EXPERIMENT 3

Input the processes along with their burst time. Find out the average waiting time and turn around time using

(a) FCFS algorithm

(b) SJF algorithm

Compare and conclude the results.

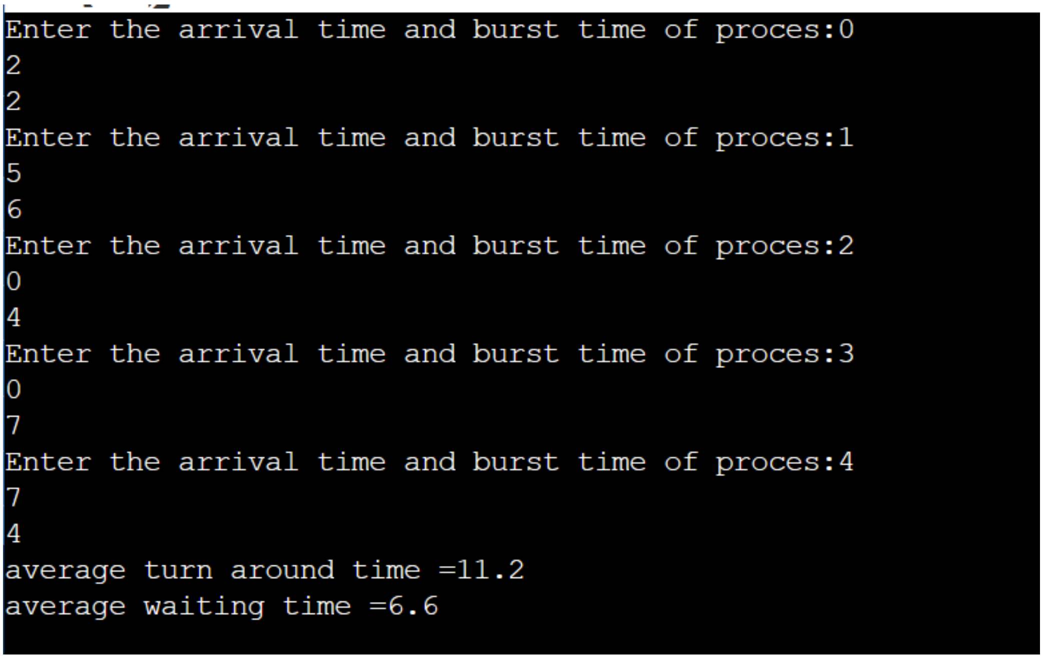
Solution:

(a) For FCFS method:

Code –

```
#include <bits/stdc++.h>

using namespace std;

int main()
{

   vector<pair<int,int>>atbt(5);
   int tat[5];
   int waitingtime[5];
   int endtime[5];

   //av tat av wt?
   for(int i=0;i<5;i++){
   cout<<"Enter the arrival time and burst time of proces:"<<i<<endl;
   cin>>atbt[i].first;
   cin>>atbt[i].second;
   }

   sort(atbt.begin(),atbt.end());
   endtime[0]=atbt[0].first+atbt[0].second;
   for(int i=0;i<4;i++){
      if(endtime[i]<atbt[i+1].first){
         endtime[i+1]=atbt[i+1].first+atbt[i+1].second;
      }
      else{
         endtime[i+1]=endtime[i]+atbt[i+1].second;
      }

   }
   for(int i=0;i<5;i++){
      tat[i]=endtime[i]-atbt[i].first;
      waitingtime[i]=tat[i]-atbt[i].second;

    }

    double atat=0;
    double awt=0;
    for(int i=0;i<5;i++){
    atat=atat+tat[i];
    }
    atat=atat/5;

    for(int i=0;i<5;i++){
    awt=awt+waitingtime[i];
    }
    awt=awt/5;

    cout<<"average turn around time ="<<atat<<endl;
    cout<<"average waiting time ="<<awt<<endl;
    return 0;
}
```

