Name: Rushikesh Kaibhati Palve
Roll No.: 31258

Date: 1/20 | Page No.

Assignment No. 2

DOP: 17-08-2021 DOS:- 17-08-2021

Problem Definition :-

Write a program in C/C++/Java/Python to simulate Memory placement strategies - Best Fit, First Fit, Next Fit and Worst Fit.

Learning Objectives :-

① To implement the simulation of memory placement strategies.
② To compare different placement algorithms.
③ To learn different memory management techniques in the operating system.

Learning Outcomes :-

After performing the assignment, students will be able to -
① Implement the simulation of memory placement strategies.
② Understand different placement algorithms.
③ Consider better placement algorithm which will avoid internal fragmentation.

Concepts Related Theory :-

In Operating system, the following are four common memory management techniques.

1) Single contiguous allocation :

Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.

2) Partitioned allocation :

Memory is divided into different blocks or partitions. Each process is allocated according to the requirement.

3) Paged memory management :

Memory is divided into fixed-sized units called page frames, used in a virtual memory environment.

4) Segmented memory management :

Memory is divided into different segments (a segment is a logical grouping of the process data or code). In this management, allocated memory doesn't have to be contiguous.

Most of the operating systems (for example Windows and Linux) use segmentation with Paging. A process is divided into segments and individual segments have pages.

In Partition Allocation, when there is more than one partition freely available to accomodate a process's request, a partition

must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.

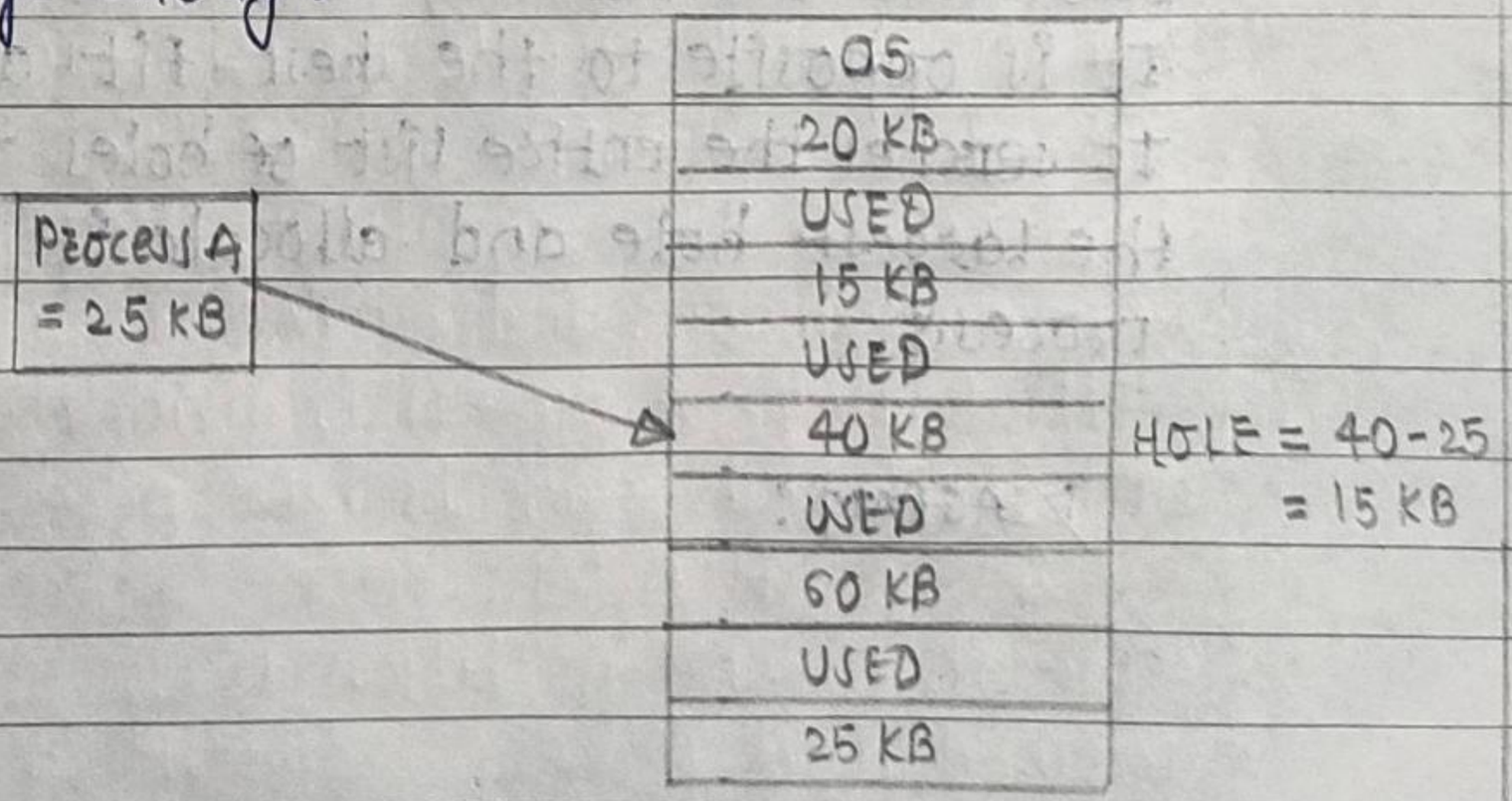
There are different Placement Algorithm:

