

## Source code:

### A.FCFS:

```
import java.util.*;
public class FCFS {
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
System.out.println("enter no of process: ");
int n = sc.nextInt();
int pid[] = new int[n]; // process ids
int ar[] = new int[n];
int bt[] = new int[n];
int ct[] = new int[n];
int ta[] = new int[n];
int wt[] = new int[n];
int pr[] = new int[n];
int temp;
float avgwt=0,avgta=0;
for(int i = 0; i < n; i++)
{
System.out.println("enter process " + (i+1) + " arrival time: ");
ar[i] = sc.nextInt();
System.out.println("enter process " + (i+1) + " burst time: ");
bt[i] = sc.nextInt();
System.out.println("enter process " + (i+1) + " priority time: ");
pr[i] = sc.nextInt();
pid[i] = i+1;
}
//sorting according to arrival times
for(int i = 0 ; i < n; i++)
{
for(int j=0; j < n-(i+1) ; j++)
{
if( ar[j] > ar[j+1] )
{
temp = ar[j];
ar[j] = ar[j+1];
ar[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;
}
}
}
// finding completion times
for(int i = 0 ; i < n; i++)
{
if( i == 0)
{
ct[i] = ar[i] + bt[i];
}
}
```

```

else
{
if( ar[i] > ct[i-1])
{
ct[i] = ar[i] + bt[i];
}
else
ct[i] = ct[i-1] + bt[i];
}
ta[i] = ct[i] - ar[i];    // turnaround time= completion time- arrival time
wt[i] = ta[i] - bt[i];    // waiting time= turnaround time- burst time
avgwt += wt[i];          // total waiting time
avgta += ta[i];          // total turnaround time
}
System.out.println("\npid arrival burst priority complete turn waiting");
for(int i = 0 ; i < n; i++)
{
System.out.println(pid[i] + " \t " + ar[i] + "\t" + bt[i] + "\t" + pr[i] + "\t"
+ ct[i] + "\t" + ta[i] + "\t" + wt[i] );
}
sc.close();
System.out.println("\naverage waiting time: "+ (avgwt/n));    // printing average
waiting time.
System.out.println("average turnaround time:"+(avgta/n));    // printing average
turnaround time.
}
}
}

```

## OUTPUT:

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left lists the project structure, including source files like `Assembler_One_V2.java`, `Assembler_Two.java`, and `MacroProcessor_One.java`. The main editor displays the Java code, which is the same as the code provided in the first block. The Console on the right shows the output of the program, which includes a table of process data and calculated average waiting and turnaround times.

```

-terminated- FCFS [Java Application] C:\Users\nach\p2\pool\plugin\org.eclipse.just.jop
enter no of process:
4
enter process 1 arrival time:
2
enter process 1 burst time:
2
enter process 1 priority time:
2
enter process 2 arrival time:
3
enter process 2 burst time:
3
enter process 2 priority time:
3
enter process 3 arrival time:
4
enter process 3 burst time:
4
enter process 3 priority time:
4
enter process 4 arrival time:
5
enter process 4 burst time:
5
enter process 4 priority time:
5
pid arrival burst priority complete turn waiting
1 2 2 2 2 4 2 0
2 3 3 3 7 4 1
3 4 4 4 11 7 3
4 5 5 5 16 11 6
average waiting time: 2.5
average turnaround time: 6.0

