**OS PRACTICAL**

|  |  |  |
| --- | --- | --- |
| 1. | a. | Write a Shell Program to find the factorial of a given n number.  **Program:**  echo Code to find Factorial of n numbers  echo Enter the number :  read n  fact=1  for((i=1 ; i<=n ; i++))  do  fact=$(expr $fact \\* $i)  done  echo factorial of $n numbers is $fact |
|  | b. | Write a C/C++ Program to simulate the copy, edit and rename command. |
| 2. | a. | Write a Shell Program to find the Fibonacci of n numbers.  **Program:**  echo Code to find Fibonacci of n numbers  echo Enter the number :  read n  f1=0  f2=1  echo Fibonacci of $n numbers :  echo $f1  echo $f2  for((i=2 ; i<n ; i++))  do  k=$(expr $f1 \+ $f2)  f1=$f2  f2=$k  echo $k  done |
|  | b. | Write a C/C++ program to simulate FCFS/FIFO CPU scheduling algorithm.  **Program:**  #include<iostream>  using namespace std;  int main() {  cout<<"Enter number of Process : ";  int n;  cin>>n;  cout<<"Enter the process\n";  int process[n];  int burst\_time[n];  for(int i=0 ; i<n ; i++) {  cout<<"Process P"<<(i+1)<<": ";  cin>>process[i];  cout<<"Burst Time : ";  cin>>burst\_time[i];  }  int turn\_around\_time[n], wait\_time[n];  turn\_around\_time[0] = burst\_time[0];  for(int i=0 ; i<n ; i++) {  turn\_around\_time[i] = turn\_around\_time[i-1]+burst\_time[i];  }  for(int i=0 ; i<n ; i++) {  wait\_time[i] = turn\_around\_time[i]-burst\_time[i];  }  int tot\_wt=0,tot\_tat=0;  for(int i=0 ; i<n ; i++) {  tot\_tat += turn\_around\_time[i];  tot\_wt += wait\_time[i];  }  cout<<"Process\tBurst Time\tTurn Around Time\tWait Time\n";  for(int i=0 ; i<n ; i++) {  cout<<process[i]<<"\t\t"<<burst\_time[i]<<"\t\t"<<turn\_around\_time[i]<<"\t\t"<<wait\_time[i]<<"\n";  }  cout<<"Average TAT = "<<(float)tot\_tat/n;  cout<<"\nAverage WT = "<<(float)tot\_wt/n;  } |
| 3. | a. | Write a shell program to perform file operations. |
|  | b. | Write a C/C++ program to simulate FCFS/FIFO Disk scheduling algorithm.  **Program:**  #include<bits/stdc++.h>  using namespace std;  int main() {  int n;  cout<<"Enter the size of the queue : ";  cin>>n;  cout<<"Enter the disk queue : ";  int disk[n];  for(int i=0 ; i<n ; i++) {  cin>>disk[i];  }  cout<<"Enter the initial head position : ";  int head;  cin>>head;  int tot\_head\_time = 0, no\_of\_head\_movements = 0;  for(int i=0 ; i<n ; i++) {  tot\_head\_time += abs(head-disk[i]);  head = disk[i];  no\_of\_head\_movements++;  }  cout<<"Toal seek time : "<<tot\_head\_time;  cout<<"\nNumber of head movements : "<<no\_of\_head\_movements;  } |
| 4. | a. | Write a Shell Program to find the sum of n numbers  **Program:**  echo Code to find the Sum of n numbers  echo Enter the number :  read n  sum=0  for((i=1 ; i<=n ; i++))  do  sum=$(expr $sum \+ $i)  done  echo Sum of $n numbers is $sum |
|  | b. | Write a C/C++ program to simulate producer consumer problem. |
| 5. | a. | Write a shell program to find the greatest of three numbers.  **Program:**  echo Code to find the greatest of 3 numbers  echo Enter the numbers  read a b c  if [ $a -ge $b ] && [ $a -ge $c ]  then  echo The greatest of given 3 numbers is a : $a  elif [ $b -gt $c ]  then  echo The greatest of given 3 numbers is b : $b  else  echo The greatest of given 3 numbers is c : $c  fi |