A.] First Fit
B.] Best Fit
C.] Worst Fit
D.] Next Fit

1.] First Fit:

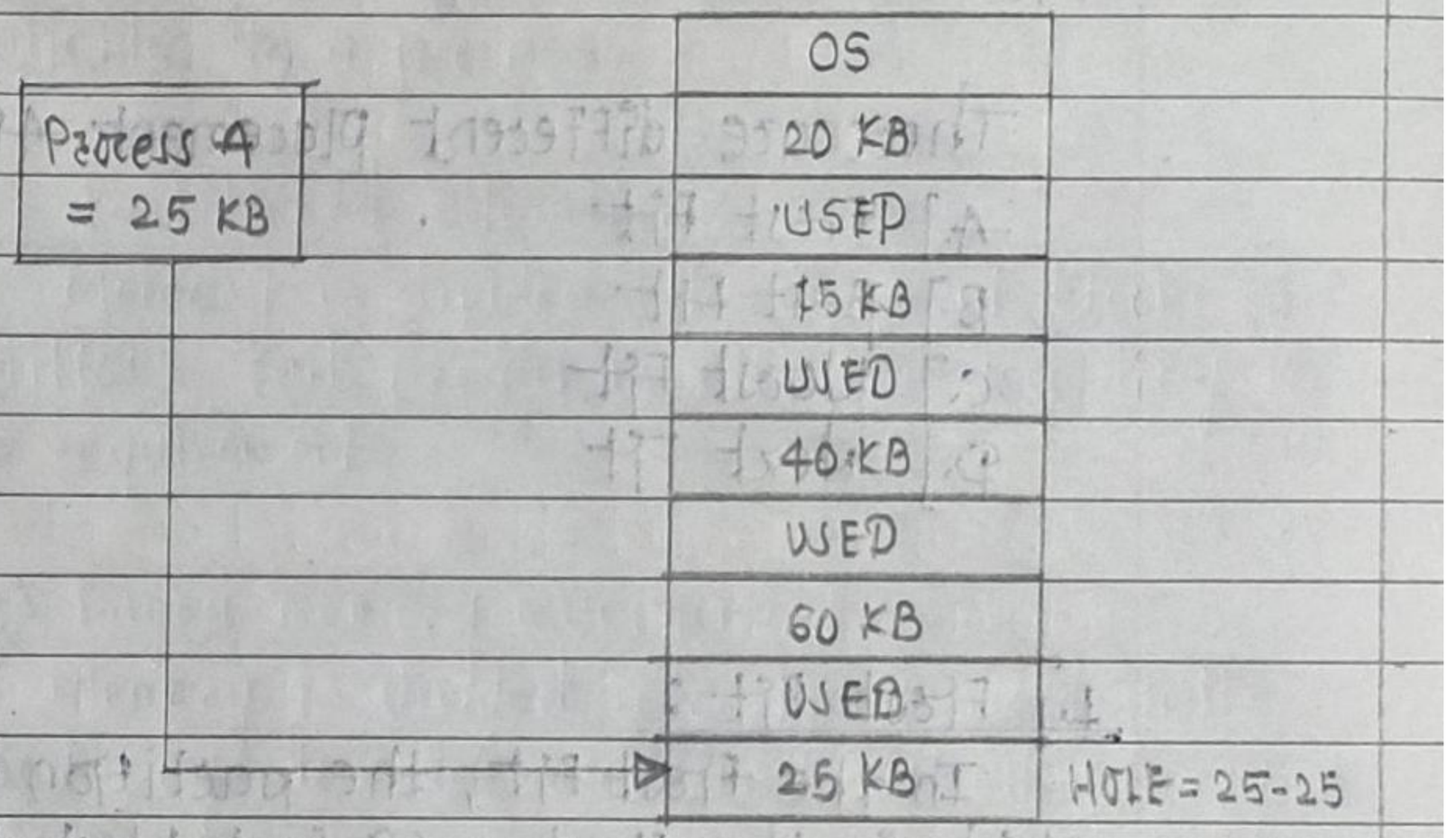
In the first Fit, the partition is allocated which is the first sufficient block from the top of Main memory. It scans memory from the beginning and chooses the first available block that is large enough.

