

**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;

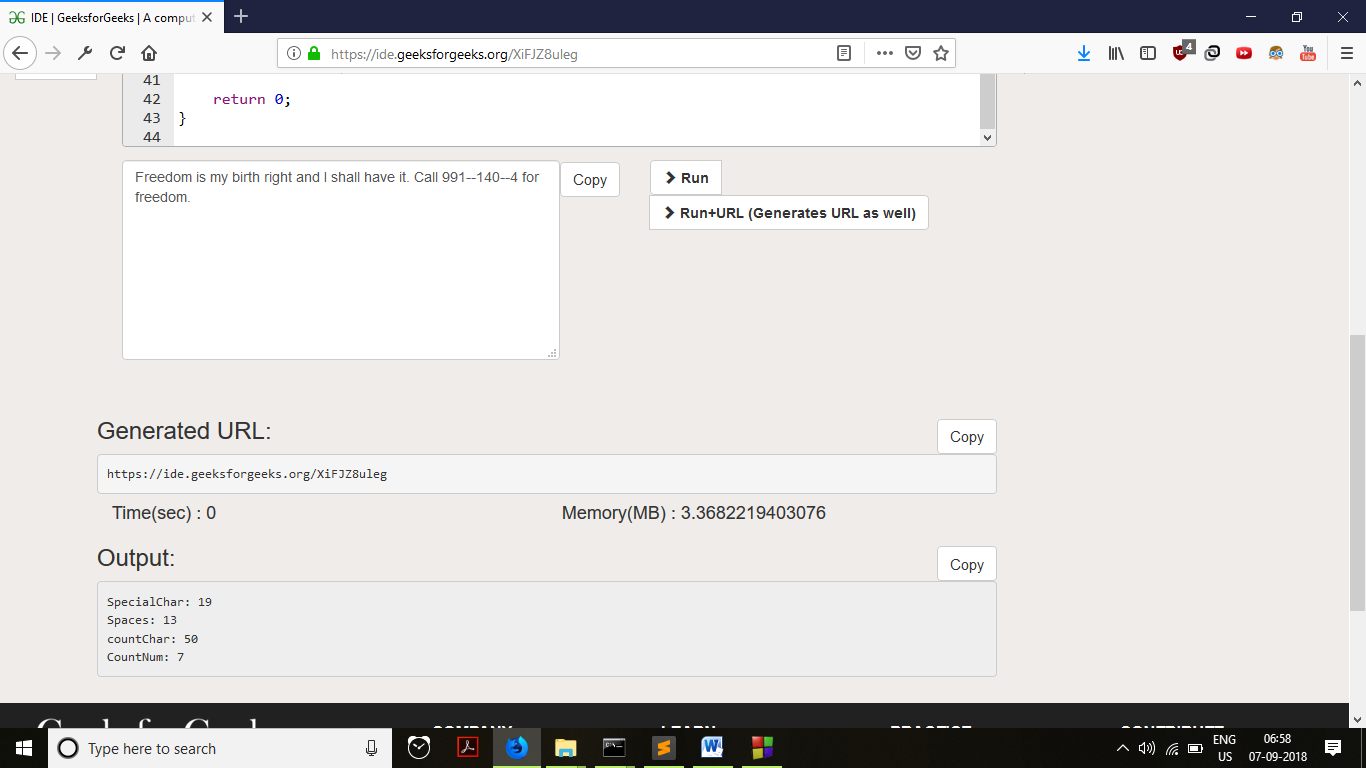
cout<<"countChar: "<<countChar<<endl;

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;

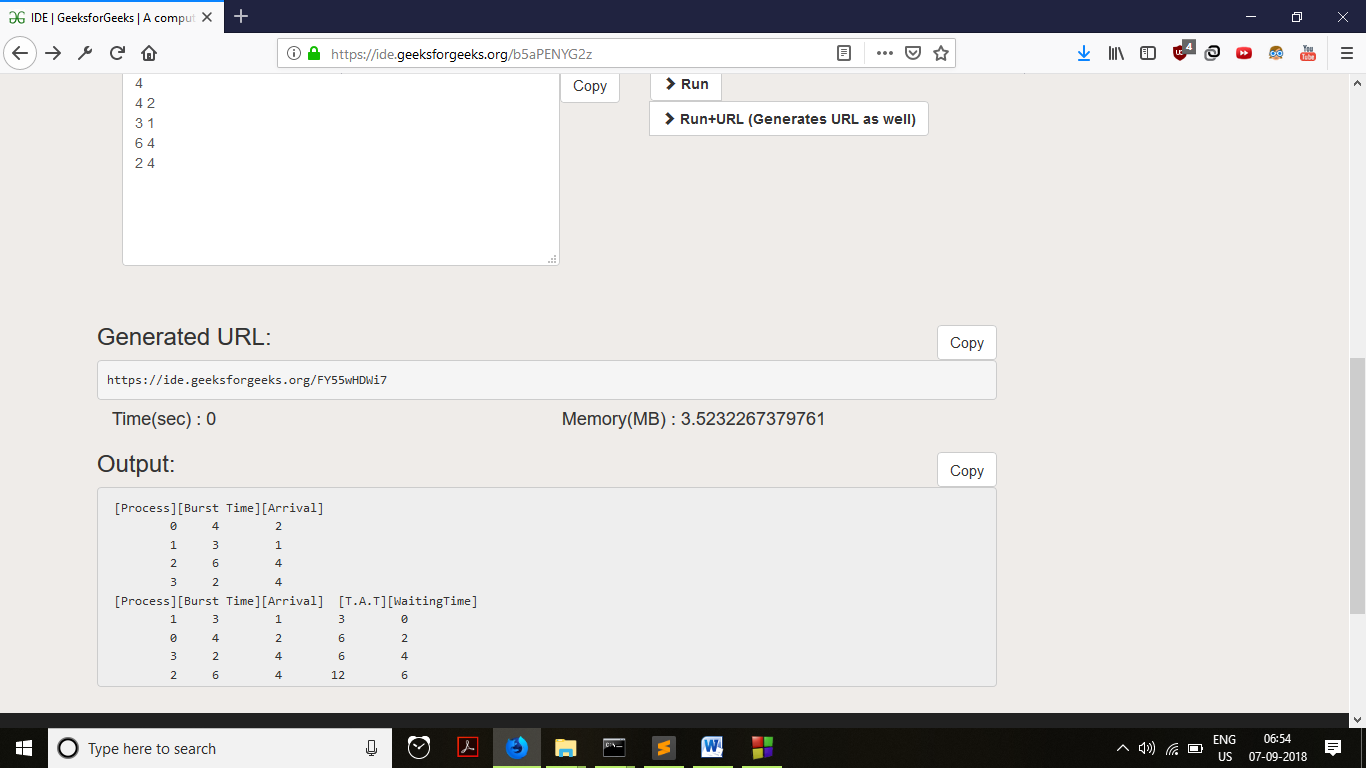
cout <<setw(10) <<index << setw(6) << pDuplicate[index].burstTime << setw(9) << pDuplicate[index].arrivalTime <<setw(9)<<completitionTime-pDuplicate[index].arrivalTime<<setw(9)<<completitionTime-pDuplicate[index].arrivalTime-pDuplicate[index].burstTime<<endl;

