Operating System (CS301)

Assignment - 5

U19CS012

1. To implement **First Fit, Best Fit** and **Worst Fit** storage allocation algorithms for memory management.

Description

- A set of **holes**, of various sizes is scattered through the memory at any given time.
- When a process arrives and needs the memory, the system **searches for a hole** that is large enough for this process.
- The first-fit, best-fit and worst-fit are strategies used to select a free hole from the set of available holes.

Implementation details

- ✓ Free space is maintained as a linked list of nodes with each node having the starting byte address and the ending byte address of a free block.
- ✓ Each memory request consists of the **process-id** and the **amount of storage space** required in bytes.
- ✓ Allocated memory space is again maintained as a linked list of nodes with each node having the process-id, starting byte address and the ending byte address of the allocated space.
- ✓ When a process finishes (taken as input) the appropriate node from the allocated list should be deleted and this free disk space should be added to the free space list.
- ✓ [Care should be taken to merge contiguous free blocks into one single block. This result in deleting more than one node from the free space list and changing the start and end address in the appropriate node].

First-Fit

Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. You can stop searching as soon as you find a free hole that is large enough.

Best-Fit

Allocate the smallest hole that is big enough. You must search the entire list unless the list is kept ordered by size. The strategy produces the smallest leftover hole.

Worst fit

Allocate the biggest hole.

Code

```cpp
#include <bits/stdc++.h>
using namespace std;

// Structure of Process Node
class Node
{
public:
    int start;
    int end;
    int size;
    int id;
    Node *next;
    Node(int start, int end)
    {
        this->start = start;
        this->end = end;
        this->size = end - start + 1;
        this->id = -1;
        this->next = NULL;
    }
};

Node *memory = NULL;

// F(x) to Print all the Choice to User [In Particular Algorithm]
void printSubChoice();

// F(x) to Check if Process Exist or Not
bool checkProcess(int processId);

// F(x) to Deallocate the Memory
void DeallocateMemory();

// F(x) to Allocate Memory Using First Fit Algorithm
void firstFit();
```

```cpp
// F(x) to Allocate Memory Using Best Fit Algorithm
void bestFit();

// F(x) to Allocate Memory Using Worst Fit Algorithm
void worstFit();

// F(x) to Print Memory Segment Details
void printMemorySegment(Node *node);

// F(x) to Display Final Summary
void displayAll();

// F(x) to Display Allocated Memory
void displayAllocated();

// Utility Function for First Fit Algorithm
void firstFitUtil();

// Utility Function for Best Fit Algorithm
void bestFitUtil();

// Utility Function for Worst Fit Algorithm
void worstFitUtil();

// F(x) to Print Menu of Various Algorithm
void printMainChoice();

// F(x) to Select Algorithm Based on Used Input
void selectAlgorithm();

int main()
{
    int noPartitions;
    cout << "Enter No. of Partitions in Memory : ";
    cin >> noPartitions;

    Node *curr;

    for (int i = 0; i < noPartitions; i++)
    {
        int start, end;
        cout << "Starting and Ending Address of partition " << i + 1 << " : ";
        cin >> start >> end;

        Node *newNode = new Node(start, end);

        if (memory == NULL)
        {
            memory = newNode;
            curr = memory;
        }
        else
        {
            curr->next = newNode;
            curr = curr->next;
        }
    }

    selectAlgorithm();
    return 0;
}

// F(x) to Print all the Choice to User [In Particular Algorithm]
void printSubChoice()
{
    cout << "    " << endl;
    cout << "1 -> Entry / Allocate" << endl;
    cout << "2 -> Exit / Deallocate" << endl;
    cout << "3 -> Display Whole Memory" << endl;
    cout << "4 -> Display Allocated Memory" << endl;
    cout << "5 -> Back to Algorithm" << endl;
    cout << "    " << endl;
}

// F(x) to Check if Process Exist or Not
bool checkProcess(int processId)
{
    Node *tmp = memory;
    bool flag = false;
    while (tmp != NULL)
    {
        if (tmp->id == processId)
        {
            flag = true;
            break;
        }
        tmp = tmp->next;
    }
    return flag;
}

// F(x) to Allocate Memory Using First Fit Algorithm
void firstFit()
{
    int processId;
    int sizeNeeded;
    cout << "Enter the Process Id : ";

    cin >> processId;
```

```cpp
    // Validation of Input from User
    if (checkProcess(processId))
    {
        cout << "Invalid Process Id" << endl;
        return;
    }

    cout << "Enter Size required by Process : ";
    cin >> sizeNeeded;

    Node *tmp = memory;
    bool flag = false;

    while (tmp != NULL)
    {
        // If Size of Block is >= than Requirement and Not Visited
        if (tmp->size >= sizeNeeded and tmp->id == -1)
        {
            // Mark it Visited
            tmp->id = processId;

            int startP = tmp->start;
            int endP = startP + sizeNeeded - 1;
            int newStart = endP + 1;

            // There is Some Remaining Space in Block
            if (newStart <= tmp->end)
            {
                Node *newNode = new Node(newStart, tmp->end);
                tmp->end = endP;
                tmp->size = sizeNeeded;
                newNode->next = tmp->next;
                tmp->next = newNode;
            }

            cout << "Memory Allocated Succesfully!" << endl;
            flag = true;
            break;
        }

        tmp = tmp->next;
    }

    if (!flag)
    {
        cout << "Can't Allocate the memory for this process!" << endl;
    }
}
```