Thus it allocates the first hole that is large enough.



2.] Best Fit :

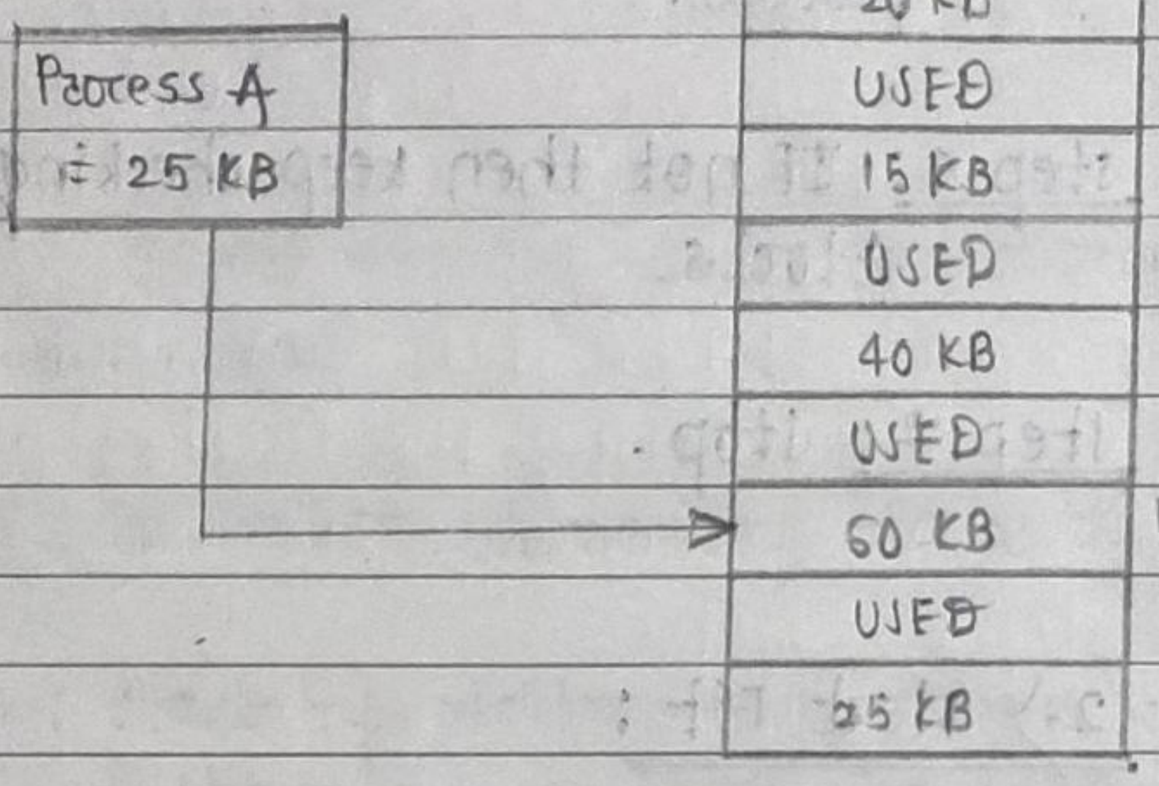
Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



3.] Worst Fit :

Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to the process.

DIAGRAM :



4.] Next Fit :-

Next Fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

ALGORITHM :-

1.) First Fit :

Step 1 : Start

Step 2 : Input memory blocks with size and processes with size.

Step 3 : Initialize all memory blocks as free.

Step 4 : Start by picking each process and check if it can be assigned to current block.

Step 5 : If size-of-process <= size-of-block

If yes then assign and check for next process.

Step 6: If not then keep checking the further blocks.

Step 7: Stop.

2.) Next Fit :

Step 1 : Start.