|  | b. | Write a C/C++ program to simulate FCFS page replacement algorithm.  **Program:**  //fifo PAGE REPLACEMENT  #include<bits/stdc++.h>  using namespace std;  int main()  {  //No of inputs in sequence  int n;  cin>>n;  //The Sequence  int arr[n];  for(int i=0; i<n; i++)  {  cin>>arr[i];  }  //The number of frames  int m;  cin>>m;  map<int,int> mpp;  for(int i=1; i<=m; i++)  mpp[i] = -1;  queue<int> q;  set<int> st;  int cnt = 0;  for(int i=0; i<n; i++)  {  if(st.find(arr[i])!=st.end())  {  for(int frame=1; frame<=m; frame++)  cout<<mpp[frame]<<" ";  }else{  if(q.size() == m)  {  int numberToBeRemoved = q.front();  st.erase(q.front());  q.pop();  int ind = 0;  for(int frame=1; frame<=m; frame++)  {  if(mpp[frame] == numberToBeRemoved)  {  ind = frame;  break;  }  }  mpp[ind] = arr[i];  q.push(arr[i]);  st.insert(arr[i]);  }else{  q.push(arr[i]);  st.insert(arr[i]);  mpp[q.size()] = arr[i];  }  cnt++;  for(int frame = 1;frame<=m; frame++)  {  cout<<mpp[frame]<<" ";  }  }  cout<<endl;  }  cout<<"Page Faults : "<<cnt<<endl;  } |
| 6. | a. | Write a C program to check whether a given file is in a directory or not. |
|  | b. | Write a C/C++ program to simulate SJF CPU scheduling algorithm.  **Program:**  #include<bits/stdc++.h>  using namespace std;  int main()  {  cout<<"SJF CPU Scheduling\n";  cout<<"Enter the number of Process : ";  int n;  cin>>n;  pair<int,int> p[n];  for(int i=0; i<n; i++)  {  cout<<"Burst time of Process P"<<(i+1)<<" : ";  cin>>p[i].first;  p[i].second = i;  }  int wait[n],tat[n];  sort(p,p+n);  int crntTime = 0;  for(int i=0; i<n; i++)  {  int ind = p[i].second;  wait[ind] = crntTime;  crntTime += p[i].first;  tat[ind] = crntTime;  }  cout<<"process waitTime\ttat\n";  double tot\_wt = 0,tot\_tat = 0;  for(int i=0; i<n; i++)  {  cout<<i + 1<<"\t\t"<<wait[i]<<"\t\t"<<tat[i]<<endl;  tot\_wt += wait[i];  tot\_tat += tat[i];  }  cout<<"avg Wait Time\tavg Tat\n";  cout<<tot\_wt/(double)n<<"\t\t"<<tot\_tat/(double)n<<endl;  } |
| 7. | a. | Write a C program simulate process system calls. |
|  | b. | Write a C/C++ program to check whether a given system is safe or not. |
| 8. | a. | Write a shell program to find the sum of n numbers.  **Program:**  echo Code to find the Sum of n numbers  echo Enter the number :  read n  sum=0  for((i=1 ; i<=n ; i++))  do  sum=$(expr $sum \+ $i)  done  echo Sum of $n numbers is $sum |
|  | b. | Write a C/C++ program to simulate priority CPU scheduling algorithms.  **Program:**  #include<bits/stdc++.h>  using namespace std;  bool comp(pair<int,pair<int,int>> &a,pair<int,pair<int,int>> &b)  {  if(a.first == b.first)  {  return a.second.second< b.second.second;  }  return a.first<b.first;  }  int main()  {  cout<<"Enter the number of Process : ";  int n;  cin>>n;  pair<int,pair<int,int>> p[n];  for(int i=0; i<n; i++)  {  cout<<"Priority of Process P"<<(i+1)<<": ";  cin>>p[i].first;  cout<<"Burst time of Process P"<<(i+1)<<" : ";  cin>>p[i].second.first;  p[i].second.second = i;  }  int wait[n],tat[n];  sort(p,p+n,comp);  int crntTime = 0;  for(int i=0; i<n; i++)  {  int ind = p[i].second.second;  wait[ind] = crntTime;  crntTime += p[i].second.first;  tat[ind] = crntTime;  }  cout<<"process waitTime\ttat\n";  double tot\_wt = 0,tot\_tat = 0;  for(int i=0; i<n; i++)  {  cout<<i + 1<<"\t\t"<<wait[i]<<"\t\t"<<tat[i]<<endl;  tot\_wt += wait[i];  tot\_tat += tat[i];  }  cout<<"avg Wait Time\tavg Tat\n";  cout<<tot\_wt/(double)n<<"\t\t"<<tot\_tat/(double)n<<endl;  } |