}

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;

}

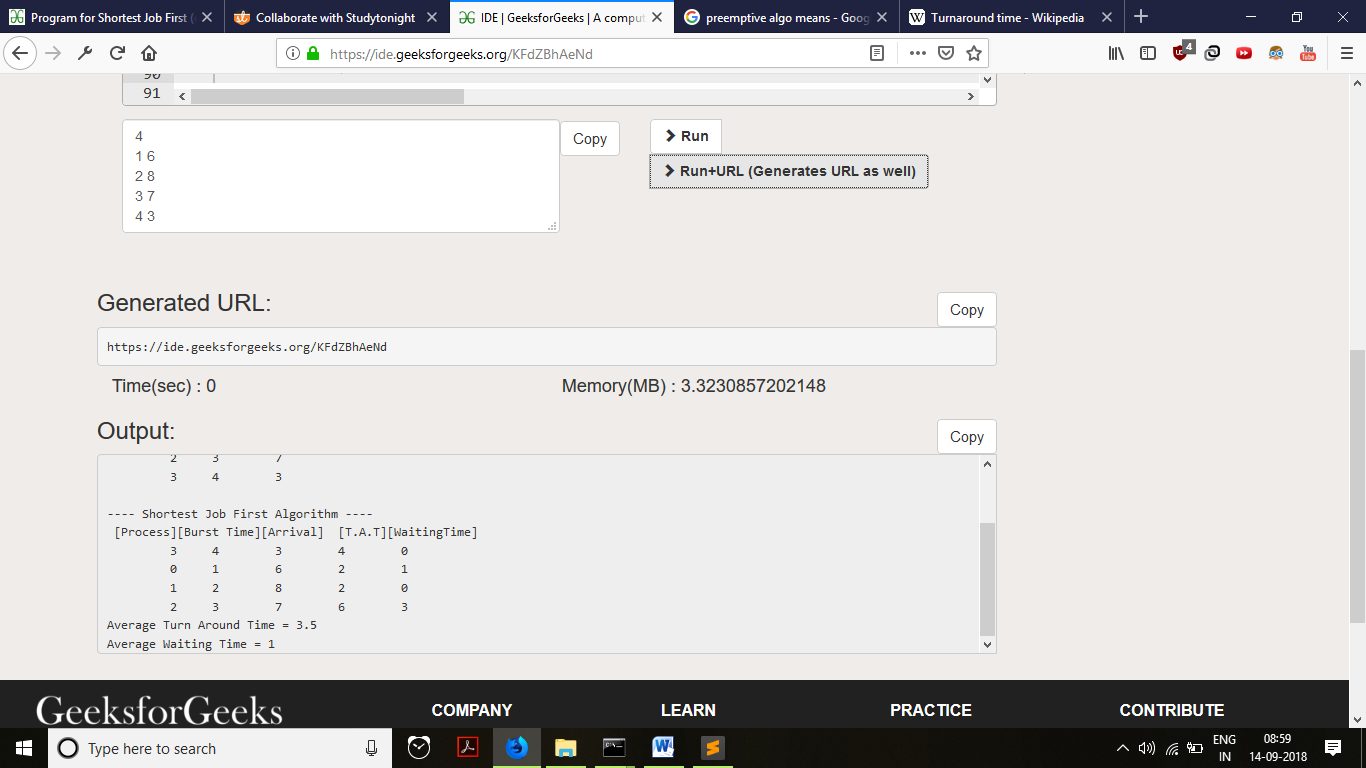
cout<<"Average Turn Around Time = " <<avgTAT/n<<endl;

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

return 0;

}

**Output:**



**Practical 4**

**Problem Statement:**  WAP to implement “Priority Queue” scheduling algorithm.

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

**PROGRAM:**

#include<iostream>

#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;

float priority;

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

{

this->burstTime=burstTime;

this->arrivalTime=arrivalTime;

turnAroundTime=0;

waitingTime=0;

this->priority=1;

}

};

class **priorityQueueComparitor**{

public:

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

return p1.priority<=p2.priority;

}

};

bool **comparitor**(process &p1, process &p2)

{

return p1.arrivalTime <= p2.arrivalTime;

}

/// WORKS :DDDD

int main()

{

// implementation of priority queue scheduling 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,z;

cin>>x;

p[i].burstTime=x;

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

cin>>y;

p[i].arrivalTime=y;