Step 2 : Input the number of memory blocks and their sizes and initializes all the blocks as free.

Step 3 : Input the number of processes and their sizes.

Step 4 : Start by picking each process and check if it can be assigned to the current block, if yes allocate it the required memory and check for next process but from the block where we left not from starting.

Step 5: If the current block size is smaller then keep checking the further blocks.

Step 6: Stop.

: Date : 1/20 | Page No :

3.) Best Fit :-

Step 1 : Start.

Step 2 : Input memory blocks and processes with sizes.

Step 3 : Initialize all memory blocks as free.

Step 4 : Start by picking each process and find the minimum block size that can be assigned to current process i.e. find min (blocksize[1], blocksize[2], ---blocksize[n] > processsize[current], if found then assign it to the current process.

Step 5 : If not then leave that process and keep checking the further processes.

Step 6 : Stop.

4.) Worst Fit :

Step 1 : Start.

Step 2 : Input memory blocks and processes with sizes.

Step 3 : Initialize all memory blocks as free.

Step 4 : Start by picking each process and find the maximum block size that can be

assigned to current process i.e. find
max (blocksize[1], blocksize[2], ... blocksize[n])
> processsize[current], if found then assign it to the current process.

Step 5: If not then leave that process and keep checking the further processes.

Step 6: Stop.

TEST CASES :-

Test Case Id	steps to be executed	Expected Result	Actual Result	Test Case Pass/Fail
1.)	No. of blocks = 5 No. of processes = 4 sizes of block = {100, 500, 200, 300, 600}	a) First Fit: P. No. | Process size | Block No. 1 | 212 | 2 2 | 417 | 5 3 | 112 | 2 4 | 426 | Not Allocated b) Next Fit: sizes of processes = {212, 417, 112, 426} P. No. | Process size | Block No. 1 | 212 | 2 2 | 417 | 5 3 | 112 | 2 4 | 426 | Not Allocated	a) First Fit: P. No. | Process size | Block No. 1 | 212 | 2 2 | 417 | 5 3 | 112 | 2 4 | 426 | Not Allocated b) Next Fit: P. No. | Process size | Block No. 1 | 212 | 2 2 | 417 | 5 3 | 112 | 2 4 | 426 | Not Allocated	pass

Test Case Id.	Steps to be Executed	Expected Result	Actual Result	Test case Pass/Fail

c) Best Fit:

P. No.	Process size	Block No.
1	212	4
2	417	2
3	112	3
4	426	5

c) Best Fit:

P. No.	Process size	Block No.
1	212	4
2	417	2
3	112	3
4	426	5

d) Worst Fit:

P. No.	Process size	Block No.
1	212	5
2	417	2
3	112	5
4	426	Not Allocated

d) Worst Fit:

P. No.	Process size	Block No.
1	212	5
2	417	2
3	112	5
4	426	Not Allocated

CONCLUSION :-

We have successfully implemented the simulation of memory placement strategies. We also understood different memory management techniques in the operating system.

CODE :-

```
/*
 * Problem Statement :-
 *  Write a program in C/C++/Java/Python to simulate memory placement strategies -
 *  First Fit, Next Fit, Best Fit, Worst Fit
 */
```

First Fit :-

```
package assignmentNo_2;

public class First_Fit
{
    void first_fit(int input_blocks[], int input_processes[], int m, int n)
    {
        int blocks[] = new int[m];
        for (int i = 0; i < input_blocks.length; i++)
            blocks[i] = input_blocks[i];

        int processes[] = new int[n];
        for (int i = 0; i < processes.length; i++)
            processes[i] = input_processes[i];

        int allocating[] = new int[n];
        for (int i = 0; i < allocating.length; i++)
            allocating[i] = -1;

        for (int j = 0; j < n; j++) {
            for (int i = 0; i < m; i++) {
                if (processes[j] <= blocks[i]) {
                    blocks[i] = blocks[i] - processes[j];
                    allocating[j] = i;
                    break;
                }
            }
        }

        System.out.println("\n\t\t\t Process No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
```

```
            System.out.print("\t\t\t  " + (i+1) + "\t\t" +
                    processes[i] + "\t\t");
            if (allocating[i] != -1)
                System.out.print(allocating[i] + 1);
            else
                System.out.print("Not Allocated");
            System.out.println();
        }
    }

}
```