```

**B.SJF:**

```
import java.util.*;
public class SJF {
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println ("enter no of process:");
        int n = sc.nextInt();
        int pid[] = new int[n];
        int at[] = new int[n];
        int bt[] = new int[n];
        int ct[] = new int[n];
        int ta[] = new int[n];
        int wt[] = new int[n];
        int pr[] = new int[n];
        int f[] = new int[n]; // f means it is flag it checks process is completed or not
        int st=0, tot=0;
        float avgwt=0, avgta=0;
        for(int i=0;i<n;i++)
        {
            System.out.println ("enter process " + (i+1) + " arrival time:");
            at[i] = sc.nextInt();
            System.out.println ("enter process " + (i+1) + " burst time:");
            bt[i] = sc.nextInt();
            System.out.println("enter process " + (i+1) + " priority time: ");
            pr[i] = sc.nextInt();
            pid[i] = i+1;
            f[i] = 0;
        }
        boolean a = true;
        while(true)
        {
            int c=n, min=999;
            if (tot == n) // total no of process = completed process loop will be terminated
                break;
            for (int i=0; i<n; i++)
            {
                /*
                * If i'th process arrival time <= system time and its flag=0 and burst<min
                * That process will be executed first
                */
                if ((at[i] <= st) && (f[i] == 0) && (bt[i]<min))
                {
                    min=bt[i];
                    c=i;
                }
            }
            /* If c==n means c value can not updated because no process arrival time< system
            time so we increase the system time */
            if (c==n)
                st++;
            else
            {
                ct[c]=st+bt[c];
                st+=bt[c];
            }
        }
    }
}
```

```

ta[c]=ct[c]-at[c];
wt[c]=ta[c]-bt[c];
f[c]=1;
tot++;
}
}
System.out.println("\npid arrival brust complete turn waiting");
for(int i=0;i<n;i++)
{
avgwt+= wt[i];
avgta+= ta[i];
System.out.println(pid[i]+"\\t"+at[i]+"\\t"+bt[i]+"\\t"+ct[i]+"\\t"+ta[i]+"\\t"+wt[i]);
}
System.out.println ("naverage tat is "+ (float)(avgta/n));
System.out.println ("average wt is "+ (float)(avgwt/n));
sc.close();
}
}

```

## OUTPUT:

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with packages like `JRE System Library`, `src`, and `SPOS`.
- Source Editor:** Displays the `SIF.java` file with line numbers from 107 to 149. The code implements a scheduling algorithm with variables for arrival time, burst time, completion time, turnaround time, and waiting time.
- Console:** Shows the program's output. It starts with a prompt for the number of processes, followed by input for each process's arrival, burst, and priority times. The output includes a table of metrics for four processes and the calculated average turnaround time (TAT) and average waiting time (WT).

**Console Output:**

```

-terminated> SIF [Java Application] C:\Users\nach\p2\pool\plugin\org.eclipse.justi.open...
enter no of process:
4
enter process 1 arrival time:
1
enter process 1 brust time:
2
enter process 1 priority time:
1
enter process 2 arrival time:
2
enter process 2 brust time:
2
enter process 2 priority time:
2
enter process 3 arrival time:
3
enter process 3 brust time:
4
enter process 3 priority time:
4
enter process 4 arrival time:
4
enter process 4 brust time:
5
enter process 4 priority time:
4
|
pid arrival brust complete turn waiting
1 1 2 3 2 0
2 2 3 6 4 1
3 3 4 10 7 3
4 4 5 15 11 6

naverage tat is 6.0
average wt is 2.5

