

**OPERATING SYSTEMS**

**Lab File**

**Submitted to**: Dr. Dinesh Udar

**Submitted by:** Ashish Gupta

2k16/MC/023

Department of Applied Mathematics

Delhi Technological University

**Vision and Mission**

**Vision**

Department of Computer Science & Engineering to be a leading world class technology department playing its role as a key node in national and global knowledge network, thus empowering the computer science industry with the wings of knowledge and power of innovation

**Mission**

* To nurture talent of students for research, innovation and excellence in the field of computer engineering starting from Under graduate level.
* To develop highly analytical and qualified computer engineers by imparting training on cutting edge technology.
* To produce socially sensitive computer engineers with professional ethics.
* To focus on R&D environment in close partnership with industry and foreign universities.
* To produce well-rounded, up to date, scientifically tempered, design oriented engineers and scientists capable of lifelong learning

**Program Educational Objectives (PEOs)**

PEO 1: To acquire in-depth knowledge of software and hardware techniques which provide a strong foundation to pursue continuing education and nurture the talent for innovation and research.

PEO 2: To nurture the talent in leadership qualities, at an appropriate level in order to address the issues in a responsive, ethical and innovative manner.

PEO 3: To excel in careers by being a part of success and growth of an organization with whom they will be associated.

PEO 4: To inculcate the ability for lifelong learning by active participation in self-study courses, seminars, research papers

**Program Specific Outcome (PSOs)**

* PSO 1: Design, analyze and develop the engineering problems.
* PSO 2: Specify, design, develop, test and maintain usable systems that behave reliably and efficiently and satisfy all the requirements that customers have defined for them.
* PSO 3: Develop software systems that would perform tasks related to Research, Education and Training and/or E-governance.

**Program Outcomes**

* **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
* **Problem analyses:** Identify, formulate, review, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
* **Design/development of solution:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal and environmental considerations.
* **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
* **Modern tool usage:** Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
* **The engineer and the society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
* **Environment and sustainability:** Understand the impacts of professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
* **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
* **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
* **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
* **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
* **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| SNO. | PROGRAM | DATE | SIGNATURE |
| 1. | WAP to count number of alphabets, numbers, special characters, spaces, lines in a text file. |  |  |
| 2. | WAP to implement “First Come First Serve” job scheduling algorithm. |  |  |
| 3. | WAP to implement “Shortest Job First” job scheduling algorithm |  |  |
| 4. | WAP to implement “Priority Queue” job scheduling algorithm. |  |  |
| 5. | WAP to implement “Shortest Remaining Time First” job scheduling algorithm |  |  |
| 6. | WAP to implement “Round Robin” job scheduling algorithm |  |  |
| 7. | WAP to implement Banker’s Algorithm for deadlock detection |  |  |
| 8. | WAP to implement “First in First Out” page replacement algorithm |  |  |
| 9. | WAP to implement “Least Recently Used page replacement algorithm |  |  |
| 10. | WAP to implement “Shortest Seek Time First” disk scheduling algorithm |  |  |

**Practical 1**

**Problem Statement:**  WAP to count number of alphabets, numbers, special characters, spaces, lines in a text file.

**Code URL:** https://ide.geeksforgeeks.org/XiFJZ8uleg

**PROGRAM:**

#include<iostream>

using namespace std;

int main()

{

string s;

getline(cin,s);

// this will produce a lot of input, It will count the number of lines?

int countChar=0;

int countNum=0;

int spaces,specialChar;

spaces=specialChar=0;

for(int i=0;i<s.length();i++)

{

char ch=s[i];

if(ch>='0' and ch<='9')

{

countNum++;

}

else if(ch>='a' and ch<='z')

{

countChar++;

}

else if(ch>='A' and ch<='Z')

{

countChar++;

}

else if (ch==' ')

{

specialChar++;

spaces++;

}

else {

specialChar++;

}

}

cout<<"SpecialChar: "<<specialChar<<endl;

cout<<"Spaces: "<<spaces<<endl;

