ASSIGNMENT NO.6

(Memory Management Algorithms)

PROGRAM:-

```
import java.util.Arrays;
import java.util.Scanner;
// Java implementation of First - Fit algorithm
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 {
   void NextFit(int blockSize[], int m, int processSize[], int n) {
      int allocation[] = new int[n];
      Arrays.fill(allocation, -1);
      int j = 0;  // Initialize the index j to 0
      for (int i = 0; i < n; i++) {
         int count = 0;
         while (count < m) {  // Ensure we check all blocks
            if (blockSize[j] >= processSize[i]) {
               allocation[i] = j;
               blockSize[j] -= processSize[i];
               break;
            }
            j = (j + 1) % m;  // Move to the next block, wrapping around
            count++;
         }
      }
      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 Main {
  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();
     String continueChoice;
     Scanner scan = new Scanner(System.in);
     while(true){
          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();
       }
        do{
       int choice;
       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");
        }System.out.print("Do you want to continue (yes/no)? ");
         continueChoice = scan.next().toLowerCase();
          }while(continueChoice.equals("yes"));return;
          }
  }
}
```

OUTPUT:-

```
Enter the number of Blocks:
4
Enter the number of Processes:
4
Enter the Size of all the blocks:
100
500
200
300
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	Not Allocated
3	112	2
4	426	Not Allocated

Do you want to continue (yes/no)? yes

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	212	4
2	417	Not Allocated
3	112	2
4	426	Not Allocated

Do you want to continue (yes/no)? yes

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	Not Allocated
2	417	Not Allocated
3	112	3
4	426	Not Allocated

Do you want to continue (yes/no)? yes

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	Not Allocated
2	417	Not Allocated
3	112	Not Allocated
4	426	Not Allocated

Do you want to continue (yes/no)?

yes

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...