

{

printf("Enter Arrival Time and Burst Time for Process Process Number %d :",counter+1);

scanf("%d",&at[counter]);

scanf("%d",&bt[counter]);

rt[counter]=bt[counter];

}

printf("Enter Time Quantum:\t");

scanf("%d",&tq);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

for(time=0,counter=0;remain!=0;)

{

if(rt[counter]<=tq && rt[counter]>0)

{

time+=rt[counter];

rt[counter]=0;

flag=1;

}

else if(rt[counter]>0)

{

rt[counter]-=tq;

time+=tq;

}

if(rt[counter]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",counter+1,time-at[counter],time-at[counter]-bt[counter]);

wt+=time-at[counter]-bt[counter];

tat+=time-at[counter];

flag=0;

}

if(counter==n-1)

counter=0;

else if(at[counter+1]<=time)

counter++;

else

counter=0;

}

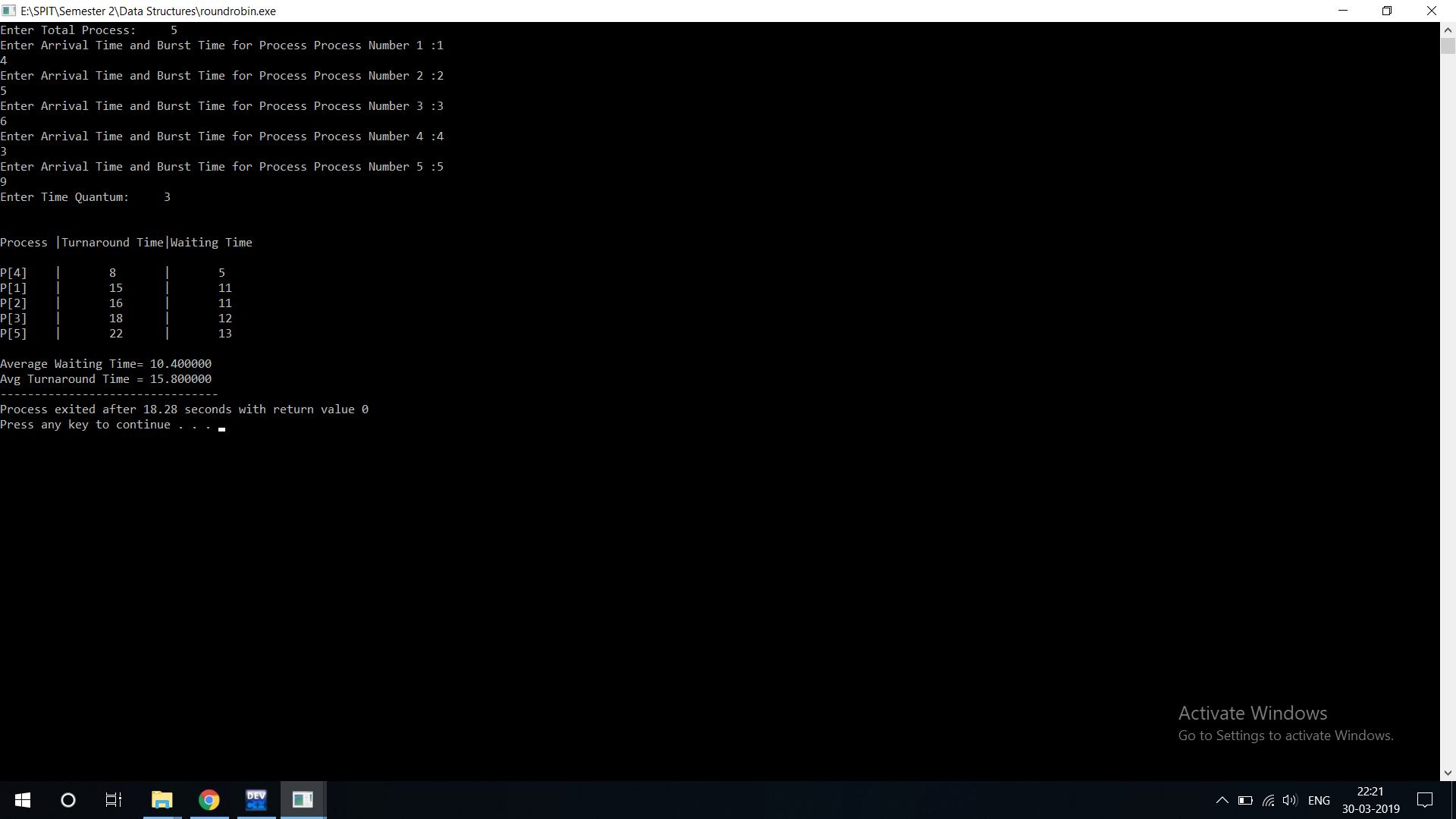
printf("\nAverage Waiting Time= %f\n",wt\*1.0/n);

printf("Avg Turnaround Time = %f",tat\*1.0/n);

return 0;