```

### C. Priority:

```
import java.util.Scanner;
public class Priority{
int burstTime[];
int priority[];
int arrivalTime[];
String[] processId;
int numberOfProcess;
void getProcessData(Scanner input)
{
System.out.print("Enter the number of Process for Scheduling      : ");
int inputNumberOfProcess = input.nextInt();
numberOfProcess = inputNumberOfProcess;
burstTime = new int[numberOfProcess];
priority = new int[numberOfProcess];
arrivalTime = new int[numberOfProcess];
processId = new String[numberOfProcess];
String st = "P";
for (int i = 0; i < numberOfProcess; i++)
{
processId[i] = st.concat(Integer.toString(i));
System.out.print("Enter the burst time   for Process - " + (i) + " : ");
burstTime[i] = input.nextInt();
System.out.print("Enter the arrival time for Process - " + (i) + " : ");
arrivalTime[i] = input.nextInt();
System.out.print("Enter the priority   for Process - " + (i) + " : ");
priority[i] = input.nextInt();
}
}
void sortAccordingArrivalTimeAndPriority(int[] at, int[] bt, int[] prt, String[] pid)
{
int temp;
String stemp;
for (int i = 0; i < numberOfProcess; i++)
{
for (int j = 0; j < numberOfProcess - i - 1; j++)
{
if (at[j] > at[j + 1])
{
//swapping arrival time
temp = at[j];
at[j] = at[j + 1];
at[j + 1] = temp;
//swapping burst time
temp = bt[j];
bt[j] = bt[j + 1];
bt[j + 1] = temp;
//swapping priority
temp = prt[j];
prt[j] = prt[j + 1];
prt[j + 1] = temp;
//swapping process identity
stemp = pid[j];
pid[j] = pid[j + 1];
```

```

pid[j + 1] = stemp;
}
//sorting according to priority when arrival timings are same
if (at[j] == at[j + 1])
{
if (prt[j] > prt[j + 1])
{
//swapping arrival time
temp = at[j];
at[j] = at[j + 1];
at[j + 1] = temp;
//swapping burst time
temp = bt[j];
bt[j] = bt[j + 1];
bt[j + 1] = temp;
//swapping priority
temp = prt[j];
prt[j] = prt[j + 1];
prt[j + 1] = temp;
//swapping process identity
stemp = pid[j];
pid[j] = pid[j + 1];
pid[j + 1] = stemp;
}
}
}
}
}
}
void priorityNonPreemptiveAlgorithm()
{
int finishTime[] = new int[numberOfProcess];
int bt[] = burstTime.clone();
int at[] = arrivalTime.clone();
int prt[] = priority.clone();
String pid[] = processId.clone();
int waitingTime[] = new int[numberOfProcess];
int turnAroundTime[] = new int[numberOfProcess];
sortAccordingArrivalTimeAndPriority(at, bt, prt, pid);
//calculating waiting & turn-around time for each process
finishTime[0] = at[0] + bt[0];
turnAroundTime[0] = finishTime[0] - at[0];
waitingTime[0] = turnAroundTime[0] - bt[0];
for (int i = 1; i < numberOfProcess; i++)
{
finishTime[i] = bt[i] + finishTime[i - 1];
turnAroundTime[i] = finishTime[i] - at[i];
waitingTime[i] = turnAroundTime[i] - bt[i];
}
float sum = 0;
for (int n : waitingTime)
{
sum += n;
}
float averageWaitingTime = sum / numberOfProcess;
sum = 0;

```

```

for (int n : turnAroundTime)
{
sum += n;
}
float averageTurnAroundTime = sum / numberOfProcess;
//print on console the order of processes along with their finish time & turn around time
System.out.println("Priority Scheduling Algorithm : ");
System.out.format("%20s%20s%20s%20s%20s%20s%20s\n", "ProcessId", "BurstTime",
"ArrivalTime", "Priority", "FinishTime", "WaitingTime", "TurnAroundTime");
for (int i = 0; i < numberOfProcess; i++) {
System.out.format("%20s%20d%20d%20d%20d%20d%20d\n", pid[i], bt[i], at[i],
prt[i], finishTime[i], waitingTime[i], turnAroundTime[i]);
}
System.out.format("%100s%20f%20f\n", "Average", averageWaitingTime,
averageTurnAroundTime);
}
public static void main(String[] args)
{
Scanner input = new Scanner(System.in);
Priority obj = new Priority();
obj.getProcessData(input);
obj.priorityNonPreemptiveAlgorithm();
}
}

```

## OUTPUT:

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with files like `FCFS.java`, `Priority.java`, `SF.java`, and various assembly and macro processor files.
- Editor:** Displays the `Priority.java` file. The code includes:
 

```

import java.util.Scanner;

public class Priority{
    int burstTime[];
    int priority[];
    int arrivalTime[];
    String[] processId;
    int numberOfProcess;

    void getProcessData(Scanner input)
    {
        System.out.print("Enter the number of Process for Scheduling ");
        int inputNumberOfProcess = input.nextInt();
        numberOfProcess = inputNumberOfProcess;
        burstTime = new int[numberOfProcess];
        priority = new int[numberOfProcess];
        arrivalTime = new int[numberOfProcess];
        processId = new String[numberOfProcess];
        String st = "p";
        for (int i = 0; i < numberOfProcess; i++)
        {
            processId[i] = st.concat(Integer.toString(i));
            System.out.print("Enter the burst time for Process - " + (i) + " ");