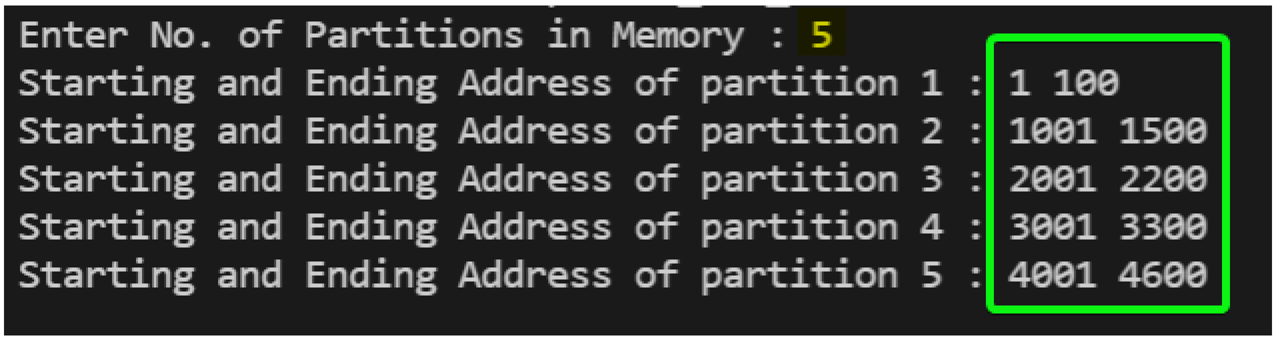
```
// F(x) to Allocate Memory Using Best Fit Algorithm
void bestFit()
{
    int processId;
    int sizeNeeded;
    cout << "Enter the process Id : ";
    cin >> processId;
    if (checkProcess(processId))
    {
        cout << "Invalid Process Id" << endl;
        return;
    }
    cout << "Enter required size by process : ";
    cin >> sizeNeeded;

    Node *tmp = memory;
    bool flag = false;
    Node *ans;

    int maxSize = INT_MAX;

    while (tmp != NULL)
    {
        if (tmp->size >= sizeNeeded and tmp->id == -1 and tmp->size < maxSize)
        {

            flag = true;
            ans = tmp;
            maxSize = tmp->size;
        }
        tmp = tmp->next;
    }
    if (!flag)
    {
        cout << "Can't allocate the memory for this process!" << endl;
        return;
    }

    ans->id = processId;
    int startP = ans->start;
    int endP = startP + sizeNeeded - 1;
    int newStart = endP + 1;

    if (newStart <= ans->end)
    {
        Node *newNode = new Node(newStart, ans->end);
        ans->end = endP;
        ans->size = sizeNeeded;
        newNode->next = ans->next;
        ans->next = newNode;
```

```cpp
    }
    cout << "Memory allocated succesfully!" << endl;
}

// F(x) to Allocate Memory Using Worst Fit Algorithm
void worstFit()
{
    int processId;
    int sizeNeeded;
    cout << "Enter the process Id : ";
    cin >> processId;
    if (checkProcess(processId))
    {
        cout << "Invalid Process Id" << endl;
        return;
    }
    cout << "Enter required size by process : ";
    cin >> sizeNeeded;

    Node *tmp = memory;
    bool flag = false;
    Node *ans;
    int maxSize = -1;

    while (tmp != NULL)
    {
        if (tmp->size >= sizeNeeded and tmp->id == -1 and tmp->size > maxSize)
        {

            flag = true;
            ans = tmp;
            maxSize = tmp->size;
        }
        tmp = tmp->next;
    }
    if (!flag)
    {
        cout << "Can't allocate the memory for this process!" << endl;
        return;
    }

    ans->id = processId;
    int startP = ans->start;
    int endP = startP + sizeNeeded - 1;
    int newStart = endP + 1;

    if (newStart <= ans->end)
    {
        Node *newNode = new Node(newStart, ans->end);
        ans->end = endP;
        ans->size = sizeNeeded;
        newNode->next = ans->next;
        ans->next = newNode;
    }
    cout << "Memory allocated succesfully!" << endl;
}

// F(x) to Deallocate the Memory
void DeallocateMemory()
{
    int processId;
    cout << "Enter the Process ID to be Deallocated : ";
    cin >> processId;

    bool flag = false;
    Node *tmp = memory;

    int offset = 0;

    // Traverse the Linked List to Find the Process to be Deallocated
    while (tmp != NULL)
    {
        if (tmp->id == processId)
        {
            flag = true;
            tmp->id = -1;
            break;
        }
        tmp = tmp->next;
        offset++;
    }

    // If Not Found
    if (!flag)
    {
        cout << "No Such process is running ! " << endl;
        return;
    }

    tmp = memory;

    while (tmp != NULL and tmp->next != NULL)
    {
        if (tmp->end + 1 == tmp->next->start and tmp->id == -1 and tmp->next->id == -1)
        {
            tmp->end = tmp->next->end;
            tmp->size = tmp->end - tmp->start + 1;

            Node *toDelete = tmp->next;
            tmp->next = tmp->next->next;
            delete toDelete;
        }
        else
        {
            tmp = tmp->next;
        }
    }

    cout << "Process Exited Successfully, Memory Freed!" << endl;
}

// F(x) to Print Memory Segment Details
void printMemorySegment(Node *node)
{
    cout << "   " << endl;
    cout << "Start : " << node->start << endl;
    cout << "End : " << node->end << endl;
    cout << "Size : " << node->size << endl;
    if (node->id != -1)
    {
        cout << "Process Id : " << node->id << endl;
    }
    else
    {
        cout << "No process Allocated" << endl;
    }
}

// F(x) to Display Final Summary
void displayAll()
{
    Node *tmp = memory;

    int totalMemory = 0;
    int freeMemory = 0;

    while (tmp != NULL)
    {
        if (tmp->id == -1)
