ASSIGNMENT NO – 6

MemoryPlacementStrategies.java file

```
import java.util.Arrays;

import java.util.Scanner;


public class MemoryPlacementStrategies {


        // Best Fit Algorithm

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

                int[] allocation = new int[n];

                int[] remblockSize = new int[m];

                System.arraycopy(blockSize, 0, remblockSize, 0, m);


                Arrays.fill(allocation, -1);


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

                        int bestIdx = -1;

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

                                if (remblockSize[j] >= processSize[i]) {

                                        if (bestIdx == -1 || remblockSize[j] < remblockSize[bestIdx])

                                                bestIdx = j;

                                }

                        }

                        if (bestIdx != -1) {

                                allocation[i] = bestIdx;

                                remblockSize[bestIdx] -= processSize[i];

                        }

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

        printAllocation(processSize, allocation, remblockSize, m);

    }


    // First Fit Algorithm

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

        int[] allocation = new int[n];

        int[] remblockSize = new

        int[m];

        System.arraycopy(blockSize, 0, remblockSize, 0, m);


        Arrays.fill(allocation, -1);


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

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

                if (remblockSize[j] >= processSize[i]) {

                    allocation[i] = j;

                    remblockSize[j] -= processSize[i];

                    break;

                }

            }

        }


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

        printAllocation(processSize, allocation, remblockSize, m);

    }


    // Next Fit Algorithm
```

```
static void nextFit(int[] blockSize, int m, int[] processSize, int n) {
    int[] allocation = new int[n];
    int[] remblockSize = new int[m];
    System.arraycopy(blockSize, 0, remblockSize, 0, m);

    Arrays.fill(allocation, -1);

    int j = 0;
    for (int i = 0; i < n; i++) {
        int count = 0;
        boolean allocated = false;
        while (count < m) {
            if (remblockSize[j] >= processSize[i]) {
                allocation[i] = j;
                remblockSize[j] -= processSize[i];
                allocated = true;
                break;
            }
            j = (j + 1) % m;
            count++;
        }
        if (!allocated) {
            allocation[i] = -1;
        } else {
            j = (j + 1) % m;
        }
    }

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

```
        printAllocation(processSize, allocation, remblockSize, m);
    }

    // Worst Fit Algorithm
    static void worstFit(int[] blockSize, int m, int[] processSize, int n) {
        int[] allocation = new int[n];
        int[] remblockSize = new int[m];
        System.arraycopy(blockSize, 0, remblockSize, 0, m);

        Arrays.fill(allocation, -1);

        for (int i = 0; i < n; i++) {
            int worstIdx = -1;
            for (int j = 0; j < m; j++) {
                if (remblockSize[j] >= processSize[i]) {
                    if (worstIdx == -1 || remblockSize[j] > remblockSize[worstIdx])
                        worstIdx = j;
                }
            }
            if (worstIdx != -1) {
                allocation[i] = worstIdx;
                remblockSize[worstIdx] -= processSize[i];
            }
        }

        System.out.println("\nWorst Fit Allocation:");
        printAllocation(processSize, allocation, remblockSize, m);
    }
```

```
// Utility to print allocation results
static void printAllocation(int[] processSize, int[] allocation, int[] remblockSize, int m) {
        System.out.println("Process No.\tProcess Size\tBlock No.\tRemaining Block Size");
        for (int i = 0; i < processSize.length; i++) {
                System.out.print((i + 1) + "\t\t" + processSize[i] + "\t\t");
                if (allocation[i] != -1) {
                        int block = allocation[i];
                        System.out.println((block + 1) + "\t\t" + remblockSize[block]);
                } else {
                        System.out.println("Not Allocated\t-");
                }
        }
}

public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.print("Enter number of memory blocks: ");
        int m = in.nextInt();

        int[] blockSize = new int[m];
        System.out.println("Enter size of each memory block:");
        for (int i = 0; i < m; i++) {
                blockSize[i] = in.nextInt();
        }

        System.out.print("Enter number of processes: ");
        int n = in.nextInt();
        int[] processSize = new int[n];
        System.out.println("Enter size of each process:");
        for (int i = 0; i < n; i++) {
            processSize[i] = in.nextInt();
        }

        // Call each strategy
        bestFit(blockSize, m, processSize, n);
        firstFit(blockSize, m, processSize, n);
        nextFit(blockSize, m, processSize, n);
        worstFit(blockSize, m, processSize, n);

        in.close();
    }
}
```

OUTPUT:

```
Enter number of memory blocks: 5
Enter size of each memory block:
100
500
200
300
600
Enter number of processes: 4
Enter size of each process:
212
417
112
426
```

```
Best Fit Allocation:

Process No.   Process Size   Block No.   Remaining Block Size

1          212          4          88

2          417          2          83

3          112          3          88

4          426          5          174


First Fit Allocation:

Process No.   Process Size   Block No.   Remaining Block Size

1          212          2          176

2          417          5          183

3          112          2          176

4          426          Not Allocated   -


Next Fit Allocation:

Process No.   Process Size   Block No.   Remaining Block Size

1          212          2          176

2          417          5          183

3          112          2          176

4          426          Not Allocated   -


Worst Fit Allocation:

Process No.   Process Size   Block No.   Remaining Block Size

1          212          5          276

2          417          2          83

3          112          5          276

4          426          Not Allocated   -
```

```
...Program finished with exit code 0

Press ENTER to exit console.
```