Output –



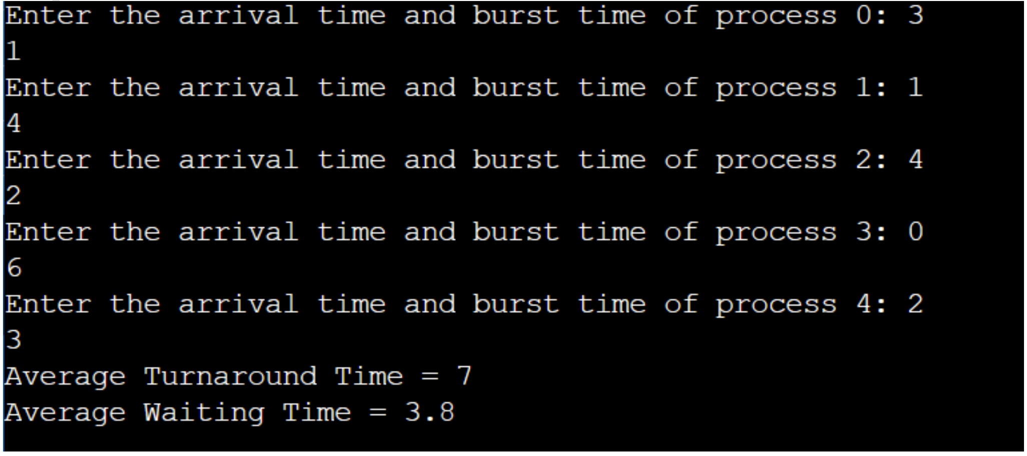
```
Enter the arrival time and burst time of proces:0
2
2
Enter the arrival time and burst time of proces:1
5
6
Enter the arrival time and burst time of proces:2
0
4
Enter the arrival time and burst time of proces:3
0
7
Enter the arrival time and burst time of proces:4
7
4
average turn around time =11.2
average waiting time =6.6
```

(b) For SJF Method:

Code –

```
#include <bits/stdc++.h>

using namespace std;

struct Process {
    int id;
    int aT;
    int bT;
    int remainingTime;
};

bool compareAT(const Process& a, const Process& b) {
    return a.aT < b.aT;
}

int main() {
    vector<Process> processes(5);
    int tat[5] = {0};
    int wT[5] = {0};
    int currentTime = 0;
    int completed = 0;

    for (int i = 0; i < 5; i++) {
        processes[i].id = i;
        cout << "Enter the arrival time and burst time of process " << i << ": ";
        cin >> processes[i].aT >> processes[i].bT;
        processes[i].remainingTime = processes[i].bT;
    }

    sort(processes.begin(), processes.end(), compareAT);

    while (completed < 5) {
        int shortPrIdx = -1;
        int shortestBurst = INT_MAX;

        for (int i = 0; i < 5; i++) {
            if (processes[i].aT <= currentTime && processes[i].remainingTime < shortestBurst &&
processes[i].remainingTime > 0) {
                shortPrIdx = i;
                shortestBurst = processes[i].remainingTime;
            }
        }
        if (shortPrIdx == -1) {
            currentTime++;
        } else {
            // Execute the shortest job for 1 unit of time (preemptive)
            processes[shortPrIdx].remainingTime--;
            currentTime++;

            if (processes[shortPrIdx].remainingTime == 0) {
                completed++;
                int currentProcess = shortPrIdx;
                int turnaroundTime = currentTime - processes[currentProcess].aT;
                int waiting = turnaroundTime - processes[currentProcess].bT;
                tat[currentProcess] = turnaroundTime;
                wT[currentProcess] = waiting;
            }
        }
    }

    double averageTAT = 0;
    double averageWT = 0;

    for (int i = 0; i < 5; i++) {
        averageTAT += tat[i];
        averageWT += wT[i];
    }

    averageTAT /= 5;
    averageWT /= 5;

    cout << "Average Turnaround Time = " << averageTAT << endl;
    cout << "Average Waiting Time = " << averageWT << endl;

    return 0;
}
```

Output –



```
Enter the arrival time and burst time of process 0: 3
1
Enter the arrival time and burst time of process 1: 1
4
Enter the arrival time and burst time of process 2: 4
2
Enter the arrival time and burst time of process 3: 0
6
Enter the arrival time and burst time of process 4: 2
3
Average Turnaround Time = 7
Average Waiting Time = 3.8
```

