

**DATA STRUCTURES**

**PROJECT**

**BATCH 10**

**BY:**

**1601-21-733-071 Srija.B**

**1601-71-733-072 Chandana.B**

**1601-21-733-081 Aishwarya.N**

**1601-21-733-085 Swathi.S**

**1601-21-733-089 Manasa.V**

**1601-21-733-118 Aditya Ram**

**PROBLEM STATEMENT:**

Alice has two queues, Q and R, which can store integers. Bob gives Alice 50 odd integers and 50 even integers and insists that she store all 100 integers in Q and R. They then play a game where Bob picks Q or R at random and then applies the round-robin scheduler, described in the chapter, to the chosen queue a random number of times. If the last number to be processed at the end of this game was odd, Bob wins. Otherwise, Alice wins. How can Alice allocate integers to queues to optimize her chances of winning? What is her chance of winning?

**ALGORITHM:**

Since linked list is used,each node is a process, which requires unique burst time,completion time so are created in struct node struct node{

int data; int bursttime; int initial\_bursttime; int completion\_time; struct node \*link;

};

main(){

numbers(); create(); assign();

roundrobin();

}

numbers(){ create array\_e using random function containing 50 even numbers create array\_o using random function containing 50 odd numbers combine array\_e and array\_o

}

create(){

input(length of queue Q) length of R=100-length of Q create\_queue(length of Q)

create\_queue(length of R)

}

create\_queue(){

create Q of length\_q using linked list

create R of length\_r using linked list

}

assign(){

input( Enter numbers from array given by bob)

store numbers in queue Q store remaining numbers in R

}

roundrobin(){ input(Select a queue) while( end of queue ){

input(Enter burst time for each number in queue)

store initial\_burst time

initiate completition time to 0

}

input(Enter quantum time)

temp=first element in queue while(){

if(bursttime\_temp > quantumtime){ -->increment completition\_time\_array

-->(bursttime\_temp)=(bursttime\_temp)-quantum\_time

-->increment final\_order\_array

-->temp=temp->link;

}

if(0 <bursttime\_temp <= quantumtime){

|  |  |
| --- | --- |
|  | -->increment completition\_time\_array |
| process) | -->print(completition and waiting time of that particular |
| calculate avg | -->store the completion and waiting time of that process to |
|  | completion and waiting time |
|  | -->delete that node from queue |

}

if( end of queue reached){

return to starting of queue

}

}

}

**DESCRIPTION:**

Two arrays, array\_e and array\_o are created, containing 50 even random numbers and 50 odd numbers, which are given by Bob. Length of queue Q is taken as input, and length of queue R is 100-(length of queue). The queues are created using linked lists and elements from the combined array (even and odd) are added into Q and remaining numbers are added into R.

The node in the linked lists should contain data, burst\_time ,initial\_burst\_time, completion time.

Round Robin scheduling is applied to the selected queue.Quantum time is taken as input, and each and every element in the node is executed till the node’s burst time is over, and each time, the element is noted in the final array. And once the burst time is over, the node is deleted from the queue. The completion (turn around time) and waiting time for each node is calculated. This process is continued till all the nodes in the queue are deleted.

We get the final array, average completion time and average waiting time.

If the number in the final array is even, Alice wins, else Bob wins.

**CODE:**

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

int array\_e[52],array\_o[52],array\_combine[104];

struct node{ //creating a node with elements- data,burst time,initial burst time,completition time

int data;

int bt; int i\_bt; int ct; struct node \*link; };

typedef struct node node;

node \*front\_q=NULL,\*rear\_q=NULL,\*front\_r=NULL,\*rear\_r=NULL; int length\_q,length\_r;

void create\_queue(int length,node \*front,node \*rear){

node \*newnode;

newnode=(node\*)malloc(sizeof(node)); newnode->data=0; int temp=0;

while(temp<length){ //CREATING

QUEUES OF REQUIRED LENGTH if (temp==0){ front=rear=newnode; front->link=NULL; rear->link=NULL;

}

else{

newnode=(node\*)malloc(sizeof(node)); rear->link=newnode; rear=newnode; rear->link=NULL;

}

temp++;

}

if(length==length\_r){

front\_r=front; rear\_r=rear;

}

else if(length==length\_q){

front\_q=front; rear\_q=rear;