| 9. | a. | Write a menu driven Shell Programming to perform the following  i)To check whether a given year is leap year or not.  ii)To find whether a given number is positive or negative or neither.  **Program:**  echo Menu driven Shell Program  echo Choice 1 : To find the given year is leap year or not  echo Choice 2 : To find given number is positive, negative or neither  echo Enter your Choice  read choice  case $choice in  1)echo Enter the year :  read year  b=$(expr $year \% 4)  if [ $b -eq 0 ]  then  echo $year is a Leap year  else  echo $year is not a Leap year  fi  ;;  2)echo Enter the number :  read n  if [ $n -gt 0 ]  then  echo The number $n is positive  elif [ $n -lt 0 ]  then  echo The number $n is negative  else  echo The number $n is neither positive nor negative  fi  ;;  \*)echo Enter the correct choice  ;;  esac |
|  | b. | Write a C/C++ program to perform  i) stat system calls  ii) FIFO Disk scheduling algorithm.  **Program:**  #include<bits/stdc++.h>  using namespace std;  int main() {  int n;  cout<<"Enter the size of the queue : ";  cin>>n;  cout<<"Enter the disk queue : ";  int disk[n];  for(int i=0 ; i<n ; i++) {  cin>>disk[i];  }  cout<<"Enter the initial head position : ";  int head;  cin>>head;  int tot\_head\_time = 0, no\_of\_head\_movements = 0;  for(int i=0 ; i<n ; i++) {  tot\_head\_time += abs(head-disk[i]);  head = disk[i];  no\_of\_head\_movements++;  }  cout<<"Toal seek time : "<<tot\_head\_time;  cout<<"\nNumber of head movements : "<<no\_of\_head\_movements;  } |
| 10. | a. | Write a shell program to find whether the given number is odd or even.  **Program:**  echo Program to find whether the given number is odd or even  echo Enter the number :  read n  rem=$(expr $n \% 2)  if [ $rem -eq 0 ]  then  echo The number $n is even  else  echo The number $n is odd  fi |
|  | b. | Write c/c++ program  i)to simulate FIFO page replacement algorithm.  **Program:**  //fifo PAGE REPLACEMENT  #include<bits/stdc++.h>  using namespace std;  int main()  {  //No of inputs in sequence  int n;  cin>>n;  //The Sequence  int arr[n];  for(int i=0; i<n; i++)  {  cin>>arr[i];  }  //The number of frames  int m;  cin>>m;  map<int,int> mpp;  for(int i=1; i<=m; i++)  mpp[i] = -1;  queue<int> q;  set<int> st;  int cnt = 0;  for(int i=0; i<n; i++)  {  if(st.find(arr[i])!=st.end())  {  for(int frame=1; frame<=m; frame++)  cout<<mpp[frame]<<" ";  }else{  if(q.size() == m)  {  int numberToBeRemoved = q.front();  st.erase(q.front());  q.pop();  int ind = 0;  for(int frame=1; frame<=m; frame++)  {  if(mpp[frame] == numberToBeRemoved)  {  ind = frame;  break;  }  }  mpp[ind] = arr[i];  q.push(arr[i]);  st.insert(arr[i]);  }else{  q.push(arr[i]);  st.insert(arr[i]);  mpp[q.size()] = arr[i];  }  cnt++;  for(int frame = 1;frame<=m; frame++)  {  cout<<mpp[frame]<<" ";  }  }  cout<<endl;  }  cout<<"Page Faults : "<<cnt<<endl;  }  ii)perform operations on fork() and exec() system calls. |