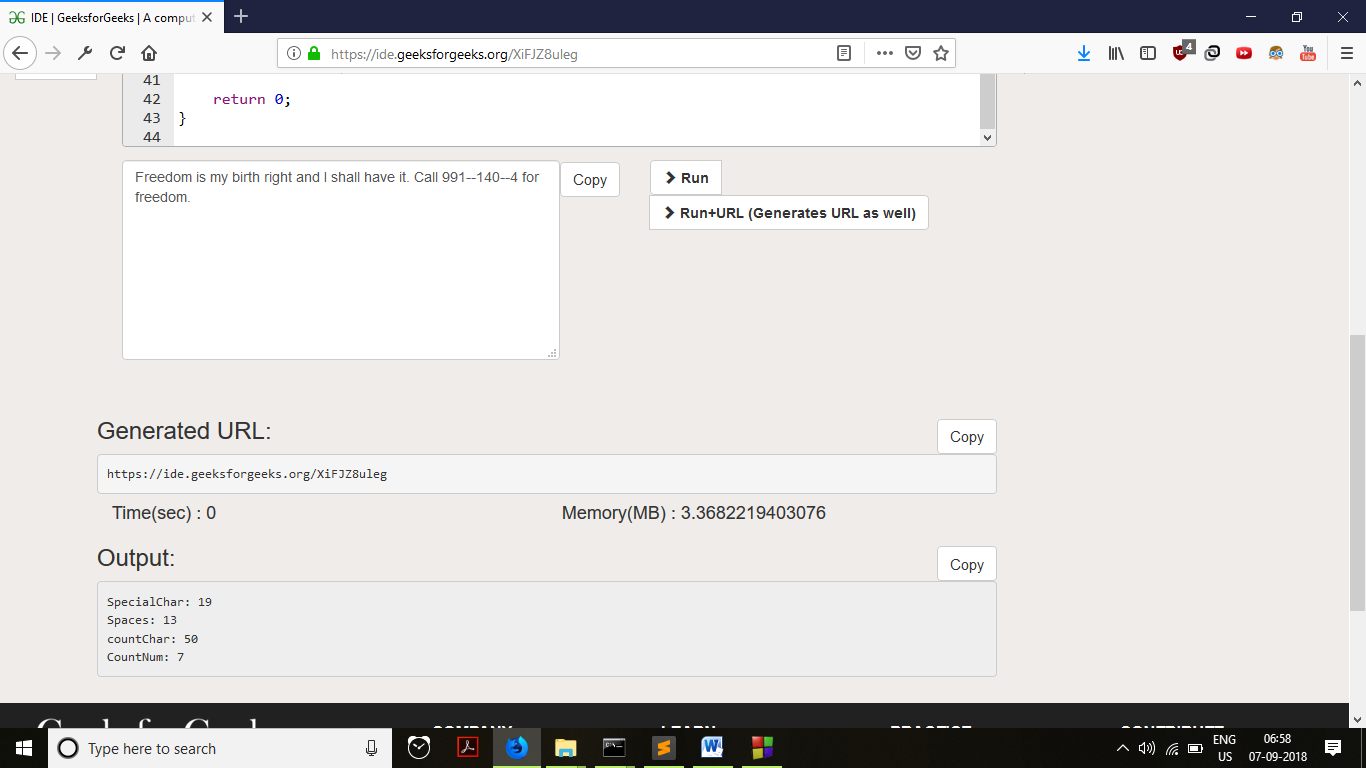
cout<<"countChar: "<<countChar<<endl;

cout<<"CountNum: "<<countNum<<endl;

return 0;

}

**Output:**



**Practical 2**

**Problem Statement:**  WAP to implement “First Come First Serve” job scheduling algorithm.

**Code URL:** https://ide.geeksforgeeks.org/FY55wHDWi7

**PROGRAM:**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

class process{

public:

int processIndex;

float burstTime;

float arrivalTime;

float waitingTime;

float turnAroundTime;

process(float burstTime=0,float arrivalTime=0)

