Ques. Write a program for implementing the FIFO page replacement Algorithm.

```
#include <iostream>
#include <vector>
using namespace std;
int IsPresent(vector<int> v, int key)
{
    int size = v.size();
    for (int i = 0; i < size; i++)
        if (v[i] == key)
            return 1;
    return 0;
}
int main()
{
    int ms, nop, count = 0, f = 0, s = 0;
    cout << "input cache size :";
    cin >> ms;
    vector<int> cv(ms, -1);
    cout << "Input no of processes :";
    cin >> nop;
    cout << "Input processes" << endl;
    vector<int> pv(nop);
    for (int i = 0; i < nop; i++)
        cin >> pv[i];
    for (int i = 0; i < nop; i++)
    {
        if (f < ms)
        {
            int found = IsPresent(cv, pv[i]);
            if (!found)
            {
                cv[f] = pv[i];
                f++;
                count++;
            }
        }
        else
        {
            int found = IsPresent(cv, pv[i]);
            if (!found)
            {
                cv[s] = pv[i];
                f++;
                count++;
                s++;
            }
```

```
        }
        if (s == ms)
            s = 0;
    }
    cout << "No. of misses (Pages Faults) is :" << count<<endl;
    cout << "No. of hits is :" << nop - count << endl;
    return 0;
}
```

Output Clear

```
$ input cache size :3
Input no of processes :7
Input processes
1 3 0 3 5 6 3
No. of misses (Pages Faults) is :6
No. of hits is :1
```

Output Clear

```
input cache size :3
Input no of processes :10
Input processes
4 7 6 1 7 6 1 2 7 2
No. of misses (Pages Faults) is :6
No. of hits is :4
```

Ques. Write a program for implementing the Round Robin Scheduling Algorithm.

```
#include <stdio.h>
int dequeue(int front, int back, int n)
{
   if (front == back || front == -1)
      front = -1;
   else
      front = (front + 1) % n;
   return front;
}
int enqueue(int queue[], int v, int front, int back, int n)
{
   int i;
   if (front != -1)
   {
      for (i = front; i != back; i = (i + 1) % n)
         if (v == queue[i])
            return back;
      if (queue[back] == v)
         return back;
   }
   if ((back + 1) % n != front)
   {
      back = (back + 1) % n;
      queue[back] = v;
   }
   return back;
}
int main()
{
   int queue[50];
   int front = -1, tail = -1, v, n, i, j, qs = 50;
   int pid[10], at[10], bt[10], bt2[10], ct[10], tat[10], wt[10], vis[10] = {0};
   int noOfZeroes = 0, tq, t = 0, temp, totaltat = 0, totalwt = 0;
   printf("Input no of Process :");
   scanf("%d", &n);
   printf("Input Quantum time :");
   scanf("%d", &tq);
   printf("AT BT\n");
   for (i = 0; i < n; i++)
      pid[i] = i;
   for (i = 0; i < n; i++)
   {
      scanf("%d", &at[i]);
      scanf("%d", &bt[i]);
   }
```

```
for (i = 0; i < n; i++)
    bt2[i] = bt[i];

for (i = 0; i < n - 1; i++)
{
    for (j = 0; j < n - i - 1; j++)
    {
        if (at[j] > at[j + 1])
        {
            temp = at[j];
            at[j] = at[j + 1];
            at[j + 1] = temp;

            temp = bt[j];
            bt[j] = bt[j + 1];
            bt[j + 1] = temp;

            temp = pid[j];
            pid[j] = pid[j + 1];
            pid[j + 1] = temp;
        }
    }
}

t += at[0];
tail = enqueue(queue, 0, front, tail, qs);
if (tail == 0 && front == -1)
    front = 0;

while (noOfZeroes != n)
{
    if (front != -1)
    {
        j = queue[front];
        if (bt[j] >= tq)
        {
            t += tq;
            bt[j] -= tq;
        }
        else
        {
            t += bt[j];
            bt[j] = 0;
        }
        if (bt[j] == 0)
        {
            ct[j] = t;
            vis[j] = 1;
```

```
            noOfZeroes++;
        }
        front = dequeue(front, tail, qs);
        if (front == -1)
            tail = -1;
        for (i = 0; i < n; i++)
        {
            if (i == j)
                continue;
            if (vis[i] == 0 && at[i] <= t && bt[i] != 0)
            {
                tail = enqueue(queue, i, front, tail, qs);
                if (tail == 0 && front == -1)
                    front = 0;
            }
        }
        if (bt[j] != 0)
        {
            tail = enqueue(queue, j, front, tail, qs);
            if (tail == 0 && front == -1)
                front = 0;
        }
    }
}

for (i = 0; i < n; i++)
{
    tat[i] = ct[i] - at[i];
    totaltat += tat[i];
}

for (i = 0; i < n; i++)
{
    wt[i] = tat[i] - bt2[i];
    totalwt += wt[i];
}

printf("\n%5s%5s%5s%5s%5s%5s\n", "pid", "at", "bt", "ct", "tat", "wt");
for (i = 0; i < n; i++)
{
    printf("%5d%5d%5d%5d%5d%5d\n", i, at[pid[i]], bt2[pid[i]], ct[pid[i]], tat[pid[i]], wt[pid[i]]);
}
printf("Avarage Turn Around Time : %0.3lf\n", totaltat * 1.0 / n);
printf("Avarage Waiting Time : %0.3lf\n", totalwt * 1.0 / n);

return 0;
}
```