}

}

void create(){

printf("\n\n\nEnter the size of queue Q \n\n"); scanf("%d",&length\_q); length\_r=100-length\_q;

create\_queue(length\_q,front\_q,rear\_q); //CREATING TWO QUEUES

create\_queue(length\_r,front\_r,rear\_r);

} void numbers(){

int i=0,j=0,k=0,num; //STORING RANDOM EVEN NUMBERS AND RANDOM ODD NUMBERS IN TWO DIFFERENT ARRAYS while(i<50 || j<50){ num=rand(); if(num%2==0 && i<50){

array\_e[i]=num;

i++;

}

else if(num%2!=0 && j<50){ array\_o[j]=num;

j++;

}

}

printf("\nList of even integers \n");

for(i=0;i<50;i++){ //COMBINING EVEN

AND ODD ARRAY INTO SINGLE ARRAY array\_combine[k]=array\_e[i]; printf("%d.%d ",i+1,array\_e[i]);

k++;

}

printf("\n\n");

printf("\nList of odd integers \n"); for(j=0;j<50;j++){ array\_combine[k]=array\_o[j]; printf("%d. %d ",j+1,array\_o[j]);

k++;

}

printf("\n\n");

printf("\nThe 100 integers Bob gave are\n");

for(k=0;k<100;k++){

printf("%d. %d ",k+1,array\_combine[k]);

}

} void assign(){ int a,i,flag=0,k; node \*temp;

printf("Alice !Please assign numbers in Q from the list BOB gave!!\n"); temp=front\_q; while(temp!=rear\_q->link){

scanf("%d",&a); //ALICE ENTERS

DATA IN QUEUE Q..

temp->data=a; temp=temp->link;

}

printf("The elements in queue Q are: \n"); node \*temp1,\*temp2; temp1=front\_q; while(temp1!=rear\_q->link){ printf("%d ",temp1->data); temp1=temp1->link;

}

printf("\n\n");

printf("The elements in queue R are \n"); //REMAINING ELEMENTS IN THE QUEUE R ARE ADDED..

temp1=front\_q;

//COMBINED ARRAY CONSISTING OF EVEN AND ODD

NUMBERS ARE COMPARED WITH

temp2=front\_r;

//ELEMENTS IN Q, IF THEY ARE NOT SAME, THE ELEMENT IS

ADDED IN QUEUE R int iter; for(i=0;i<100;i++){

flag=0; temp1=front\_q; iter=array\_combine[i]; while(temp1!=rear\_q->link){ if(temp1->data==iter){

flag=1; break;

}

temp1=temp1->link;

}

if(flag==0){ temp2->data=iter; printf("%d ",temp2->data); temp2=temp2->link;

}

}

printf("\n\n");

}

void roundrobin(){

printf("BOB! pick a queue!\n");

int

i=0,j=0,k=0,qt,count=0,sq=0,final[100],ct[100],wt[100],sum=0,temp1,len,sum1

=0; //finalorder array,

node \*temp,\*iter,\*front,\*rear;

//completition time array,waiting time array,quantum time,length..

char c; //of array(Q or

R)

scanf("%s",&c); if(c=='Q'){ front=front\_q; rear=rear\_q; len=length\_q;

}

else if(c=='R'){

front=front\_r; rear=rear\_r; len=length\_r;

}

temp=front;

while(temp!=rear->link){

printf("Enter the burst time for process %d ",temp->data); scanf("%d",&temp->bt);

temp->i\_bt=temp->bt; //STORING INITIAL

BURST TIME

temp->ct=0; //INITIATING

COMPLETITION TIME TO 0

temp=temp->link;

}

printf("Enter the quantum time"); //EACH TIME WE ENCOUNTER A PROCESS, WE ADD IT TO FINAL ARRAY scanf("%d",&qt); temp=front; ct[0]=0; wt[0]=0; while(temp!=NULL){

if((temp->bt)>qt){ //IF REMAINING BURST TIME OF THE PROCESS IS GREATER THAN QUANTUM

temp1=ct[count]; //TIME,FIRST, WE

INREMENT THE CT ARRAY,THEN REMANINING BURST TIME

count++; //CALCULATED..