EXPERIMENT 4

Input the processes along with their burst time. Find out the average waiting time and turn aroundtime using

(a) Round Robbin algorithm

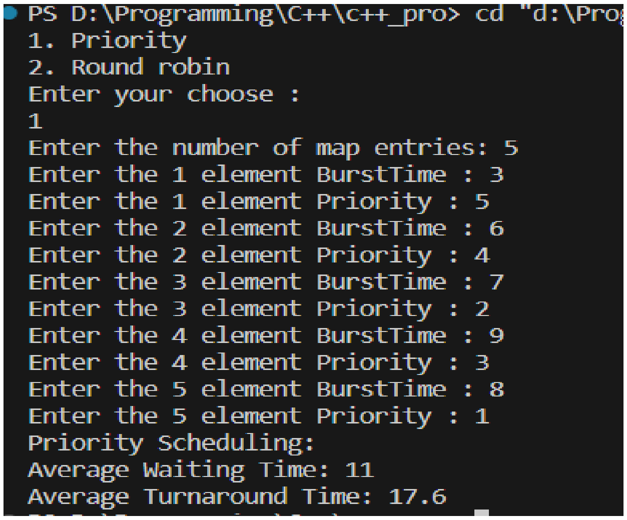
(b) Priority algorithm

```
#include <bits/stdc++.h>

using namespace std;
void Priority();
void Round_robin();

void Round_robin()
{
   int numEntries, ts;
   cout << "Enter the TS: ";
   cin >> ts;
   cout << "Enter the number of map entries: ";
   cin >> numEntries;

   queue<int> myqueue;
   int bt[numEntries];
   for (int i = 0; i < numEntries; i++)
   {
      int key1;
      cout << "Enter the " << i + 1 << " element BurstTime : ";
      cin >> key1;
      myqueue.push(key1);
      bt[i] = key1;
   }
   float current = 0;
   vector<int> tat;
   while (!myqueue.empty())
   {
      if (myqueue.front() == 0)
      {
         tat.push_back(current);
         myqueue.pop();
      }
```