{

this->burstTime=burstTime;

this->arrivalTime=arrivalTime;

turnAroundTime=0;

waitingTime=0;

}

};

bool comparitor(process p1, process p2)

{

return p1.arrivalTime <= p2.arrivalTime;

}

int main()

{

// implementation of first come first serve algorithm

int n;

// cout<<"Enter the number of processes: ";

cin>>n;

process \*p=new process[n];

process \*pDuplicate=new process[n];

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

{

p[i].processIndex=i;

// cout<<"enter burst time for process : "<<i+1<<" : ";

float x,y;

cin>>x;

p[i].burstTime=x;

// cout<<"enter arrival time for process : "<<i+1<<" : ";

cin>>y;

p[i].arrivalTime=y;

pDuplicate[i]=p[i];

}

cout <<setw(10) << "[Process]" << setw(6) << "[Burst Time]" << setw(9) << "[Arrival]"<<endl;

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

{

int index=i;

cout <<setw(10) <<index << setw(6) << p[index].burstTime << setw(9) << p[index].arrivalTime <<endl;

}

// **Algorithm works as follows**

**// first come first serve algorithm**

**// sort the processes on the basis of arrival time**

**// take the process one by one and have a currentTime pointer to denote the processes that have come**

**// complete the process and complete it, set the currentTime pointer to the end of the process.**

**// if the next process [arives] before the current time, then process it and set current time as currentTime+ burstTime, [store the current time and index of the process], it denotes the end of the process,**

**// else also process it, but the currentTime = arrivalTime of next process + burstTime for the next process [ store the current time and the index of the "NEXT" process], since it denotes the end of the next process**

float currentTime=0;

sort(p, p+n, comparitor);

queue<pair<int,float>> q; // index, currentTime

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

{

if(p[i].arrivalTime <= currentTime)

{

currentTime=currentTime+p[i].burstTime;

pair<int,float> Pair(p[i].processIndex,currentTime);

q.push(Pair);

}

else{

currentTime=p[i].arrivalTime+p[i].burstTime;

pair<int,float>Pair(p[i].processIndex,currentTime);

q.push(Pair);

}

}

// table construction sample

cout <<setw(10) << "[Process]" << setw(6) << "[Burst Time]" << setw(9) << "[Arrival]" <<setw(9)<<"[T.A.T]"<<setw(9)<<"[WaitingTime]"<<endl;

while(!q.empty())

{

pair<int,float>Pair=q.front();

q.pop();

int index=Pair.first;

int completitionTime=Pair.second;