ct[count]=temp1+qt; (temp->bt)=(temp->bt)-qt; final[i]=temp->data;

|  |  |  |
| --- | --- | --- |
| temp=temp->link;  DONE, WE PROCEES TO NEXT NODE |  | //PROCESS EXECUTION |
| if(temp==NULL){ |  | //IF WE REACH |

LAST NODE, WE AGAIN START FROM FIRST

temp=front;

}

i++;

continue;

}

else if((temp->bt)==qt || 0<=(temp->bt)<qt){

temp1=ct[count]; count++;

ct[count]=temp1+temp->bt;

printf("\nThe completion time of %d is %d\n",temp-

>data,ct[count]);

printf("\nThe waiting time of %d is %d\n",temp-

>data,ct[count]-temp->i\_bt);

sum1=sum1+ct[count]-temp->i\_bt; //IF BURST TIME ID LESS THAN OR EQUAL TO QUANTUM TIME,

sum=sum+ct[count]; //THE

COMPLETITION AND WAITING TIME OF THE PROCESS IS RECORDED

|  |  |  |
| --- | --- | --- |
| final[i]=temp->data;  PROCESS IS REMOVED FROM THE QUEUE  iter=front; |  | //AND THE |
| if(temp==front){  PROCESS IS FRONT NODE |  | //REMOVAL IF |

temp=temp->link; front=temp;

}

else{

while(iter!=temp->link)

{

if(iter->link==temp){

iter->link=temp->link; //REMOVAL

IF PROCESS IS OTHER THAN FRONT NODE

free(temp); temp=iter->link;

break;

}

iter=iter->link;

}

}

i++;

if(temp==NULL){ temp=front;

}

}

}

printf("The final array is"); //FINAL ARRAY CONSISTING OF

THE ORDER OF THE PROCESSES IS DISPLAYED for(k=0;k<i;k++){ printf("%d ",final[k]);

}

printf("\n\nThe avg completition time is %d/%d \n",sum,len); printf("\n\nThe avg waiting time is %d/%d \n",sum1,len); if(final[i-1]%2!=0){

printf("\n\n BOB WON!!Sorry ALICE! better luck next time!!");

//IF THE LAST ELEMENT IN THE FINAL ARRAY IS ODD, BOB WINS ELSE, ALICE WINS

}

else{

printf("\n\nALICE WON!! Sorry BOB better luck next time!!");