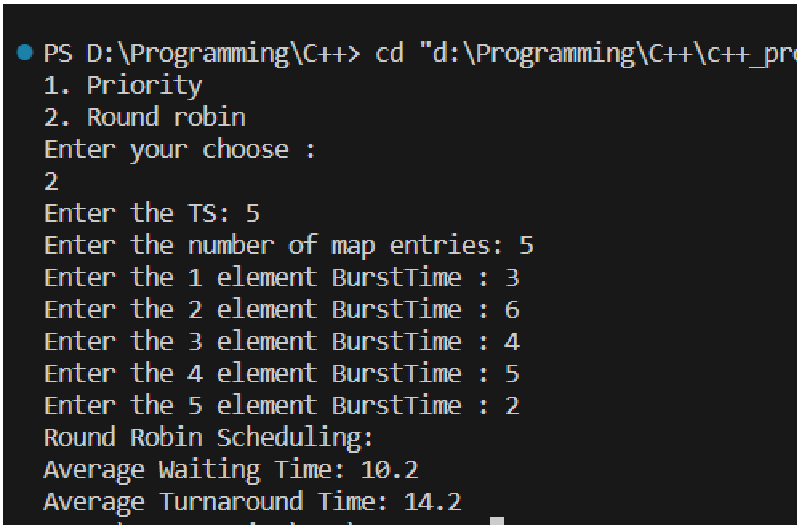
```
        else if (myqueue.front() == 1)
        {
            current += 1;
            tat.push_back(current);
            myqueue.pop();
        }
        else
        {
            current += ts;
            int temp = myqueue.front() - ts;
            if (temp == 0)
            {
                tat.push_back(current);
                myqueue.pop();
            }
            else
            {
                myqueue.pop();
                myqueue.push(temp);
            }
        }
        cout << myqueue.front() << endl;
    }

    sort(bt, bt + numEntries);
    float total_tat = accumulate(tat.begin(), tat.end(), 0);

    int wt[numEntries];
    float total_wt = 0;
    for (int i = 0; i < tat.size(); i++)
    {
        wt[i] = tat[i] - bt[i];
        total_wt += wt[i];
    }

    cout << "Round Robin Scheduling:" << endl;
    cout << "Average Waiting Time: " << total_wt / numEntries << endl;
    cout << "Average Turnaround Time: " << total_tat / numEntries << endl;
}

struct CompareMap
{
    bool operator()(const pair<int, int> &a, const pair<int, int> &b) const
    {
        if (a.second != b.second)
        {
            return a.second > b.second; // Sort by values in descending order
        }
        return a.first < b.first; // Sort by keys in ascending order when values are the same
    }
};

void Priority()
{
    map<pair<int, int>, int, CompareMap> myMap;
    vector<map<pair<int, int>, int>::iterator> it;

    int numEntries;
    cout << "Enter the number of map entries: ";
    cin >> numEntries;
    int total = 0;
    vector<int> pre_tat;
    vector<int> post_tat;
    vector<int> wait;
    vector<int>::iterator it1;

    for (int i = 0; i < numEntries; i++)
    {
        int key1, key2;
        cout << "Enter the " << i + 1 << " element BurstTime : ";
        cin >> key1;
        cout << "Enter the " << i + 1 << " element Priority : ";
        cin >> key2;
        myMap[make_pair(key1, key2)] = i; // Value is set to the loop index for demonstration
        total = total + key1;
    }

    int sum = 0;
    cout << "\nOriginal map:" << endl;
    // for (const auto &entry : myMap)
    // {
    //     cout << entry.first.first << ", " << entry.first.second << endl;
    // }
    pre_tat.push_back(0);
    for (auto it = myMap.begin(); it != myMap.end(); it++)
    {
        sum = sum + it->first.first;
        pre_tat.push_back(sum);
    }

    sort(pre_tat.begin(), pre_tat.end(), greater<int>());

    float total_tat = 0;
    float total_wt = 0;
    for (int i = 0; i < numEntries; i++)
    {
        total_tat += pre_tat[i];
    }
    int sum1 = 0;
    int i = 0;
    int j = -1;
    for (auto it = myMap.rbegin(); it != myMap.rend(); ++it)
    {
        do
        {
            sum1 = pre_tat[i] - it->first.first;
            cout << sum1 << endl;
            wait.push_back(sum1);
            i = i + 1;

        } while (i < j);
    }
    for (int i = 0; i < numEntries; i++)
    {
        total_wt += wait[i];
    }
    cout << "Priority Scheduling:" << endl;
    cout << "Average Waiting Time: " << total_wt / numEntries << endl;
    cout << "Average Turnaround Time: " << total_tat / numEntries << endl;
}

int main()
{
    int n;
    cout << "Enter your choose : " << endl;
    cin >> n;

    switch (n)
    {
    case 1:
        Priority();
        break;
```

```
   case 2:
      Round_robin();
      break;
   default:
      cout << "Choose given one." << endl;
   }
   return 0;
}
```

Output :-



```
PS D:\Programming\C++\c++_pro> cd "d:\Pro
1. Priority
2. Round robin
Enter your choose :
1
Enter the number of map entries: 5
Enter the 1 element BurstTime : 3
Enter the 1 element Priority : 5
Enter the 2 element BurstTime : 6
Enter the 2 element Priority : 4
Enter the 3 element BurstTime : 7
Enter the 3 element Priority : 2
Enter the 4 element BurstTime : 9
Enter the 4 element Priority : 3
Enter the 5 element BurstTime : 8
Enter the 5 element Priority : 1
Priority Scheduling:
Average Waiting Time: 11
Average Turnaround Time: 17.6
```



```
PS D:\Programming\C++> cd "d:\Programming\C++\c++_pr
1. Priority
2. Round robin
Enter your choose :
2
Enter the TS: 5
Enter the number of map entries: 5
Enter the 1 element BurstTime : 3
Enter the 2 element BurstTime : 6
Enter the 3 element BurstTime : 4
Enter the 4 element BurstTime : 5
Enter the 5 element BurstTime : 2
Round Robin Scheduling:
Average Waiting Time: 10.2
Average Turnaround Time: 14.2
```