}

Screenshot :



5) Queue using linked list:

Code:

#include<stdio.h>

#include<stdlib.h>

void enqueue();

void display();

void dequeue();

int counter=0;

// A linked list node

struct Node

{

int data;

struct Node \*next;

}\*temp, \*front = NULL, \*r, \*q, \*p;

//menu driven program for insertion deletion and to display linked list

int main()

{

int ch;

do

{

printf("\n Enter Choice :-\n");

printf("\n 1. Enqueue an element.\n 2. Dequeue an element.\n 3. Display.\n 4. Exit.\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

printf("\n Bye!!!\n");

exit(0);

break;

default:

printf("\n Invalid Choice\n");

break;

}

}while(ch!=4);

}

void enqueue()

{

int data1;

if(front==NULL)

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1; //setting very first element->next to null eg list->data = data1 and list->next = NULL.

p->next = NULL;

front = p;

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1;

p->next = NULL; //NULL because it'll be the last element of the linked list.

q = front;

while(q->next != NULL)

{

q = q->next; //To check pointer q has reached last location of the linked list.

}

q->next = p; //setting last element to p which will be like adding a new element at the last location.

}

}

void dequeue()

{

if(front == NULL)

{

printf("\n Queue Underflow.\n"); //to check whether it is empty or not

}

else

{

q = front;

front = front->next; //since list is pointing at first location and we're storing it in q and pointing out list to next element. Free q will delocate memeory.

if(q->next == NULL)

{

front = NULL;

}

free(q);

printf("\n Dequeued an element.\n");

}

}

void display()

{

if(front==NULL)

{

printf("\n Queue is empty.\n");

}

else

{

printf(" Elements :-\n\n\t\t");

q=front;

counter = 0;

while(q != NULL)

{

printf("%d\t",q->data);

q = q->next;