}

}

void main(){

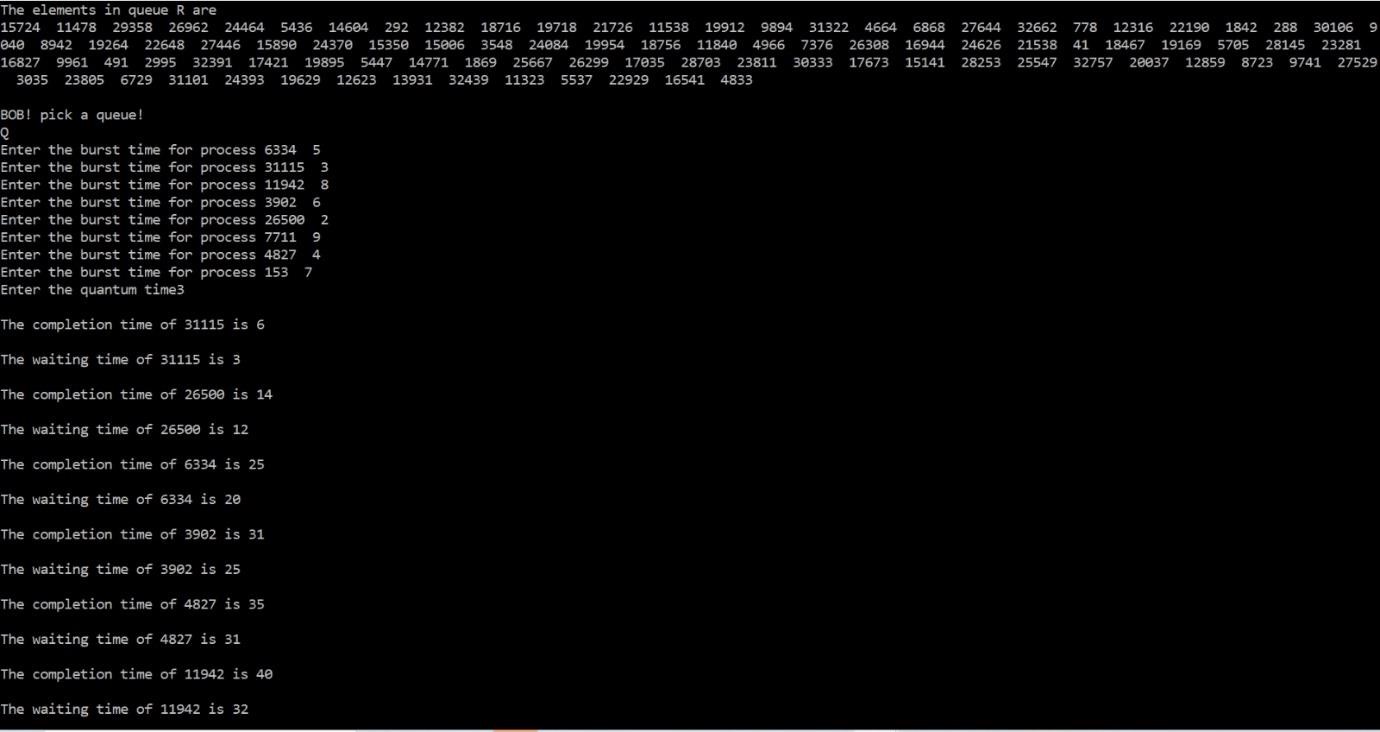
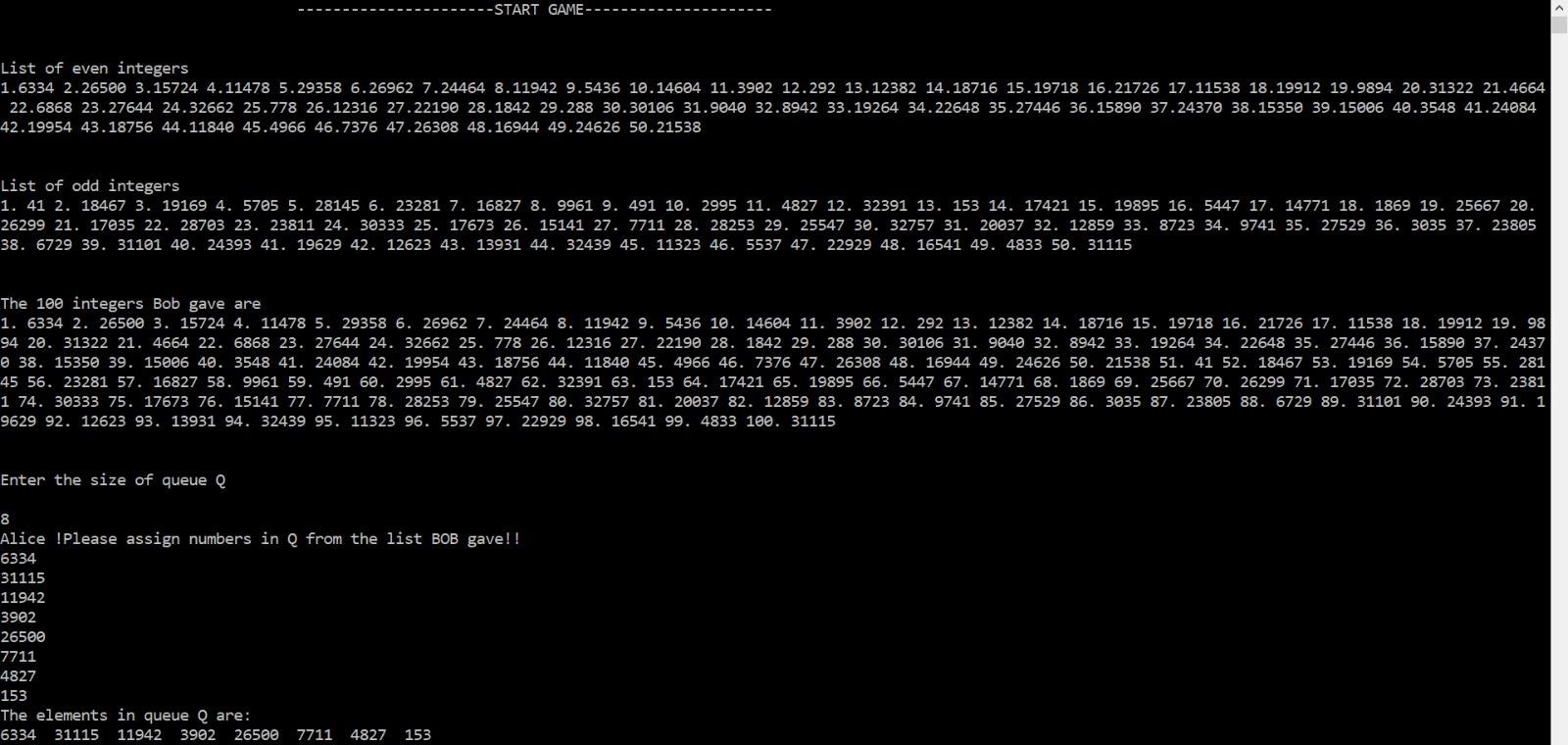
printf(" ----------------------START GAME--------------