// cout<<"enter the priority for the process: "<<i+1<<" : ";

cin>>z;

p[i].priority=z;

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;

}

**// Priority Queue scheduling algorithm**

**// Sort the process according to start time**

**// increase the start time by 1 in a loop. have a priority queue and process queue. in the priority queue, the process**

**// get added as they come with time. If the processes priority is > priority of running process, then preemt it and run the**

**// higher priority process first**

float currentTime=0;

sort(p, p+n, comparitor);

queue<process> q; // index, currentTime

queue<pair<int,float>> myQ;

priority\_queue<process,vector<process>,priorityQueueComparitor> pq;

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

{

q.push(p[i]);

}

// 3

// 3 1 2

// 3 2 7

// 3 3 5

**while(!q.empty() || !pq.empty())**

{

// cout<<"Here"<<endl;

process x;

if(!q.empty())

{

x = q.front();

if(x.arrivalTime<=currentTime)

{

pq.push(x);

q.pop();

}

}

if(!pq.empty())

{

// do what you were doing

process H=pq.top();

cout<<"processing process: "<<H.processIndex<<endl;

pq.pop();

H.burstTime-=1;

if(H.burstTime==0)

{

pair<int,float> Pair(H.processIndex,currentTime);

cout<<"process finished (1 - index): "<<H.processIndex + 1<<endl;

myQ.push(Pair);

}

else{

pq.push(H);

}

}

currentTime++;

}

**// table construction sample**

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

while(!myQ.empty())

{

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

myQ.pop();

int index=Pair.first;

int completitionTime=Pair.second;

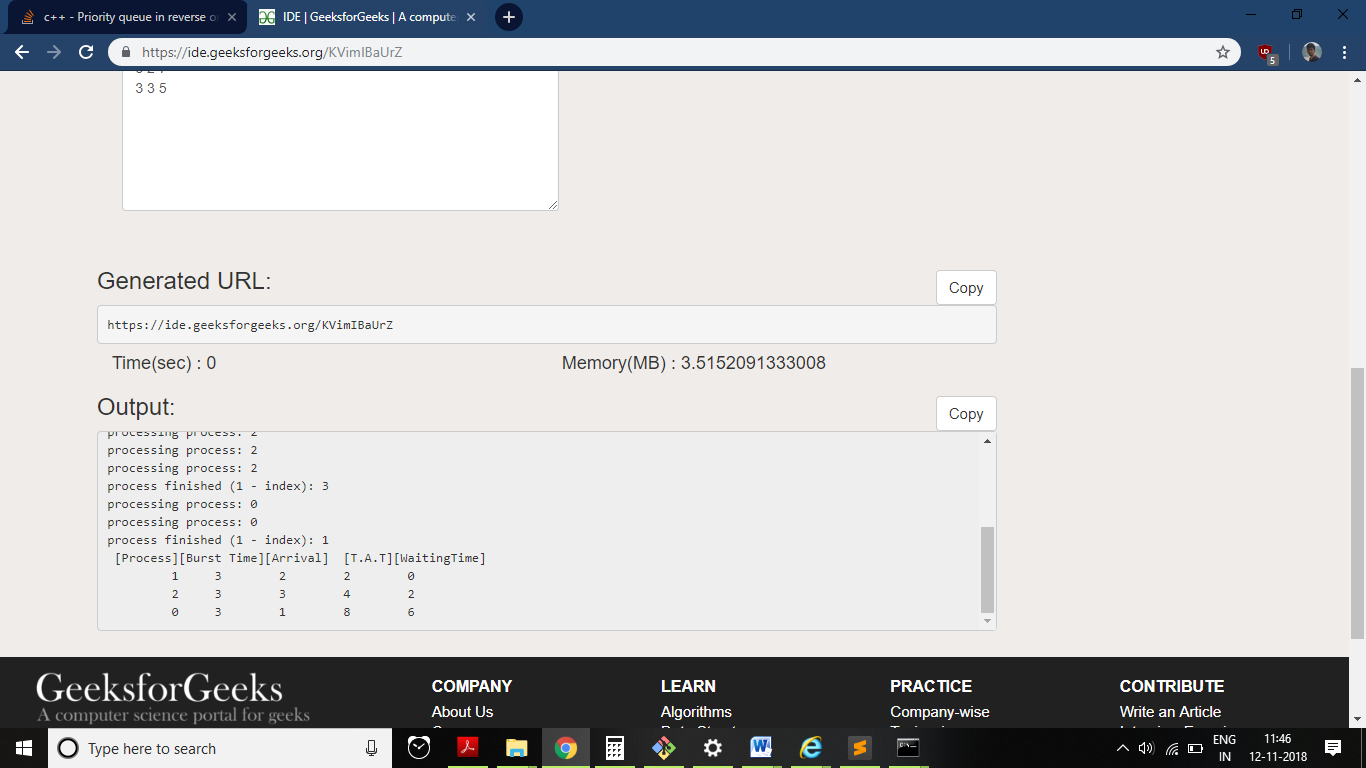
cout <<setw(10) <<index << setw(6) << pDuplicate[index].burstTime << setw(9) << pDuplicate[index].arrivalTime <<setw(9)<<completitionTime-pDuplicate[index].arrivalTime<<setw(9)<<completitionTime-pDuplicate[index].arrivalTime-pDuplicate[index].burstTime<<endl;

}

return 0;

}

**OUTPUT:**



**Practical 5**

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

**Code URL: -- none --**

**PROGRAM:**

#include <bits/stdc++.h>

using namespace std;

struct **Process** {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

void **findWaitingTime**(Process proc[], int n, int wt[])

{

int rt[n];

// Copy the burst time into rt[]

