import java.util.Arrays;

import java.util.Scanner;

/\*\*

\*

\* @author ADMIN

\*/

class first\_fit

{

// Method to allocate memory to

// blocks as per First fit algorithm

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();

}

}

}

//Java program for next fit memory management algorithm

class next\_fit

{

//Function to allocate memory to blocks as per Next fit

//algorithm

void NextFit(int blockSize[], int m, int processSize[], int n) {

// Stores block id of the block allocated to a

// process

int allocation[] = new int[n], 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 < n; i++) {

// Do not start from beginning

int count =0;

while (j < m) {

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 (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;

}

// mod m will help in traversing the blocks from

// starting block after we reach the end.

j = (j + 1) % m;

}

}

System.out.print("\nProcess No.\tProcess Size\tBlock no.\n");

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("");

}

}

}

//Java implementation of worst - Fit algorithm

class worst\_fit

{

// Method to allocate memory to blocks as per worst fit

// algorithm

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();

}

}

}

//Java implementation of Best - Fit algorithm

class best\_fit

{

// Method to allocate memory to blocks as per Best fit

// algorithm

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();

}

}

}

// Driver Code for All Algos:

public class Memory {

public static void main(String[] args){

first\_fit first = new first\_fit();

next\_fit next = new next\_fit();

worst\_fit worst = new worst\_fit();

best\_fit best = new best\_fit();

Scanner scan = new Scanner(System.in);

while(true){

int choice;

System.out.println();

System.out.println("Enter the number of Blocks: ");

int m = scan.nextInt();

System.out.println("Enter the number of Processes: ");

int n = scan.nextInt();

int blockSize[] = new int[m];

int processSize[] = new int[n];

System.out.println("Enter the Size of all the blocks: ");

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

blockSize[i] = scan.nextInt();

}

System.out.println("Enter the size of all processes: ");

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

processSize[i] = scan.nextInt();

}

System.out.println();

System.out.println("Menu");

System.out.println("1. First Fit ");

System.out.println("2. Next Fit");

System.out.println("3. Worst Fit");

System.out.println("4. Best Fit");

System.out.println("5. exit");

System.out.println("Select the algorithm you want to implement: ");

choice = scan.nextInt();

switch(choice){

case 1:

System.out.println("First Fit Output");

first.firstFit(blockSize, m, processSize, n);

break;

case 2:

System.out.println("Next Fit Output");

next.NextFit(blockSize, m, processSize, n);

break;

case 3:

System.out.println("Worst Fit Output");

worst.worstFit(blockSize, m, processSize, n);

break;

case 4:

System.out.println("Best Fit Output");

best.bestFit(blockSize, m, processSize, n);

break;

case 5:

System.out.println("Exiting the code...");

return;

default:

System.out.println("Invalid option");

}

}

}

}

**Output:-**

Enter the number of Blocks:

5

Enter the number of Processes:

4

Enter the Size of all the blocks:

100

500

200

300

600

Enter the size of all processes:

212

417

112

426

Menu

1. First Fit

2. Next Fit

3. Worst Fit

4. Best Fit

5. exit

Select the algorithm you want to implement:

1

First Fit Output

Process No. Process Size Block no.

1 212 2

2 417 5

3 112 2

4 426 Not Allocated

Enter the number of Blocks:

3

Enter the number of Processes:

3

Enter the Size of all the blocks:

5

10

20

Enter the size of all processes:

10

20

5

Menu

1. First Fit

2. Next Fit

3. Worst Fit

4. Best Fit

5. exit

Select the algorithm you want to implement:

2

Next Fit Output

Process No. Process Size Block no.

1 10 2

2 20 3

3 5 1

Enter the number of Blocks:

5

Enter the number of Processes:

4

Enter the Size of all the blocks:

100

500

200

300

600

Enter the size of all processes:

212

417

112

426

Menu

1. First Fit

2. Next Fit

3. Worst Fit

4. Best Fit

5. exit

Select the algorithm you want to implement:

3

Worst Fit Output

Process No. Process Size Block no.

1 212 5

2 417 2

3 112 5

4 426 Not Allocated

Enter the number of Blocks:

5

Enter the number of Processes:

4

Enter the Size of all the blocks:

100

500

200

300

600

Enter the size of all processes:

212

417

112

426

Menu

1. First Fit

2. Next Fit

3. Worst Fit

4. Best Fit

5. exit

Select the algorithm you want to implement:

4

Best Fit Output

Process No. Process Size Block no.

1 212 4

2 417 2

3 112 3

4 426 5

Enter the number of Blocks:

1

Enter the number of Processes:

1

Enter the Size of all the blocks:

10

Enter the size of all processes:

10

Menu

1. First Fit

2. Next Fit

3. Worst Fit

4. Best Fit

5. exit

Select the algorithm you want to implement:

5

Exiting the code...

BUILD SUCCESSFUL (total time: 4 minutes 52 seconds)