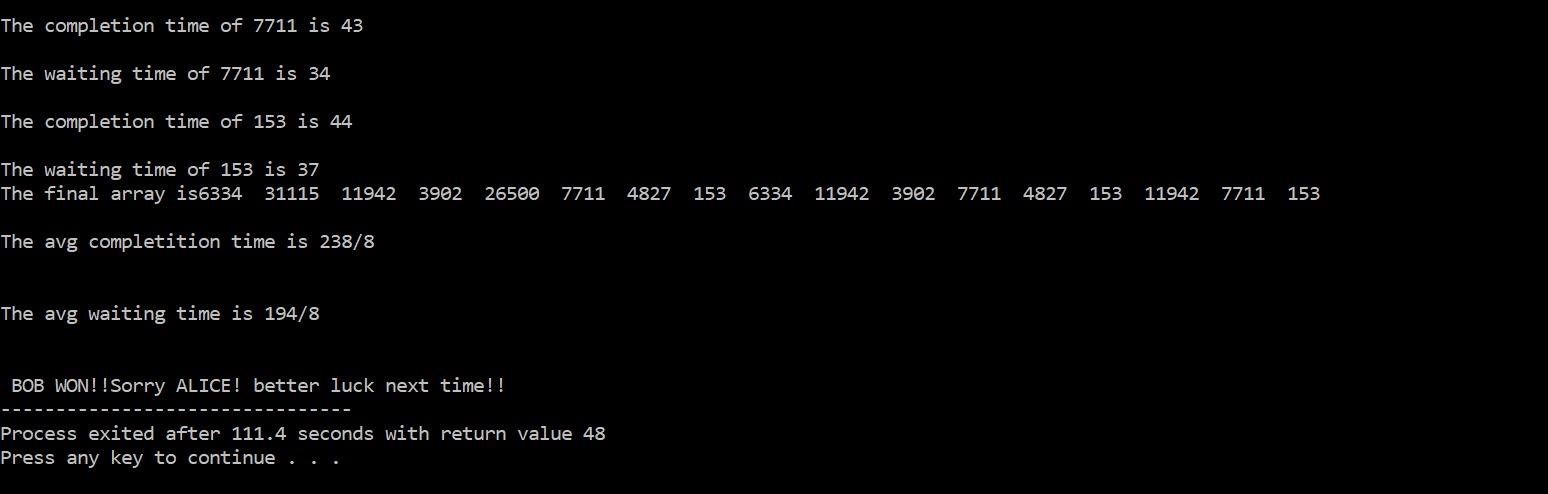
-------");

printf("\n\n"); numbers(); create(); assign(); roundrobin();

}

**OUTPUT:**





How can Alice allocate integers to queues to optimize her chances of winning?

What is her chance of winning?

**TEST CASE:**

Alice should put on an even integer in Q and all the other 99 integers in R.

The chance that Bob picks Qis 0.5, where Alice must win.

The chance that Bob pocks R is also 0.5, where Alice wins at the chance of 49/99 (49 even integers out of 99 overall integers).

This gives her 0.5\*(1+49/99) = 74/99 chance of winning.

**ALANYSIS:**

Time complexity of best case is O(1), which is the test case.

Average time complexity is O(burst\_time/quantum\_time) if the burst times of all the processes are taken to be same.

**SOLUTION:**

