import java.util.Scanner;

class MemoryBlock {

int size;

boolean isAllocated;

public MemoryBlock(int size) {

this.size = size;

this.isAllocated = false;

}

}

class Process {

int requiredSize;

public Process(int requiredSize) {

this.requiredSize = requiredSize;

}

}

public class MemoryPlacementSimulator {

// Best Fit Memory Allocation

public static void bestFit(MemoryBlock[] memory, Process[] processes) {

System.out.println("\nBest Fit Allocation:");

for (Process process : processes) {

int bestIndex = -1;

int minDifference = Integer.MAX\_VALUE;

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

if (!memory[i].isAllocated && memory[i].size >= process.requiredSize) {

int difference = memory[i].size - process.requiredSize;

if (difference < minDifference) {

minDifference = difference;

bestIndex = i;

}

}

}

if (bestIndex != -1) {

memory[bestIndex].isAllocated = true;

System.out.println("Allocated Process of size " + process.requiredSize + " to block of size " + memory[bestIndex].size);

} else {

System.out.println("Process of size " + process.requiredSize + " could not be allocated.");

}

}

}

// Next Fit Memory Allocation

public static void nextFit(MemoryBlock[] memory, Process[] processes) {

System.out.println("\nNext Fit Allocation:");

int lastAllocatedIndex = 0;

for (Process process : processes) {

boolean allocated = false;

// Start allocation from the last allocated position

for (int i = lastAllocatedIndex; i < memory.length; i++) {

if (!memory[i].isAllocated && memory[i].size >= process.requiredSize) {

memory[i].isAllocated = true;

lastAllocatedIndex = i; // Update last allocated block index

System.out.println("Allocated Process of size " + process.requiredSize + " to block of size " + memory[i].size);

allocated = true;

break;

}

}

// If not found, wrap around to the beginning of the memory

if (!allocated) {

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

if (!memory[i].isAllocated && memory[i].size >= process.requiredSize) {

memory[i].isAllocated = true;

lastAllocatedIndex = i;

System.out.println("Allocated Process of size " + process.requiredSize + " to block of size " + memory[i].size);

allocated = true;

break;

}

}

}

if (!allocated) {

System.out.println("Process of size " + process.requiredSize + " could not be allocated.");

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int choice = 0;

// Read the number of memory blocks and the processes only once

System.out.print("Enter the number of memory blocks: ");

int n = sc.nextInt();

MemoryBlock[] memory = new MemoryBlock[n];

System.out.println("Enter the sizes of memory blocks:");

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

memory[i] = new MemoryBlock(sc.nextInt());

}

System.out.print("Enter the number of processes: ");

int m = sc.nextInt();

Process[] processes = new Process[m];

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

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

processes[i] = new Process(sc.nextInt());

}

// Loop for continuous user interaction

while (true) {

System.out.println("\nSelect Memory Allocation Strategy:");

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

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

System.out.println("3. Both Best Fit and Next Fit");

System.out.println("4. Exit");

System.out.print("Enter your choice (1/2/3/4): ");

choice = sc.nextInt();

switch (choice) {

case 1:

// Best Fit Allocation

bestFit(memory, processes);

break;

case 2:

// Next Fit Allocation

nextFit(memory, processes);

break;

case 3:

// Both Best Fit and Next Fit Allocation

bestFit(memory, processes);

// Reset memory allocation status for next fit

for (MemoryBlock block : memory) {

block.isAllocated = false;

}

nextFit(memory, processes);

break;

case 4:

// Exit the program

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

return;

default:

System.out.println("Invalid choice. Please select 1, 2, 3, or 4.");

}

}

}

}