- **Problem Statement :** Write a program to simulate memory placement strategies – best fit , first fit , next fit and worst fit.

- **Programs:**

1. first_fit.java

```
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();
         }
        }
}
```

2. next_fit.java:

```
import java.util.Arrays;

import java.util.Scanner;

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("");
        }
}
```

3. worst_fit.java:

```
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();
                }
        }
}
```

4. best_fit.java:

```
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();
        }
}
}
```

5. Main.java:

```
import java.util.Arrays;

import java.util.Scanner;
// 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();

        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:**

1. **First Fit Output :**

```
C:\Users\HP\Desktop\Best fit worst fit>javac Main.java

C:\Users\HP\Desktop\Best fit worst fit>java Main

Enter the number of Blocks:
5
Enter the number of Processes:
4
Enter the Size of all the blocks:
20
100
40
200
10
Enter the size of all processes:
90
50
30
40

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              90              2
 2              50              4
 3              30              3
 4              40              4
```

2. Best Fit Output :

```
Enter the number of Blocks:
4
Enter the number of Processes:
3
Enter the Size of all the blocks:
100
50
30
120
Enter the size of all processes:
40
10
30

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             40               2
   2             10               2
   3             30               3
```

3. 3. Worst Fit :

```
Enter the number of Blocks:
5
Enter the number of Processes:
4
Enter the Size of all the blocks:
100
50
30
120
35
Enter the size of all processes:
140
60
10
20

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             140              Not Allocated
   2             60               4
   3             10               1
   4             20               1
```

4. Next Fit Output

```
Enter the number of Blocks:
5
Enter the number of Processes:
4
Enter the Size of all the blocks:
100
50
120
30
40
Enter the size of all processes:
50
20
30
75

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               50              1
2               20              1
3               30              1
4               75              3
```