Output Clear

```
$ Input no of Process :4
Input Quantum time :2
AT BT
0 5
1 4
2 2
4 1

pid  at   bt   ct  tat   wt
0    0    5   12   12    7
1    1    4   11   10    6
2    2    2    6    4    2
3    4    1    9    5    4
Avarage Turn Around Time : 7.750
Avarage Waiting Time : 4.750
```

Ques. Write a program for implementing the SRTF (Shortest Remaining Time First) Scheduling Algorithm.

```
#include <iostream>
#include <vector>
#include <stdbool.h>
#include <algorithm>
using namespace std;
class DataDetails
{
public:
    int ari, pno, bur, tempbur;
    int ct, tat, wt;
    int visit;
};
bool comparator(DataDetails d1, DataDetails d2)
{
    if (d1.bur != d2.bur)
        return (d1.bur < d2.bur);
    else
    {
        if (d1.ari != d2.ari)
            return (d1.ari < d2.ari);
        return (d1.pno < d2.pno);
    }
}
bool comparatorPno(DataDetails d1, DataDetails d2)
{
    return (d1.pno < d2.pno);
}
int main()
{
    cout << " SRTF SCHEDULING" << endl;
    int size, t = 0, ch = 0;
    float avgtat = 0, avgwt = 0;
    cout << "Enter no of process :";
    cin >> size;
    cout << "AT BT" << endl;
    vector<DataDetails> vrr(size);
    for (int i = 0; i < size; i++)
    {
        cin >> vrr[i].ari >> vrr[i].bur;
        vrr[i].pno = i + 1;
        vrr[i].ct = vrr[i].tat = vrr[i].wt = 0;
        vrr[i].visit = 0;
        vrr[i].tempbur = vrr[i].bur;
    }
    sort(vrr.begin(), vrr.end(), comparator);
```

```
    while (1)
    {
        int ch = 0;
        int br = 0;
        for (int i = 0; i < size; i++)
        {
            if (vrr[i].bur != 0)
                br = 1;
            if ((vrr[i].bur != 0) && (vrr[i].visit == 0) && t >= vrr[i].ari)
            {
                t += 1;
                vrr[i].bur -= 1;
                if (vrr[i].bur == 0)
                {
                    vrr[i].ct = t;
                    vrr[i].tat = vrr[i].ct - vrr[i].ari;
                    vrr[i].wt = vrr[i].tat - vrr[i].tempbur;
                    avgtat += vrr[i].tat;
                    avgwt += vrr[i].wt;
                    vrr[i].visit = 1;
                    ch = 1;
                }
                break;
            }
        }
        if (!br)
            break;
        sort(vrr.begin(), vrr.end(), comparator);
    }
    sort(vrr.begin(), vrr.end(), comparatorPno);
    cout << endl;
    cout << " SOLUTION" << endl;
    cout << "PN "
         << "AT "
         << "BT "
         << "CT "
         << "TAT "
         << "WT " << endl;
    for (int i = 0; i < size; i++)
          cout << "P" << vrr[i].pno << " " << vrr[i].ari << " " << vrr[i].tempbur << " " << vrr[i].ct << "
" << vrr[i].tat << " " << vrr[i].wt << " " << endl;
    avgtat = avgtat / size;
    cout << "Average TurnAroundTime is :" << avgtat << endl;
    avgwt = avgwt / size;
    cout << "Average WaitingTime is :" << avgwt << endl;
    return 0;
}
```

Output Clear

```
$  SRTF SCHEDULING
Enter no of process :6
AT BT
0  8
1  4
2  2
3  1
4  3
5  2

 SOLUTION
PN  AT BT CT  TAT WT
P1  0  8  20  20  12
P2  1  4  10  9   5
P3  2  2  4   2   0
P4  3  1  5   2   1
P5  4  3  13  9   6
P6  5  2  7   2   0
Average TurnAroundTime is :7.333
Average WaitingTime is :4
```