```
- Console:** Shows the execution output:
 

```

Enter the number of Process for Scheduling : 4
Enter the burst time for Process - 0 : 1
Enter the arrival time for Process - 0 : 2
Enter the priority for Process - 0 : 1
Enter the burst time for Process - 1 : 2
Enter the arrival time for Process - 1 : 3
Enter the priority for Process - 1 : 2
Enter the burst time for Process - 2 : 3
Enter the arrival time for Process - 2 : 4
Enter the priority for Process - 2 : 3
Enter the burst time for Process - 3 : 4
Enter the arrival time for Process - 3 : 5
Enter the priority for Process - 3 : 4
Priority Scheduling Algorithm :

```
- Table:** A table is displayed in the console showing the scheduling results:
 

ProcessId	BurstTime	ArrivalTime
P0	1	2
P1	2	3
P2	3	4
P3	4	5

#### D.Round Robin:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int n,i,qt,count=0,temp,sq=0,bt[10],wt[10],tat[10],rem_bt[10];
//n signifies number of process
//i is for using loops
//qt denotes Quantum Time
//count denotes when one process is completed
//temp and sq are temproray variables
//bt[10] denotes burst time
//wt[10] denotes waiting time
//tat[10] denotes turnaround time
//rem_bt[10] denotes remaining burst time
float awt=0,atat=0;
//awt represents average waiting time
//atat represents average turnaround time
printf("Enter number of process (upto 10) = ");
scanf("%d",&n);
printf("Enter burst time of process\n");
for (i=0;i<n;i++)
{
printf("P%d = ",i+1);
scanf("%d",&bt[i]);
rem_bt[i]=bt[i];
}
printf("Enter quantum time ");
scanf("%d",&qt);
while(1)
{
for (i=0,count=0;i<n;i++)
{
temp=qt;
if(rem_bt[i]==0)
{
count++;
continue;
}
if(rem_bt[i]>qt)//changing the value of remaining burst time
rem_bt[i]=rem_bt[i]-qt;
else
if(rem_bt[i]>=0)//if process is exhausted then setting remaining burst time
{
temp=rem_bt[i];
rem_bt[i]=0;
}
sq=sq+temp; //calculating turnaround time
tat[i]=sq;
}
if(n==count)//breaking the loop when all process are exhausted
break;
}
printf("\nProcess\tBurst Time\tTurnaround Time\tWaiting Time\n");
```



```

for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
awt=awt+wt[i];
atat=atat+tat[i];
printf("\n%d\t%d\t%d\t%d",i+1,bt[i],tat[i],wt[i]);
}
awt=awt/n;
atat=atat/n;
printf("\nAverage waiting Time = %f\n",awt);
printf("Average turnaround time = %f",atat);
return 0;
}

```

## OUTPUT:

The screenshot shows the GDB online Debugger interface. The top panel displays the C code for a round-robin scheduling algorithm. The bottom panel shows the program's execution output, including user input for process details and the resulting scheduling metrics.