**#Factorial of a number**

**echo "Enter the number"**

**read number**

**fact=1**

**while [ $number -gt 1 ]**

**do**

**fact=$((fact\*number))**

**number=$((number-1))**

**done**

**echo $fact**

**read n;**

**fact=1;**

**for((i=1;i<=n;i++))**

**do**

**fact=$((fact\*i))**

**done**

**echo The factorial is $fact;**

**read n;**

**fact=0;**

**for((i=1;i<=n;i++))**

**do**

**fact=$((fact+i))**

**done**

**echo The sum of $n is $fact;**

**read n**

**a=$((n%4))**

**if [ $a -eq 0 ]**

**then**

**echo is leap year**

**else**

**echo is not a leap year;**

**fi;**

**read a b c;**

**if [ $a -gt $b -a $a -gt $c ]**

**then**

**echo "the $a is greatest"**

**elif [ $b -gt $c -a $b -gt $a ]**

**then**

**echo "the $b is greatest";**

**else**

**echo "the $c is greatest";**

**fi;**

**read a ;**

**if [ $a -gt 0 ]**

**then**

**echo is positive**

**else**

**echo is negative**

**fi**

**read n;**

**a=0**

**b=1**

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

**do**

**echo $a**

**f=$((a+b))**

**a=$b**

**b=$f**

**done**

**#2.Fibbonacci of n numbers**

**echo "Enter the number"**

**read num**

**a=0**

**b=1**

**echo "$a "**

**echo "$b "**

**for((i=2;i<num;i++))**

**{**

**sum=$((a+b))**

**echo "$sum "**

**a=$b**

**b=$sum**

**}**

**#4.Sum of n numbers**

**echo "Enter the number"**

**read num**

**sum=0**

**for((i=1;i<=num;i++))**

**do**

**sum=$((sum+i))**

**done**

**echo $sum**

**#5.Greatest of 3 numbers**

**echo "Enter the number1"**

**read num1**

**echo "Enter the number2"**

**read num2**

**echo "Enter the number3"**

**read num3**

**if [ $num1 -ge $num2 ] && [ $num1 -ge $num3 ]**

**then**

**echo "$num1 is greatest"**

**elif [ $num2 -ge $num1 ] && [ $num2 -ge $num3 ]**

**then**

**echo "$num2 is greatest"**

**else**

**echo "$num3 is greatest"**

**fi**

**9.1)Leap year**

**read year**

**if(($year%4==0 && $year%100!=0 || $year%400==0))**

**then echo The year is leap year**

**else**

**echo The year is not leap year**

**fi**

**#9.2)Positive negative or neither**

**echo "Enter the number"**

**read num**

**if [ $num -gt 0 ]**

**then**

**echo "$num is Positive"**

**elif [ $num -lt 0 ]**

**then**

**echo "$num is Negative"**

**else**

**echo "$num is neither Positive nor negative"**

**fi**

**read a**

**read b**

**read c**

**if(($a>=$b && $a>=$c))**

**then echo $a**

**elif(($b>=$a && $b>=$c))**

**then echo $b**

**else**

**echo $c**

**fi**

**#10.Odd or Even**

**echo "Enter the number"**

**read num**

**if [ $(($num%2)) -eq 0 ]**

**then**

**echo "$num is Even"**

**else**

**echo "$num is Odd"**

**fi**

**#3**

**while true; do**

**echo "Choose an option:"**

**echo "1. Perform cat command"**

**echo "2. Perform cp command"**

**echo "3. Perform mv command"**

**echo "4. Perform wc command"**

**echo "5. Perform grep command"**

**echo "6. Perform head command"**

**echo "7. Perform tail command"**

**echo "8. Perform sort command"**

**echo "9. Exit"**

**read -p "Enter your choice: " choice**

**case $choice in**

**1)**

**read -p "Enter the filename for cat command: " filename**

**if [ -f "$filename" ]; then**

**cat "$filename"**

**else**

**echo "File not found: $filename"**

**fi**

**;;**

**2)**

**read -p "Enter the source file for cp command: " srcfile**

**read -p "Enter the destination file for cp command: " destfile**

**if [ -f "$srcfile" ]; then**

**cp "$srcfile" "$destfile"**

**echo "File copied to $destfile"**

**else**

**echo "Source file not found: $srcfile"**

**fi**

**;;**

**3)**

**read -p "Enter the source file for mv command: " srcfile**

**read -p "Enter the destination file for mv command: " destfile**

**if [ -f "$srcfile" ]; then**

**mv "$srcfile" "$destfile"**

**echo "File moved to $destfile"**

**else**

**echo "Source file not found: $srcfile"**

**fi**

**;;**

**4)**