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

// Process until all processes gets

// completed

while (complete != n) {

// Find process with minimum

// remaining time among the

// processes that arrives till the

// current time`

for (int j = 0; j < n; j++) {

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Reduce remaining time by one

rt[shortest]--;

// Update minimum

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

// If a process gets completely

// executed

if (rt[shortest] == 0) {

// Increment complete

complete++;

check = false;

// Find finish time of current

// process

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

void **findTurnAroundTime**(Process proc[], int n, int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = proc[i].bt + wt[i];

}

**// Function to calculate average time**

void **findavgTime**(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0,

total\_tat = 0;

// Function to find waiting time of all

// processes

findWaitingTime(proc, n, wt);

// Function to find turn around time for

// all processes

findTurnAroundTime(proc, n, wt, tat);

**// Display processes along with all**

**// details**

cout << "Processes "

<< " Burst time "

<< " Waiting time "

<< " Turn around time\n";

**// Calculate total waiting time and**

**// total turnaround time**

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << proc[i].pid << "\t\t"

<< proc[i].bt << "\t\t " << wt[i]

<< "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

**int main()**

{

Process proc[] = { { 1, 6, 1 }, { 2, 8, 1 },

{ 3, 7, 2 }, { 4, 3, 3 } };

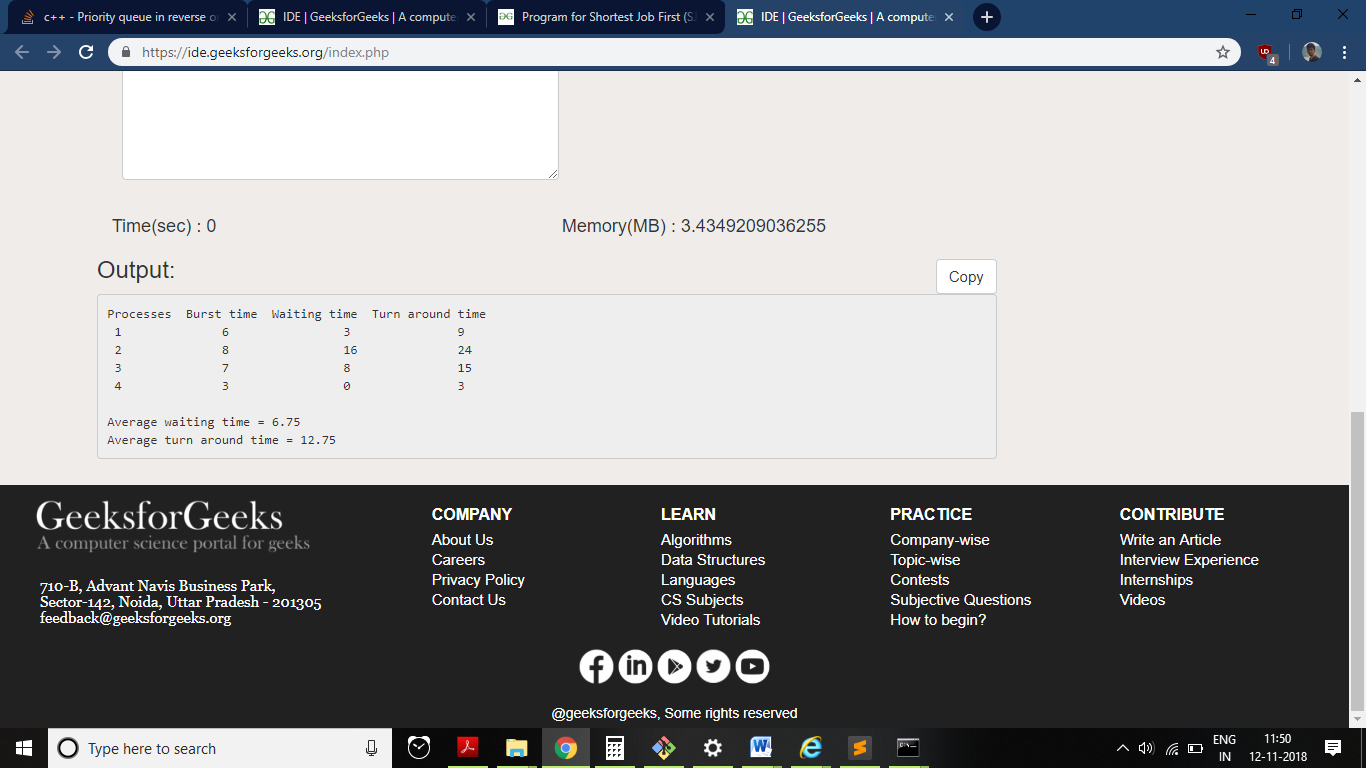
int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