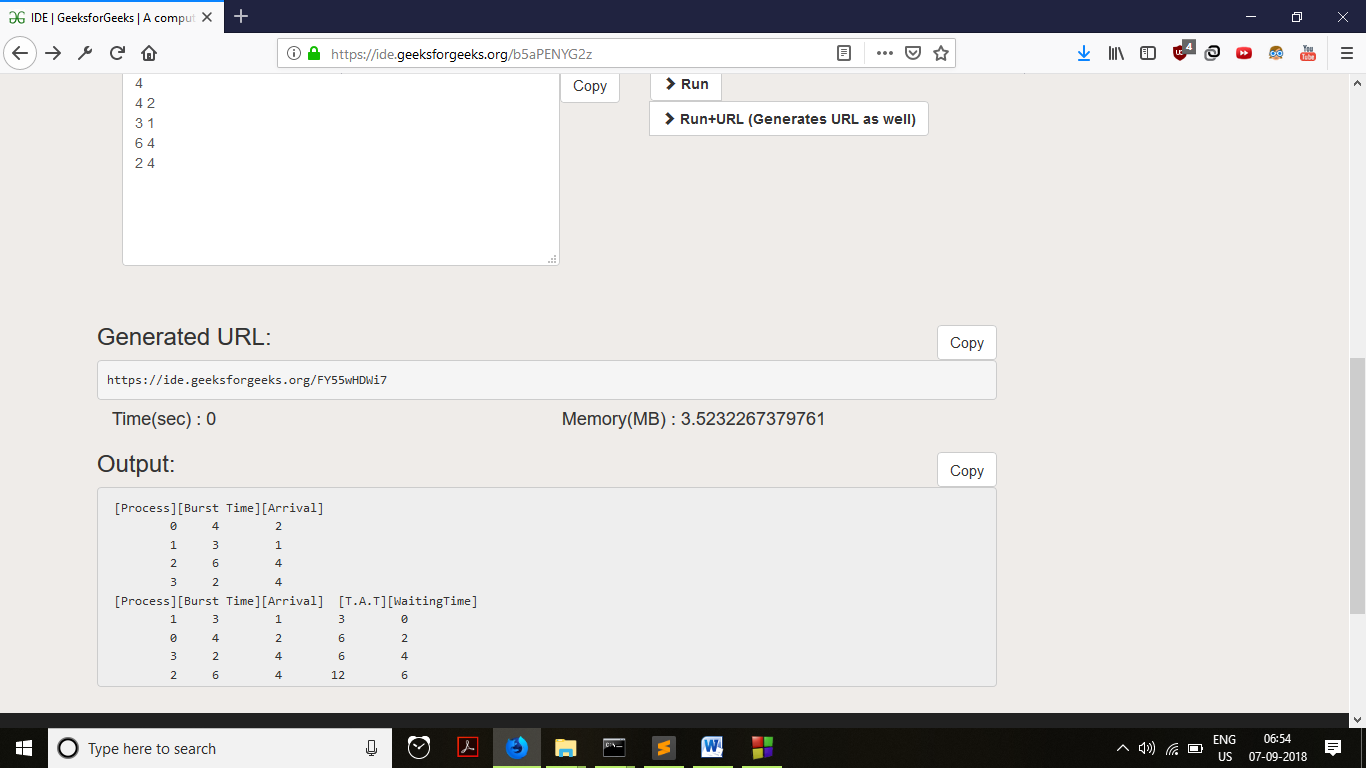
cout <<setw(10) <<index << setw(6) << pDuplicate[index].burstTime << setw(9) << pDuplicate[index].arrivalTime <<setw(9)<<completitionTime-pDuplicate[index].arrivalTime<<setw(9)<<completitionTime-pDuplicate[index].arrivalTime-pDuplicate[index].burstTime<<endl;

}

return 0;

}

**OUTPUT:**



**Practical 3**

**Problem Statement:**  WAP to implement “Shortest Job First” scheduling algorithm.

**Code URL: https://ide.geeksforgeeks.org/KFdZBhAeNd**

**PROGRAM:**

#include<iostream>

#include<bits/stdc++.h> // Works Well :D

using namespace std;

class **process**{

public:

int processIndex;

float burstTime;

float arrivalTime;

float waitingTime;

float turnAroundTime;

bool considered;

process(float burstTime=0,float arrivalTime=0)

{

this->burstTime=burstTime;

this->arrivalTime=arrivalTime;

turnAroundTime=0;

waitingTime=0;

this->considered=false;

}

};

/// WORKS :DDDD

int main()

{

**// implementation of shortest Job First algorithm**

int n;

// cout<<"Enter the number of processes: ";

cin>>n;

process \*p=new process[n];

process \*pDuplicate=new process[n];

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

{

p[i].processIndex=i;

// cout<<"enter burst time for process : "<<i+1<<" : ";

float x,y;

cin>>x;

p[i].burstTime=x;

// cout<<"enter arrival time for process : "<<i+1<<" : ";

cin>>y;

p[i].arrivalTime=y;

pDuplicate[i]=p[i];

}

float avgTAT=0, avgWT=0;

int currentTime=0;

// Output of the processes entered

cout <<setw(10) << "[Process]" << setw(6) << "[Burst Time]" << setw(9) << "[Arrival]"<<endl;

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

{

int index=i;

cout <<setw(10) <<index << setw(6) << p[index].burstTime << setw(9) << p[index].arrivalTime <<endl;