**read -p "Enter the filename for wc command: " filename**

**if [ -f "$filename" ]; then**

**wc "$filename"**

**else**

**echo "File not found: $filename"**

**fi**

**;;**

**5)**

**read -p "Enter the pattern to search for with grep command: " pattern**

**read -p "Enter the filename for grep command: " filename**

**if [ -f "$filename" ]; then**

**grep "$pattern" "$filename"**

**else**

**echo "File not found: $filename"**

**fi**

**;;**

**6)**

**read -p "Enter the filename for head command: " filename**

**if [ -f "$filename" ]; then**

**head "$filename"**

**else**

**echo "File not found: $filename"**

**fi**

**;;**

**7)**

**read -p "Enter the filename for tail command: " filename**

**if [ -f "$filename" ]; then**

**tail "$filename"**

**else**

**echo "File not found: $filename"**

**fi**

**;;**

**8)**

**read -p "Enter the filename for sort command: " filename**

**if [ -f "$filename" ]; then**

**sort "$filename"**

**else**

**echo "File not found: $filename"**

**fi**

**;;**

**9)**

**echo "Exiting..."**

**break**

**;;**

**\*)**

**echo "Invalid choice. Please enter a number between 1 and 9."**

**;;**

**esac**

**done**

**// 3.b.)C++ program to demonstrate**

**// FCFS Disk Scheduling algorithm**

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

**using namespace std;**

**int size = 8;**

**void FCFS(int arr[], int head)**

**{**

**int seek\_count = 0;**

**int distance, cur\_track;**

**for (int i = 0; i < size; i++) {**

**cur\_track = arr[i];**

**// calculate absolute distance**

**distance = abs(cur\_track - head);**

**// increase the total count**

**seek\_count += distance;**

**// accessed track is now new head**

**head = cur\_track;**

**}**

**cout << "Total number of seek operations = "**

**<< seek\_count << endl;**

**// Seek sequence would be the same**

**// as request array sequence**

**cout << "Seek Sequence is" << endl;**

**for (int i = 0; i < size; i++) {**

**cout << arr[i] << endl;**

**}**

**}**

**// Driver code**

**int main()**

**{**

**// request array**

**int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };**

**int head = 50;**

**FCFS(arr, head);**

**return 0;**

**}**

**//6.b)Sjf CPU scheduling**

**#include <iostream>**

**using namespace std;**

**int main() {**

**// Matrix for storing Process Id, Burst**

**// Time, Average Waiting Time & Average**

**// Turn Around Time.**

**int A[100][4];**

**int i, j, n, total = 0, index, temp;**

**float avg\_wt, avg\_tat;**

**cout << "Enter number of process: ";**

**cin >> n;**

**cout << "Enter Burst Time:" << endl;**

**// User Input Burst Time and alloting Process Id.**

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

**A[i][0] = i + 1;**

**cout << "P" << i + 1 << ": ";**

**cin >> A[i][1];**

**}**

**// Sorting process according to their Burst Time.**

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

**index = i;**

**for (j = i + 1; j < n; j++)**

**if (A[j][1] < A[index][1])**

**index = j;**

**temp = A[i][1];**

**A[i][1] = A[index][1];**

**A[index][1] = temp;**

**temp = A[i][0];**

**A[i][0] = A[index][0];**

**A[index][0] = temp;**

**}**

**A[0][2] = 0;**

**// Calculation of Waiting Times**

**for (i = 1; i < n; i++) {**

**// A[i][2] = 0;**

**// for (j = 0; j < i; j++)**

**// A[i][2] += A[j][1];**

**A[i][2]=A[i-1][1]+A[i-1][2];**

**total += A[i][2];**

**}**

**avg\_wt = (float)total / n;**

**total = 0;**

**cout << "P BT WT TAT" << endl;**

**// Calculation of Turn Around Time and printing the**

**// data.**

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

**A[i][3] = A[i][1] + A[i][2];**

**total += A[i][3];**

**cout << "P" << A[i][0] << " " << A[i][1] << " " << A[i][2] << " " << A[i][3] << endl;**

**}**

**avg\_tat = (float)total / n;**

**cout << "Average Waiting Time= " << avg\_wt << endl;**

**cout << "Average Turnaround Time= " << avg\_tat << endl;**

**}**

**// 5.b)implementation of FIFO page replacement**

**// in Operating Systems.**

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