**OUTPUT:**



**Practical 6**

**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();

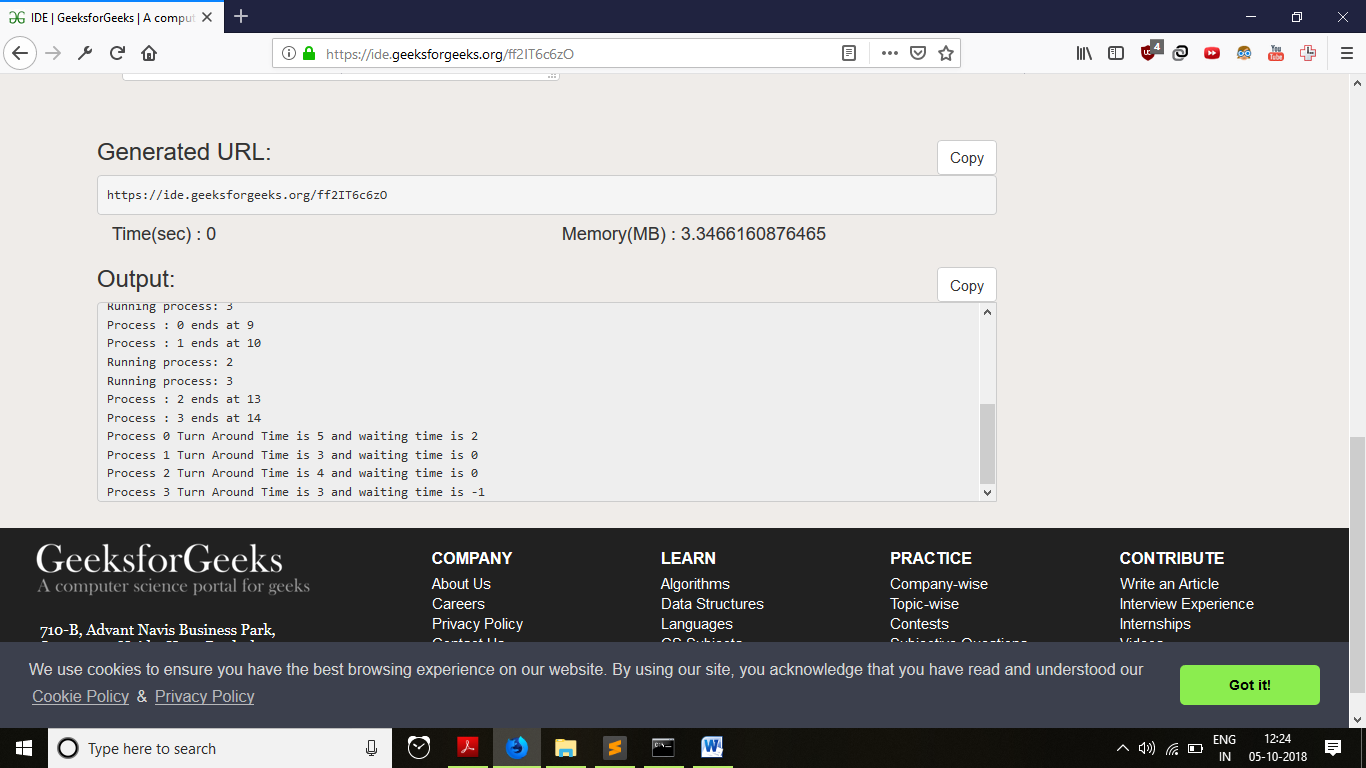
**cout<<"Process "<<x.processIndex<<" Turn Around Time is "<<x.turnAroundTime<<" and waiting time is "<<x.waitingTime<<endl;**

}

return 0;

}

**Output:**



**Practical 7**

**Problem Statement:**  WAP to implement “Banker’s Algorithm” for deadlock detection.

**Code URL: https://ide.geeksforgeeks.org/uDQPw5QrSkPROGRAM:**

#include <iostream>

#include <iomanip>

#include <cstring>

using namespace std;

**#define RESOURCES 3**

**#define PROCESSES 4**

**// The allocation table.**

**int allocation[][RESOURCES] = {**

**{ 2, 1, 3 },**

**{ 1, 2, 3 },**

**{ 5, 4, 3 },**

**{ 2, 1, 2 }**

**};**

**// The maximum table.**

**int maximum[][RESOURCES] = {**

**{ 4, 9, 4 },**

**{ 5, 3, 3 },**

**{ 6, 4, 3 },**

**{ 4, 8, 2 }**

**};**

**// Total resources.**

**int total\_resources[] = { 12, 9, 12 };**

// The finish table.

bool finish[] = { false, false, false, false };

// The array holding the available resources.

int available[RESOURCES];

// The Work array.

int work[RESOURCES];

// The Need array.

int need[PROCESSES][RESOURCES];

/\*\*

\* Format the output.

\*/

template<typename T> void print\_cell(T t, const int& width) {

cout << left << setw(width) << setfill(' ') << t;

}

void **process**(void) {

int i = 0;

int finished\_amount = 0;

int finished\_backup = 0;

bool finished\_one = true;

while(true) {

// Check if the process is not already finished.

if(!finish[i]) {

bool good = true;

// Make sure that there are enough resources for this process:

// Need[i] < Work.

for(int j = 0; j < RESOURCES; j++) {

if(need[i][j] > work[j]) {

good = false;

}

}

// If there are enough resources available to complete this process

// then add the allocated for this process resources back to the

// work array.

if(good) {

**cout << "Able to complete process " << i << " - Work: ";**

for(int j = 0; j < RESOURCES; j++) {

work[j] += allocation[i][j];

print\_cell(work[j], 4);

}

finish[i] = true;

finished\_amount++;

cout << endl;

finished\_one = true;

}

}

i = (i + 1) % PROCESSES;

if((i == 0 && finished\_amount == finished\_backup)) {

**cout << "The system is in an unsafe state" << endl;**

break;

}

else if(finished\_amount == PROCESSES) {

**cout << "The system is in a safe state" << endl;**

break;

}

else if(i == 0) {

finished\_backup = finished\_amount;

}

}

}

void **request**\_**resources**(int \*request, int len, int j) {

bool good = false;

for(int i = 0; i < len; i++) {

if(request[i] > need[j][i]) {

good = false;

break;

}

}

if(good) {

for(int i = 0; i < len; i++) {

if(request[i] > available[i]) {

good = false;

break;

}

}

if(good) {

**cout << "Allocating the resources for process " << j << endl;**

for(int i = 0; i < len; i++) {

available[i] -= request[i];

allocation[j][i] += request[i];

need[j][i] -= request[i];

}

}

else {

**cout << "Process " << j << " must wait. No resources available." << endl;**

}

}

else {

cout << "Process " << j << " can't be given the requested resources ";

cout << "because the request exceeds the need." << endl;

}

}

/\*\*

\* Main entry to the program.