```

main.c
108
109 {
110     wt[i]=tat[i]-bt[i];
111     awt=awt+wt[i];
112     atat=atat+tat[i];
113     printf("\n%d\t%d\t%d\t%d",i+1,bt[i],tat[i],wt[i]);
114 }
115
116 awt=awt/n;
117
118 atat=atat/n;
119 printf("\nAverage waiting Time = %f\n",awt);
120 printf("Average turnaround time = %f",atat);
121 return 0;
122 }

```

input

```

Enter number of process (upto 10) = 3
Enter burst time of process
P1 = 4
P2 = 3
P3 = 5
Enter quantum time 2

Process Burst Time    Turnaround Time    Waiting Time
1      4              8              4
2      3              9              6
3      5              12             7
Average waiting Time = 5.666667
Average turnaround time = 9.666667

...Program finished with exit code 0
Press ENTER to exit console.

```

## Conclusion:

CPU policies implemented successfully.

Input:

- No. of jobs (js) & No. of blocks (bs)
- Job size of all jobs & Block size of all blocks

For Example:

js=4

bs=5

block[] = {100, 500, 200, 300, 600};

jobs[] = {212, 417, 112, 426};

### Code:

```
import java.util.Arrays;
class First
{
// Method to allocate memory to
// blocks as per First fit algorithm
static void firstFit(int blockSize[], int m,
int processSize[], int n)
{
// Stores block id of the
// block allocated to a process
int allocation[] = new int[n];
// Initially no block is assigned to any process
for (int i = 0; i < allocation.length; i++)
allocation[i] = -1;
// pick each process and find suitable blocks
// according to its size ad assign to it
for (int i = 0; i < n; i++)
{
for (int j = 0; j < m; j++)
{
if (blockSize[j] >= processSize[i])
{
// allocate block j to p[i] process
allocation[i] = j;
// Reduce available memory in this block.
blockSize[j] -= processSize[i];
break;
}
}
}
System.out.println("\nProcess No.\tProcess Size\tBlock no.");
for (int i = 0; i < n; i++)
{
System.out.print(" " + (i+1) + "\t\t" +
processSize[i] + "\t\t");
if (allocation[i] != -1)
System.out.print(allocation[i] + 1);
else
System.out.print("Not Allocated");
System.out.println();
}
}
```

```

static void bestFit(int blockSize[], int m, int processSize[],
int n)
{
// Stores block id of the block allocated to a
// process
int allocation[] = new int[n];
// Initially no block is assigned to any process
for (int i = 0; i < allocation.length; i++)
allocation[i] = -1;
// pick each process and find suitable blocks
// according to its size ad assign to it
for (int i=0; i<n; i++)
{
// Find the best fit block for current process
int bestIdx = -1;
for (int j=0; j<m; j++)
{
if (blockSize[j] >= processSize[i])
{
if (bestIdx == -1)
bestIdx = j;
else if (blockSize[bestIdx] > blockSize[j])
bestIdx = j;
}
}
// If we could find a block for current process
if (bestIdx != -1)
{
// allocate block j to p[i] process
allocation[i] = bestIdx;
// Reduce available memory in this block.
blockSize[bestIdx] -= processSize[i];
}
}
System.out.println("\nProcess No.\tProcess Size\tBlock no.");
for (int i = 0; i < n; i++)
{
System.out.print("  " + (i+1) + "\t\t" + processSize[i] + "\t\t");
if (allocation[i] != -1)
System.out.print(allocation[i] + 1);
else
System.out.print("Not Allocated");
System.out.println();
}
}

static void worstFit(int blockSize[], int m, int processSize[],
int n)
{
// Stores block id of the block allocated to a
// process
int allocation[] = new int[n];
// Initially no block is assigned to any process
for (int i = 0; i < allocation.length; i++)
allocation[i] = -1;
// pick each process and find suitable blocks

```

```

// according to its size ad assign to it
for (int i=0; i<n; i++)
{
// Find the best fit block for current process
int wstIdx = -1;
for (int j=0; j<m; j++)
{
if (blockSize[j] >= processSize[i])
{
if (wstIdx == -1)
wstIdx = j;
else if (blockSize[wstIdx] < blockSize[j])
wstIdx = j;
}
}
// If we could find a block for current process
if (wstIdx != -1)
{
// allocate block j to p[i] process
allocation[i] = wstIdx;
// Reduce available memory in this block.
blockSize[wstIdx] -= processSize[i];
}
}
System.out.println("\nProcess No.\tProcess Size\tBlock no.");
for (int i = 0; i < n; i++)
{
System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t");
if (allocation[i] != -1)
System.out.print(allocation[i] + 1);
else
System.out.print("Not Allocated");
System.out.println();
}
}