}

**// Start of Algorithm**

int prIndex=0;

int minPrTime=100000;

cout<<endl<<"---- **Shortest Job First Algorithm** ----"<<endl;

cout <<setw(10) << "[Process]" << setw(6) << "[Burst Time]" << setw(9) << "[Arrival]" <<setw(9)<<"[T.A.T]"<<setw(9)<<"[WaitingTime]"<<endl;

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

{

minPrTime=100000;

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

{

// O(n^2) approach

if(!p[i].considered && p[i].arrivalTime <= currentTime)

{

if(p[i].burstTime<minPrTime)

{

minPrTime=p[i].burstTime;

prIndex=i; // this is done

}

}

}

if(minPrTime==100000)

{

currentTime+=1;

i--;

continue;

}

currentTime+=p[prIndex].burstTime;

p[prIndex].considered=true;

cout <<setw(10) <<prIndex << setw(6) << p[prIndex].burstTime << setw(9) << p[prIndex].arrivalTime <<setw(9)<<currentTime-p[prIndex].arrivalTime<<setw(9)<<currentTime-p[prIndex].arrivalTime-p[prIndex].burstTime<<endl;

}

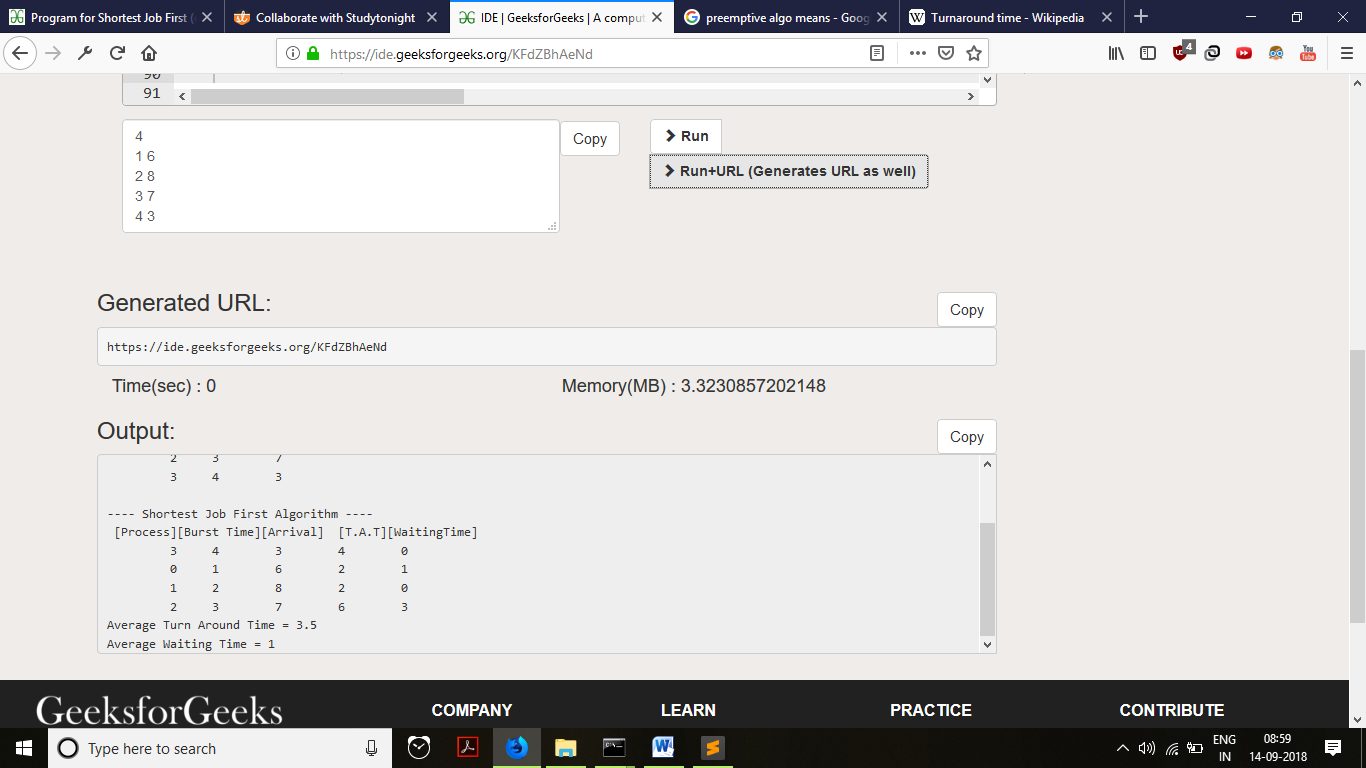
cout<<"Average Turn Around Time = " <<avgTAT/n<<endl;