\*/

int main(void) {

**cout << "Calculating the Available resources" << endl;**

memcpy(available, total\_resources, RESOURCES \* sizeof(int)); // shortcut

**cout << "Calculating the Need array." << endl;**

// Fill in the Need array and compute the available resources at the same time.

for(int i = 0; i < PROCESSES; i++) {

for(int j = 0; j < RESOURCES; j++) {

available[j] -= allocation[i][j];

need[i][j] = maximum[i][j] - allocation[i][j];

print\_cell(need[i][j], 4);

}

cout << endl;

}

// Copy the available array into the work array.

memcpy(work, available, RESOURCES \* sizeof(int));

**cout << "Available = Work = ";**

// Just print the available array.

for(int i = 0; i < RESOURCES; i++) {

print\_cell(available[i], 4);

}

cout << endl;

process();

**cout << "Requesting additional resources for Process 1" << endl;**

// Reset the work array.

memcpy(work, available, RESOURCES \* sizeof(int));

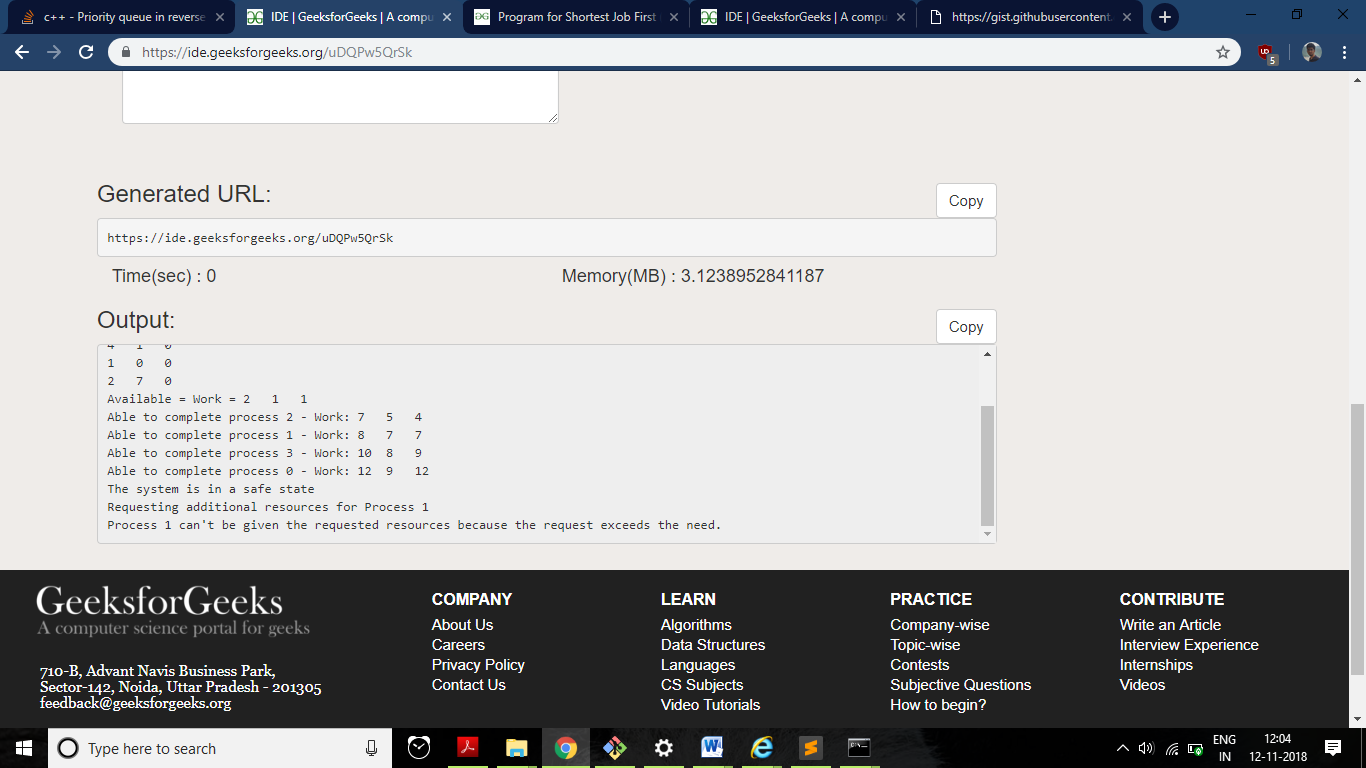
int new\_resources[] = {2, 0, 0};

request\_resources(new\_resources, 3, 1);

return 0;

}

**OUTPUT:**



**Practical 8**

**Problem Statement:**  WAP to implement “FIFO Replacement Algorithm”

**Code URL:**

**PROGRAM:**

#include<iostream>

using namespace std;

class node{

public:

int data;

node\*next;

node(int data=0)

{

this->data=data;

this->next=NULL;

}

};

class PageMemory{

node\*head;

node\*tail;

int pageSize;

int hitCount;

// int pageCount;

int currSize;

public:

PageMemory(int pageSize=3)

{

this->head=this->tail=NULL;

this->pageSize=pageSize;

currSize=hitCount=0;

}

bool search(int x)

{

node\*temp=head;

while(temp!=NULL)

{

if(temp->data==x)

{

return true;

}

temp=temp->next;

}

return false;

}

void insert(int x)

{

if(search(x))

{

return;

}

else{

hitCount++;

node\* n=new node(x);

cout<<"here"<<endl;

if(currSize<pageSize)

{

currSize++;

if(!head)

{

head=tail=n;

}

else{

tail->next=n;

tail=tail->next;

}

return;

}

else{

head=head->next;// Delete permanently from heap memory should be done

tail->next=n;

tail=tail->next;

return;

}

}

cout<<"here too"<<endl;

}

int getHitCount()

{

return this->hitCount;

}

void printPageMemory()

{

node\*temp=head;

while(temp)