static void NextFit(int blockSize1[], int m1, int processSize1[], int n1) {
// Stores block id of the block allocated to a
// process
int allocation[] = new int[n1], j = 0;
// Initially no block is assigned to any process
Arrays.fill(allocation, -1);
// pick each process and find suitable blocks
// according to its size ad assign to it
for (int i = 0; i < n1; i++) {
// Do not start from beginning
int count = 0;
while (j < m1) {
count++; //makes sure that for every process we traverse through entire
array maximum once only.This avoids the problem of going into infinite loop if memory is
not available
if (blockSize1[j] >= processSize1[i]) {
// allocate block j to p[i] process
allocation[i] = j;
// Reduce available memory in this block.
blockSize1[j] -= processSize1[i];
}
}
}
}

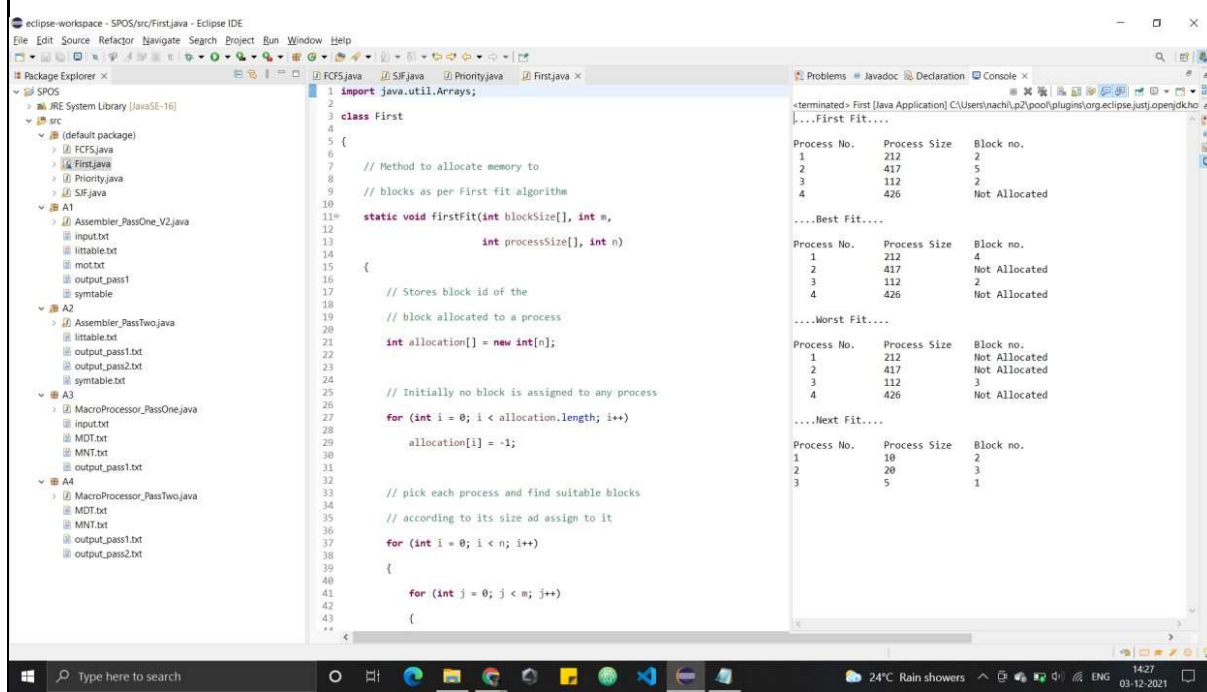
```

```

break;
}
// mod m will help in traversing the blocks from
// starting block after we reach the end.
j = (j + 1) % m1;
}
}
System.out.print("\nProcess No.\tProcess Size\tBlock no.\n");
for (int i = 0; i < n1; i++) {
System.out.print(i + 1 + "\t\t" + processSize1[i]
+ "\t\t");
if (allocation[i] != -1) {
System.out.print(allocation[i] + 1);
} else {
System.out.print("Not Allocated");
}
System.out.println("");
}
}
// Driver Code
public static void main(String[] args)
{
System.out.println("....First Fit....");
int blockSize[] = {100, 500, 200, 300, 600};
int processSize[] = {212, 417, 112, 426};
int m = blockSize.length;
int n = processSize.length;
firstFit(blockSize, m, processSize, n);
/* int blockSize[] = {100, 500, 200, 300, 600};
int processSize[] = {212, 417, 112, 426};
int m = blockSize.length;
int n = processSize.length;*/
System.out.println(" ");
System.out.println("....Best Fit....");
bestFit(blockSize, m, processSize, n);
System.out.println(" ");
System.out.println("....Worst Fit....");
worstFit(blockSize, m, processSize, n);
System.out.println(" ");
System.out.println("....Next Fit....");
int blockSize1[] = {5, 10, 20};
int processSize1[] = {10, 20, 5};
int m1 = blockSize1.length;
int n1 = processSize1.length;
NextFit(blockSize1, m1, processSize1, n1);
}
}

```

## Output:



The screenshot shows the Eclipse IDE with a Java project named 'SPOS'. The main editor displays the 'First.java' file, which implements a memory allocation strategy. The code includes a 'firstFit' method that takes an array of block sizes and a list of process sizes, and returns an array of block numbers assigned to each process. The console on the right shows the output of the program, which includes the results of the First Fit, Best Fit, Worst Fit, and Next Fit algorithms.