Next Fit :-

```
package assignmentNo_2;

public class Next_Fit
{
    void next_fit(int input_blocks[], int input_processes[], int m, int n)
    {
        int blocks[] = new int[m];
        for (int i = 0; i < input_blocks.length; i++)
            blocks[i] = input_blocks[i];

        int processes[] = new int[n];
        for (int i = 0; i < processes.length; i++)
            processes[i] = input_processes[i];

        int allocating[] = new int[n];
        for (int i = 0; i < allocating.length; i++)
            allocating[i] = -1;

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

            if (flag == m - 1)
                i = 0;

            while (i < m) {
                if (processes[j] <= blocks[i]) {
                    blocks[i] = blocks[i] - processes[j];
                    allocating[j] = i;
```

```
                    flag = i;
                    break;
                }
                i++;
            }
        }

        System.out.println("\n\t\t\t Process No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
            System.out.print("\t\t\t  " + (i+1) + "\t\t" +
                    processes[i] + "\t\t");
            if (allocating[i] != -1)
                System.out.print(allocating[i] + 1);
            else
                System.out.print("Not Allocated");
            System.out.println();
        }
    }
}
```

Best Fit :-

```
package assignmentNo_2;

public class Best_Fit
{
    void best_fit(int input_blocks[], int input_processes[], int m, int n)
    {
        int blocks[] = new int[m];
        for (int i = 0; i < input_blocks.length; i++)
            blocks[i] = input_blocks[i];

        int processes[] = new int[n];
        for (int i = 0; i < processes.length; i++)
            processes[i] = input_processes[i];

        int allocating[] = new int[n];
        for (int i = 0; i < allocating.length; i++)
            allocating[i] = -1;

        for (int j = 0; j < n; j++) {
            int Best_block = -1;
```

```
            for (int i = 0; i < m; i++) {
                if (blocks[i] >= processes[j]) {
                    if (Best_block == -1)
                        Best_block = i;
                    else if (blocks[i] < blocks[Best_block])
                        Best_block = i;
                }
            }
            if (Best_block != -1) {
                allocating[j] = Best_block;
                blocks[Best_block] = blocks[Best_block] - processes[j];
            }
        }

        System.out.println("\n\t\t\t Process No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
            System.out.print("\t\t\t  " + (i+1) + "\t\t" +
                    processes[i] + "\t\t");
            if (allocating[i] != -1)
                System.out.print(allocating[i] + 1);
            else
                System.out.print("Not Allocated");
            System.out.println();
        }
    }
}
```

Worst Fit :-

```
package assignmentNo_2;

public class Worst_Fit
{
    void worst_fit(int input_blocks[], int input_processes[], int m, int n)
    {
        int blocks[] = new int[m];
        for (int i = 0; i < input_blocks.length; i++)
            blocks[i] = input_blocks[i];

        int processes[] = new int[n];
        for (int i = 0; i < processes.length; i++)
            processes[i] = input_processes[i];
```

```
        int allocating[] = new int[n];
        for (int i = 0; i < allocating.length; i++)
            allocating[i] = -1;

        for (int j = 0; j < n; j++) {
            int Max_block = -1;
            for (int i = 0; i < m; i++) {
                if (blocks[i] >= processes[j]) {
                    if (Max_block == -1)
                        Max_block = i;
                    else if (blocks[i] > blocks[Max_block])
                        Max_block = i;
                }
            }
            if (Max_block != -1) {
                allocating[j] = Max_block;
                blocks[Max_block] -= processes[j];
            }
        }

        System.out.println("\n\t\t\t Process No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++)
        {
            System.out.print("\t\t\t  " + (i+1) + "\t\t" +
                    processes[i] + "\t\t");
            if (allocating[i] != -1)
                System.out.print(allocating[i] + 1);
            else
                System.out.print("Not Allocated");
            System.out.println();
        }
    }
}
```