{

cout<<temp->data<<"->";

temp=temp->next;

}

}

};

int main()

{

int pageSize=3;

int arr[]={1,2,3,1,2,5,6};

PageMemory pm(pageSize);

for(int i=0;i<7;i++)

{

pm.insert(arr[i]);

// cout<<"Hit count after inserting "<<arr[i]<<" is "<<pm.getHitCount()<<endl;

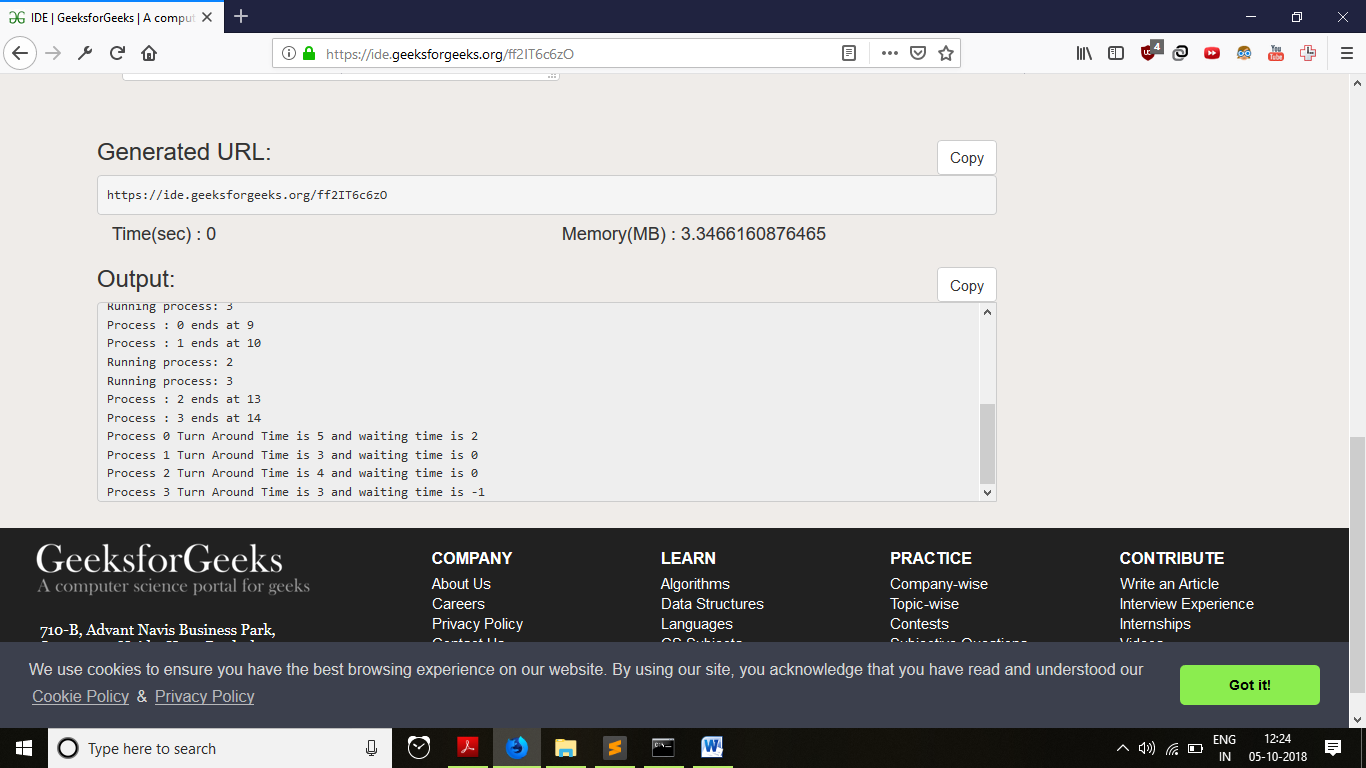
}

cout<<"Hit count is : "<<pm.hitCount<<endl;

return 0;

}

**Output:**



**Practical 9**

**Problem Statement:**  WAP to implement “LRU Replacement Algorithm”

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

**PROGRAM:**

#include<iostream>

using namespace std;

**#include<list>**

// **I am solving this problem using list stl**

**// advanced delete to do work of 2 functions at a time**

**// This function checks whether key is there and deletes the key if present**

**bool deleteKey(list<int> &myList,int key)**

{

auto it=myList.begin();

bool flag=false;

for(it=myList.begin();it!=myList.end();it++)

{

if(\*it == key)

{

flag=true;

break;

}

}

if(flag)

{

myList.erase(it);

return true;

}

return false; // not found key

}

**void printCache(list<int>& l)**

{

for(auto it=l.begin();it!=l.end();it++)

{

cout<<\*it<<" ";

}

}

**int main()**

{

int n;

**cout<<"Enter the number of pages coming: ";**

cin>>n;

**cout<<"Enter incoming pages: ";**

int \*incommingPages=new int[n];

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

{

cin>>incommingPages[i];

}

int pageMemorySize;

**cout<<"Enter the size of Cache: ";**

cin>>pageMemorySize;

**// initialize the page memory as blank**

list<int> pageCache;

int hitCount=0;

**// so the page cache is empty**

**// Main code starts**

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

{

int key=incommingPages[i];

// for each key do this :

**// if cache is empty do this :**

if(pageCache.size()<pageMemorySize)

{

if(deleteKey(pageCache,key))

{

pageCache.push\_front(key); **// no hit count, so bring the key forward**

else{

pageCache.push\_front(key);

hitCount++;

cout<<"Miss!"<<endl;

}

}

**// if cache is not empty**

else{

if(deleteKey(pageCache,key))

{

pageCache.push\_front(key);

}

else{

pageCache.push\_front(key);

pageCache.pop\_back();

hitCount++;