```
1 import java.util.Arrays;
2
3 class First
4 {
5     // Method to allocate memory to
6     // blocks as per First fit algorithm
7     static void firstFit(int blockSize[], int m,
8         int processSize[], int n)
9     {
10         // Stores block id of the
11         // block allocated to a process
12         int allocation[] = new int[n];
13
14         // Initially no block is assigned to any process
15         for (int i = 0; i < allocation.length; i++)
16             allocation[i] = -1;
17
18         // pick each process and find suitable blocks
19         // according to its size and assign to it
20         for (int i = 0; i < n; i++)
21         {
22             for (int j = 0; j < m; j++)
23             {
24                 // If the block size is greater than the process size,
25                 // then it is not suitable for the process.
26                 if (blockSize[j] < processSize[i])
27                     continue;
28                 // If the block size is equal to the process size,
29                 // then it is suitable for the process.
30                 if (blockSize[j] == processSize[i])
31                 {
32                     allocation[i] = j;
33                     break;
34                 }
35             }
36         }
37     }
38 }
```

Output:

```
<terminated> First [Java Application] C:\Users\nachi\p2\pool\plugins\org.eclipse.justi.openjdkhc
....First Fit....
Process No.    Process Size    Block no.
1             212            2
2             417            5
3             112            2
4             426            Not Allocated
....Best Fit....
Process No.    Process Size    Block no.
1             212            4
2             417            Not Allocated
3             112            2
4             426            Not Allocated
....Worst Fit....
Process No.    Process Size    Block no.
1             212            Not Allocated
2             417            Not Allocated
3             112            3
4             426            Not Allocated
....Next Fit....
Process No.    Process Size    Block no.
1             18             2
2             20             3
3             5             1
```

## Conclusion:

successfully implemented simulation of memory placement strategies.