counter++; // to count total number of elements in the linked list

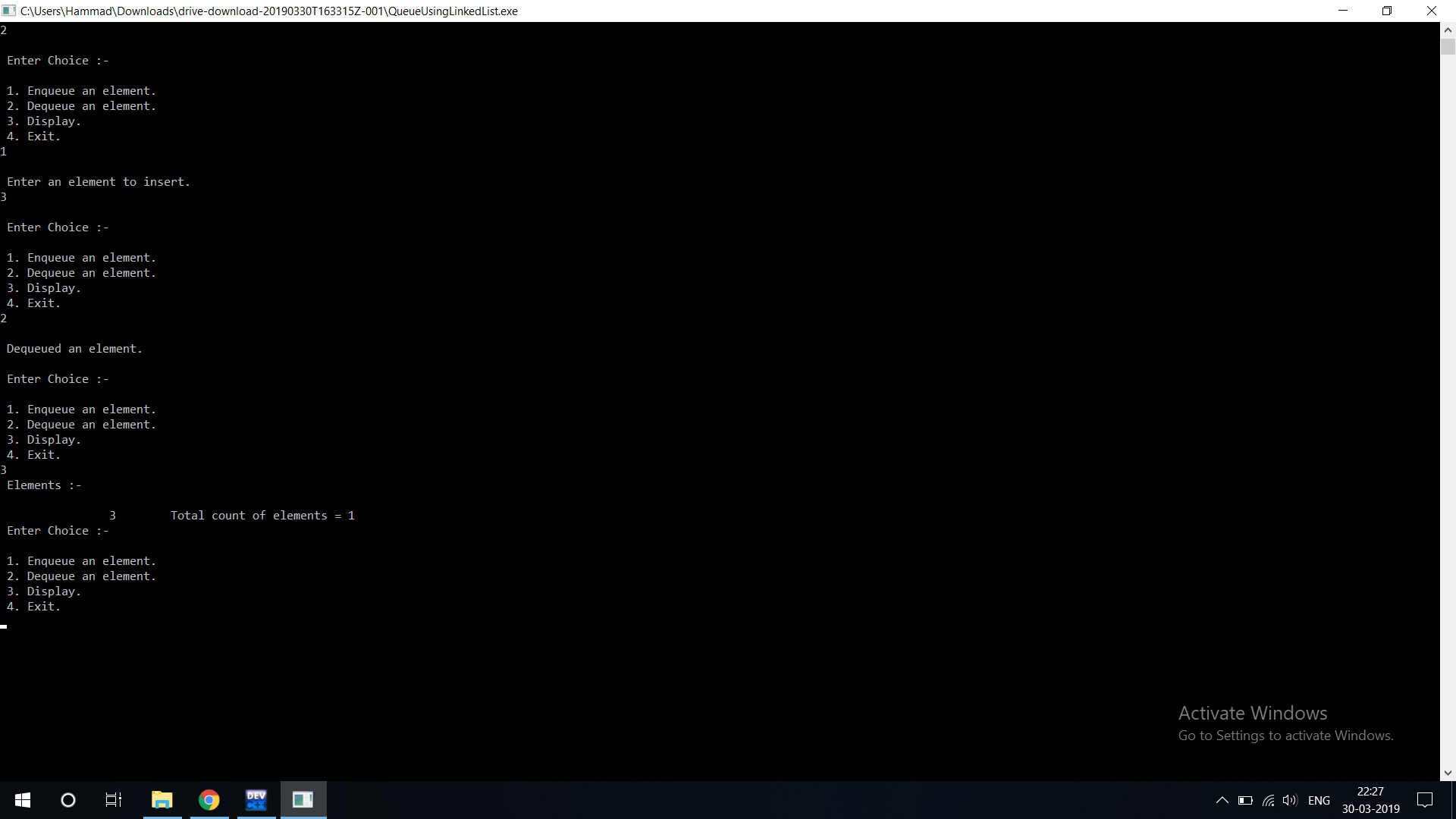
}

printf(" Total count of elements = %d",counter);

}

}

Screenshot:



6) Johnson's Algorithm

Code :

#include<iostream>

#define max 100

using namespace std;

class Johnson{

int a[max],b[max],n,front1,rear1,front2,rear2,S[max],temp1,temp2,data,i,J[max],min1,min2;

bool flag;

public:

Johnson(){

front1 = 0;

rear1= -1;

front2 = n-1;

rear2 = n;

flag = 0;

}

void getdata(){

cout<<"Enter the no of jobs:";

cin>>n;

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

cout<<"Enter job no:";

cin>>J[i];

cout<<"Enter the processing time on first machine:";

cin>>a[i];

cout<<"Enter the processing time on second machine:";

cin>>b[i];

}

}

void l\_enqueue(int data){

if(rear1==rear2-1)

{

cout<<" Queue is Overflow";

}

else

{

rear1++;

// cout<<"Enter element : "<<endl;

// cin>>n;

S[rear1]=data;

cout<<S[rear1];

}

}

void l\_dequeue(int data){

if(front1>rear1)

{

cout<<"Queue is Underflow";

}

else

{

//cout<<"Deleted element is :"<<S[front1++]<<endl;

front1--;

}

}

void r\_enqueue(int data){

if(rear1==rear2-1)

{

cout<<" Queue is Overflow";

}

else

{

rear2--;

// cout<<"Enter element : "<<endl;

// cin>>n;

S[rear2]=data;

}

}

void r\_dequeue(int data){

if(front2<rear2)

{

cout<<" Queue is Underflow";

}

else

{

// cout<<"Deleted element is :"<<S[front2--]<<endl;

front2--;

}

}

void Johnson\_algo(){

for (i=0;i<n;i++)

{

min1=a[0];

if(a[i]< min1)//a[i+1])

{

min1=a[i];

}

cout<<endl<<"temp1"<<min1;

/\* min=ma[0][0];

for (i=0; i<n; i++)

{

for (j=0; j<n1; j++)

{

if(ma[i][j]<min)

min= ma[i][j];

}

}\*/

min2=b[0];

if(b[i]<min2)

{

min2 = b[i];

}

cout<<"temp2"<<min2;

cal();

}

}

void cal()

{

for(i=0;i<n;i++)

{

// if (a[temp1]<b[temp2])

if(min1 < min2)

{

flag = 1;

}

else if(min1 > min2)

{

flag = 0;

}

if (flag == 1)

{

l\_enqueue( min1);

// r\_enqueue (min2);

}

else if(flag==0)

{

// l\_enqueue( min1);

r\_enqueue (min2);

}

//display();

}

display();

}

void display(){

cout<<"Jobs\t"<<"a(i)\t"<<"b(i)\t"<<endl;

for(i=0;i<n;i++)

{

cout<<J[i]<<"\t"<<a[i]<<"\t"<<b[i]<<"\t";

cout<<endl;

}

/\* cout<<"Scheduled time:"<<endl;

if(rear1<front1 && rear2<front2)

{

cout<<"\nUnderflow!";

}

else

{

int i;

for(i=front1;i<=front2;i++)

{

cout<<S[i]<<" "<<endl;

}

}\*/

cout<<endl<<"Queue is :"<<endl;

for(i=front1;i<=rear1;i++)

{

if(front1>rear1)

{

cout<<"Queue is empty :"<<endl;

}

else

{

cout<<S[i]<<" ";

}

}

for(i=rear2;i<=front2;i++)

{

if(front2<rear2)

{

cout<<"Queue is emepty :"<<endl;

}

else

{

cout<<S[i]<<" ";

}

}

cout<<endl<<endl;

}

};

int main(){

Johnson j;

j.getdata();

j.Johnson\_algo();

j.display();

return 0;

}

Screenshot :