**cout<<"Miss!"<<endl;**

}

}

**cout<<"The new cache is : ";**

**printCache(pageCache);**

**cout<<endl;**

}

cout<<"The total miss count is : "<<hitCount<<endl;

return 0;

}

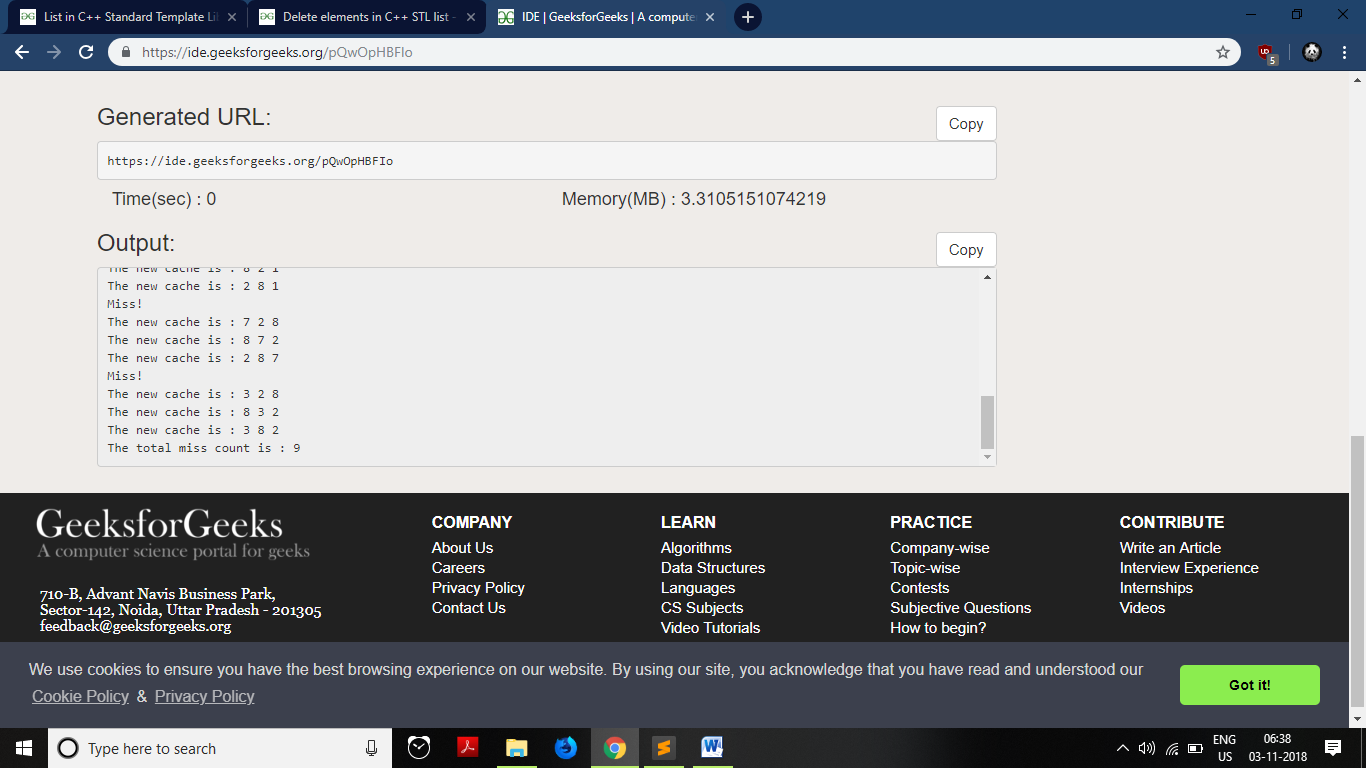
**Input:**

20

0 9 0 1 8 1 8 7 8 7 1 2 8 2 7 8 2 3 8 3

3

**Output:**



Enter the number of pages coming:

Enter incoming pages:

Enter the size of Cache:

Miss!

The new cache is : 0

Miss!

The new cache is : 9 0

The new cache is : 0 9

Miss!

The new cache is : 1 0 9

Miss!

The new cache is : 8 1 0

The new cache is : 1 8 0

The new cache is : 8 1 0

Miss!

The new cache is : 7 8 1

The new cache is : 8 7 1

The new cache is : 7 8 1

The new cache is : 1 7 8

Miss!

The new cache is : 2 1 7

Miss!

The new cache is : 8 2 1

The new cache is : 2 8 1

Miss!

The new cache is : 7 2 8

The new cache is : 8 7 2

The new cache is : 2 8 7

Miss!

The new cache is : 3 2 8

The new cache is : 8 3 2

The new cache is : 3 8 2

The total miss count is : 9

**Practical 10**

**Problem Statement:**  WAP to implement “Shortest Seek Time First Disk Scheduling” algorithm.

**Code URL:**

**PROGRAM:**

**#include<iostream>**

using namespace std;

int main()

{

int n;

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

cin>>n;

int \*arr=new int[n];

int \*selected=new int[n];

cout<<"Enter the processes position on disk : ";

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

{

cin>>arr[i];

selected[i]=0; //selected

}

int k;

cout<<"enter initial diskHead position : ";

cin>>k;

int diskHead=k;

int travelDistance=0;

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

{

// for each of the n processes;

int minimum=100000;

int minIndex=0;

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

{

if(selected[j]==0)

{

int diff=abs(diskHead-arr[j]);

if(diff<minimum)

{

minimum=diff;

minIndex=j;

cout<<diff<<endl;

}

}

}

selected[minIndex]=1; // that selected is true;

travelDistance+=abs(diskHead-arr[minIndex]);

cout<<"Selected process : "<<minIndex+1<<endl;

diskHead=arr[minIndex];

}

cout<<"The sum of distance traveled by head is : "<<travelDistance<<endl;

return 0;

}

**Output:**