```

****LRU****
*/
import java.io.*;
class lru
{
    public static void main(String args[])throws IOException
    {
        BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
        int f,page=0,ch,pgf=0,n,chn=0;
        boolean flag;
        int pages[];          //pgf-page fault

        System.out.println("1.LRU");
        int pt=0;
        System.out.println("enter no. of frames: ");
        f=Integer.parseInt(obj.readLine());
        int frame[]=new int[f];

        for(int i=0;i<f;i++)
        {
            frame[i]=-1;
        }

        System.out.println("enter the no of pages ");
        n=Integer.parseInt(obj.readLine());

        pages=new int[n];
        System.out.println("enter the page no ");

        for(int j=0;j<n;j++)
            pages[j]=Integer.parseInt(obj.readLine());

        int pg=0;
        for(pg=0;pg<n;pg++)
        {
            page=pages[pg];
            flag=true;
            for(int j=0;j<f;j++)
            {
                if(page==frame[j])
                {
                    flag=false;
                    break;
                }
            }
            int temp,h=3,i;
            if(flag)
            {
                if( frame[1]!=-1 && frame[2]!=-1 && frame[0]!=-1)
                {
                    temp=pages[pg-3];
                    if(temp==pages[pg-2] || temp==pages[pg-1])
                        temp=pages[pg-4];
                }
            }
        }
    }
}

```

```

        for(i=0;i<f;i++)
            if(temp==frame[i])
                break;
        frame[i]=pages[pg];
    }
    else
    {
        if(frame[0]==-1)
            frame[0]=pages[pg];
        else if(frame[1]==-1)
            frame[1]=pages[pg];
        else if(frame[2]==-1)
            frame[2]=pages[pg];
    }

    System.out.print("frame :");
    for(int j=0;j<f;j++)
        System.out.print(frame[j]+" ");
    System.out.println();
    pgf++;
}
else
{
    System.out.print("frame :");
    for(int j=0;j<f;j++)
        System.out.print(frame[j]+" ");
    System.out.println();
}

} //for

System.out.println("Page fault:"+pgf);

} //main
} //class

****Optimal****
*/
import java.util.*;
import java.io.*;
class Optimal
{
    public static void main(String args[])throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int numberOfFrames, numberOfPages, flag1, flag2, flag3, i, j, k, pos = 0, max;
        int faults = 0;
        int temp[] = new int[10];

        System.out.println("Enter number of Frames: ");
        numberOfFrames = Integer.parseInt(br.readLine());
        int frame[] = new int[numberOfFrames];

```



```

System.out.println("Enter number of Pages: ");
numberOfPages = Integer.parseInt(br.readLine());

int pages[] = new int[numberOfPages];
System.out.println("Enter the pages: ");
for(i=0; i<numberOfPages; i++)
    pages[i] = Integer.parseInt(br.readLine());

for(i = 0; i < numberOfFrames; i++)
frame[i] = -1;

for(i = 0; i < numberOfPages; ++i){
    flag1 = flag2 = 0;

    for(j = 0; j < numberOfFrames; ++j){
        if(frame[j] == pages[i]){
            flag1 = flag2 = 1;
            break;
        }
    }

    if(flag1 == 0){
        for(j = 0; j < numberOfFrames; ++j){
            if(frame[j] == -1){
                faults++;
                frame[j] = pages[i];
                flag2 = 1;
                break;
            }
        }
    }
}

if(flag2 == 0){
    flag3 = 0;

    for(j = 0; j < numberOfFrames; ++j){
        temp[j] = -1;

        for(k = i + 1; k < numberOfPages; ++k){
            if(frame[j] == pages[k]){
                temp[j] = k;
                break;
            }
        }
    }
}

for(j = 0; j < numberOfFrames; ++j){
    if(temp[j] == -1){
        pos = j;
        flag3 = 1;
        break;
    }
}

```

```

    }

    if(flag3 == 0){
        max = temp[0];
        pos = 0;

        for(j = 1; j < numberOfFrames; ++j){
            if(temp[j] > max){
                max = temp[j];
                pos = j;
            }
        }

        frame[pos] = pages[i];
        faults++;
    }

//    System.out.print();

    for(j = 0; j < numberOfFrames; ++j){
        System.out.print("\t" + frame[j]);
    }
}

System.out.println("\n\nTotal Page Faults: " + faults);
}
}

```

### Optimal Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

### LRU Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6