cout<<"Average Waiting Time = " <<avgWT/n<<endl;

return 0;

}

**Output:**



**Practical 4 or 5**

**Problem Statement:**  WAP to implement “Round Robin” scheduling algorithm.

**Code URL: https://ide.geeksforgeeks.org/ff2IT6c6zO**

**PROGRAM:**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

class process{

public:

int processIndex;

float burstTime;

float arrivalTime;

float waitingTime;

float turnAroundTime;

float tempBurstTime;

process(float burstTime=0,float arrivalTime=0)

{

this->burstTime=burstTime;

this->arrivalTime=arrivalTime;

turnAroundTime=0;

waitingTime=0;

}

};

class pqComparitor

{

public:

bool operator() (process &p1, process&p2) const

{

return p1.burstTime<p2.burstTime;

}

};

bool comparitor(process p1, process p2)

{

return p1.arrivalTime <= p2.arrivalTime;

}

int main()

{

int n;

**// cout<<"Enter the number of processes: ";**

cin>>n;

process \*p=new process[n];

// process \*pCopy=new process[n];

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

{

p[i].processIndex=i;

**// cout<<"enter burst time for process : "<<i+1<<" : ";**

float x,y;

cin>>x;

p[i].burstTime=x;

**// cout<<"enter arrival time for process : "<<i+1<<" : ";**

cin>>y;

p[i].arrivalTime=y;

p[i].tempBurstTime=x;

// pCopy[i].processIndex=i;

// pCopy[i].burstTime=x;

// pCopy[i].arrivalTime=y;

}

int tQuantum;

**cout<<"Enter the time quantum : "<<endl;**

cin>>tQuantum;

int currentTime=0;

int myProcessIndex=0;

sort(p,p+n,comparitor);

queue<process> running;

queue<process> ended;

// the array is an array so...

while(!running.empty() or myProcessIndex<n)

{

while(running.empty())

{

for(int i=myProcessIndex;i<n;i++)

{

if(currentTime<=p[i].arrivalTime)

{

running.push(p[i]); // push the process in running

myProcessIndex+=1;

}

}

if(running.empty()) currentTime+=1; // increment the time till the running has to

}

// pick the process from running and run it for the time quantum

process x=running.front();

running.pop(); // the running array will pop

if(x.burstTime > tQuantum)

{

cout<<"Running process: "<<x.processIndex<<endl;

currentTime+=tQuantum;

x.burstTime-=tQuantum;

running.push(x);

}

else{

currentTime+=x.burstTime;

x.burstTime=0;

**cout<<"Process : "<<x.processIndex<<" ends at "<<currentTime<<endl;**

x.turnAroundTime=currentTime-x.arrivalTime;

x.waitingTime=x.turnAroundTime-x.tempBurstTime;

ended.push(x);

}

for(int i=myProcessIndex;i<=n;i++)

{

if(currentTime<=p[i].arrivalTime)

{

running.push(p[i]); // push the process in running

myProcessIndex+=1;

}

}

}

while(!ended.empty())

{

process x=ended.front();

ended.pop();

**cout<<"Process "<<x.processIndex<<" Turn Around Time is "<<x.turnAroundTime<<" and waiting time is "<<x.waitingTime<<endl;**

}

return 0;

}

**Output:**