**using namespace std;**

**// Function to find page faults using FIFO**

**int pageFaults(int pages[], int n, int capacity)**

**{**

**// To represent set of current pages. We use**

**// an unordered\_set so that we quickly check**

**// if a page is present in set or not**

**unordered\_set<int> s;**

**// To store the pages in FIFO manner**

**queue<int> indexes;**

**// Start from initial page**

**int page\_faults = 0;**

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

**{**

**// Check if the set can hold more pages**

**if (s.size() < capacity)**

**{**

**// Insert it into set if not present**

**// already which represents page fault**

**if (s.find(pages[i])==s.end())**

**{**

**// Insert the current page into the set**

**s.insert(pages[i]);**

**// increment page fault**

**page\_faults++;**

**// Push the current page into the queue**

**indexes.push(pages[i]);**

**}**

**}**

**// If the set is full then need to perform FIFO**

**// i.e. remove the first page of the queue from**

**// set and queue both and insert the current page**

**else**

**{**

**// Check if current page is not already**

**// present in the set**

**if (s.find(pages[i]) == s.end())**

**{**

**// Store the first page in the**

**// queue to be used to find and**

**// erase the page from the set**

**int val = indexes.front();**

**// Pop the first page from the queue**

**indexes.pop();**

**// Remove the indexes page from the set**

**s.erase(val);**

**// insert the current page in the set**

**s.insert(pages[i]);**

**// push the current page into**

**// the queue**

**indexes.push(pages[i]);**

**// Increment page faults**

**page\_faults++;**

**}**

**}**

**}**

**return page\_faults;**

**}**

**// Driver code**

**int main()**

**{**

**int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,**

**2, 3, 0, 3, 2};**

**int n = sizeof(pages)/sizeof(pages[0]);**

**int capacity = 4;**

**cout << pageFaults(pages, n, capacity);**

**return 0;**

**}**

**//8.b) C++ program for implementation of FCFS using priority**

**// scheduling**

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

**using namespace std;**

**struct Process {**

**int pid; // Process ID**

**int bt; // CPU Burst time required**

**int priority; // Priority of this process**

**};**

**// Function to sort the Process acc. to priority**

**bool comparison(Process a, Process b)**

**{**

**return (a.priority > b.priority);**

**}**

**// Function to find the waiting time for all**

**// processes**

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

**{**

**// waiting time for first process is 0**

**wt[0] = 0;**

**// calculating waiting time**

**for (int i = 1; i < n; i++)**

**wt[i] = proc[i - 1].bt + wt[i - 1];**

**}**

**// Function to calculate turn around time**

**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 << "\nProcesses "**

**<< " Burst time "**

**<< " Waiting time "**

**<< " Turn around time\n";**

**// Calculate total waiting time and total turn**

**// around 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 " << 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;**

**}**

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

**{**

**// Sort processes by priority**

**sort(proc, proc + n, comparison);**

**cout << "Order in which processes gets executed \n";**

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

**cout << proc[i].pid << " ";**

**findavgTime(proc, n);**

**}**

**// Driver code**

**int main()**

**{**

**Process proc[]**

**= { { 1, 10, 2 }, { 2, 5, 0 }, { 3, 8, 1 } };**

**int n = sizeof proc / sizeof proc[0];**

**priorityScheduling(proc, n);**

**return 0;**

**}**