AssignmentNo2.java

```
package assignmentNo_2;

import java.util.Scanner;

public class AssignmentNo2
{
    public static void main(String args[])
    {
        int blocks[],  processes[],  nh,  np;
        Scanner sc = new Scanner(System.in);
        boolean flag = true;

        System.out.println("\n\n\t\t\t === MEMORY PLACEMENT STRATEGIES ===");

        System.out.print("\n\n\t\t Enter the number of Holes : ");
        nh = sc.nextInt();

        System.out.print("\n\t\t Enter the number of Processes : ");
        np = sc.nextInt();

        blocks = new int[nh];
        processes = new int[np];

        System.out.println("\n\t\t === Enter the sizes of blocks ... ");
        for(int i=0;i<nh;i++)
        {
            System.out.print("\n\t\t\t Enter the size of block no. "+ (i+1) +" = ");

            blocks[i]=sc.nextInt();
        }

        System.out.println("\n\t\t === Enter the sizes of Processes ... ");
        for(int i=0;i<np;i++)
        {
            System.out.print("\n\t\t\t Enter the size of Process no. "+ (i+1) +" = ");

            processes[i] = sc.nextInt();
        }

        while(flag)
        {
            System.out.println("\n ==== Main-
Menu ==== \n\t 1. First Fit \n\t 2. Next Fit"
                    + "\n\t 3. Best Fit \n\t 4. Worst Fit \n\t 5. EXIT...");
```

```
            System.out.print("\n\t Enter choice : ");
            int choice = sc.nextInt();

            switch(choice)
            {
            case 1:
                System.out.println("\n\t\t\t\t 1.]  First Fit ");
                First_Fit obj1 = new First_Fit();
                obj1.first_fit(blocks, processes, nh, np);
                System.out.println("\n\t\t\t ========================================
=");
                break;

            case 2:
                System.out.println("\n\t\t\t\t 2.]  Next Fit ");
                Next_Fit obj2 = new Next_Fit();
                obj2.next_fit(blocks, processes, nh, np);
                System.out.println("\n\t\t\t ========================================
=");
                break;

            case 3:
                System.out.println("\n\t\t\t\t 3.]  Best Fit ");
                Best_Fit obj3 = new Best_Fit();
                obj3.best_fit(blocks, processes, nh, np);
                System.out.println("\n\t\t\t ========================================
=");
                break;

            case 4:
                System.out.println("\n\t\t\t\t 4.]  Worst Fit ");
                Worst_Fit obj4 = new Worst_Fit();
                obj4.worst_fit(blocks, processes, nh, np);
                System.out.println("\n\t\t\t ========================================
=");
                break;

            case 5:
                flag = false;
                System.out.print("\n\t\t\t\t Thank You ...!!");
                System.out.println("\n\n\t\t\t\t ****=====*****");
                break;

            default:
                System.out.print("\n\t\t Invalid Choice ...!!");
```

```
            }
        }
        sc.close();
    }
}
```

OUTPUT :-

```
                === MEMORY PLACEMENT STRATEGIES ===

            Enter the number of Holes : 5

            Enter the number of Processes : 4

            === Enter the sizes of blocks ...

                Enter the size of block no. 1 = 100

                Enter the size of block no. 2 = 500

                Enter the size of block no. 3 = 200

                Enter the size of block no. 4 = 300

                Enter the size of block no. 5 = 600

            === Enter the sizes of Processes ...

                Enter the size of Process no. 1 = 212

                Enter the size of Process no. 2 = 417

                Enter the size of Process no. 3 = 112

                Enter the size of Process no. 4 = 426

 ==== Main-Menu ====
      1. First Fit
      2. Next Fit
      3. Best Fit
      4. Worst Fit
      5. EXIT...
```

```
      Enter choice : 1

                   1.]  First Fit

                Process No.     Process Size     Block no.
                 1        212        2
                 2        417        5
                 3        112        2
                 4        426        Not Allocated

                ========================================

==== Main-Menu ====
      1. First Fit
      2. Next Fit
      3. Best Fit
      4. Worst Fit
      5. EXIT...

      Enter choice : 2

                   2.]  Next Fit

                Process No.     Process Size     Block no.
                 1        212        2
                 2        417        5
                 3        112        2
                 4        426        Not Allocated

                ========================================

==== Main-Menu ====
      1. First Fit
      2. Next Fit
      3. Best Fit
      4. Worst Fit
      5. EXIT...

      Enter choice : 3

                   3.]  Best Fit

                Process No.     Process Size     Block no.
                 1        212        4
                 2        417        2
                 3        112        3
                 4        426        5

                ========================================

==== Main-Menu ====
      1. First Fit
      2. Next Fit
      3. Best Fit
```

```
      4. Worst Fit
      5. EXIT...

      Enter choice : 4

                      4.]  Worst Fit

                Process No.     Process Size     Block no.
                 1        212        5
                 2        417        2
                 3        112        5
                 4        426        Not Allocated

                ========================================

==== Main-Menu ====
      1. First Fit
      2. Next Fit
      3. Best Fit
      4. Worst Fit
      5. EXIT...

      Enter choice : 5

                     Thank You ...!!

                     *****=====*****
```