        {
            freeMemory += tmp->size;
        }
        totalMemory += tmp->size;
        printMemorySegment(tmp);
        tmp = tmp->next;
    }

    cout << "   " << endl;
    cout << "Total memory : " << totalMemory << endl;
    cout << "Free memory : " << freeMemory << endl;
    cout << "Allocated memory : " << totalMemory - freeMemory << endl;
    cout << "    " << endl;
}

// F(x) to Display Allocated Memory
void displayAllocated()
{
    Node *tmp = memory;
    int allocatedMemory = 0;
    while (tmp != NULL)
    {
        if (tmp->id != -1)
        {
            allocatedMemory += tmp->size;
            printMemorySegment(tmp);
        }

        tmp = tmp->next;
    }
    cout << "    " << endl;
    cout << "Allocated memory : " << allocatedMemory << endl;
    cout << "    " << endl;
}

// Utility Function for First Fit Algorithm
void firstFitUtil()
{
    int subChoice = -1;
    while (subChoice != 5)
    {
        printSubChoice();
        cout << "Choice : ";
        cin >> subChoice;
        switch (subChoice)
        {
        case 1:
        {
            firstFit();
            break;
        }
        case 2:
        {
            DeallocateMemory();
            break;
        }
        case 3:
        {
            displayAll();
            break;
        }
        case 4:
        {
            displayAllocated();
            break;
        }
        case 5:
        {
            cout << "Back to Main Menu (<-)" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input! " << endl;
            break;
        }
        }
    }
}

// Utility Function for Best Fit Algorithm
void bestFitUtil()
{
    int subChoice = -1;
    while (subChoice != 5)
    {
        printSubChoice();
        cout << "Choice : ";
        cin >> subChoice;
        switch (subChoice)
        {
        case 1:
        {
            bestFit();

            break;
        }
        case 2:
        {
            DeallocateMemory();
            break;
        }
        case 3:
        {
            displayAll();
            break;
        }
        case 4:
        {
            displayAllocated();
            break;
```

```cpp
        }
        case 5:
        {
            cout << "Back to Main Menu (<-)" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input\n";
            break;
        }
        }
    }
}

// Utility Function for Worst Fit Algorithm
void worstFitUtil()
{
    int subChoice = -1;
    while (subChoice != 5)
    {
        printSubChoice();
        cout << "Choice : ";
        cin >> subChoice;
        switch (subChoice)
        {
        case 1:
        {
            worstFit();
            break;
        }
        case 2:
        {
            DeallocateMemory();
            break;
        }
        case 3:
        {
            displayAll();
            break;
        }
        case 4:
        {
            displayAllocated();
            break;
        }
        case 5:
        {
            cout << "Back to Main Menu (<-)" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input\n";
            break;
        }
        }
    }
}

// F(x) to Print Menu of Various Algorithm
void printMainChoice()
{
    cout << "   " << endl;
    cout << "Select the Algorithm" << endl;
    cout << "1 -> First Fit Algorithm" << endl;
    cout << "2 -> Best Fit Algorithm" << endl;
    cout << "3 -> Worst Fit Algorithm" << endl;
    cout << "4 -> Exit the App" << endl;
    cout << "   " << endl;
}

// F(x) to Select Algorithm Based on Used Input
void selectAlgorithm()
{
    int mainChoice = -1;
    while (mainChoice != 4)
    {
        printMainChoice();
        cout << "Choice : ";
        cin >> mainChoice;
        switch (mainChoice)
        {
        case 1:
        {
            firstFitUtil();
            break;
        }
        case 2:
        {
            bestFitUtil();
            break;
        }
        case 3:
        {
            worstFitUtil();
            break;
        }
        case 4:
        {
            cout << "Thank you for using our Application!" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input\n";
            break;
        }
        }
    }
}
```

Output

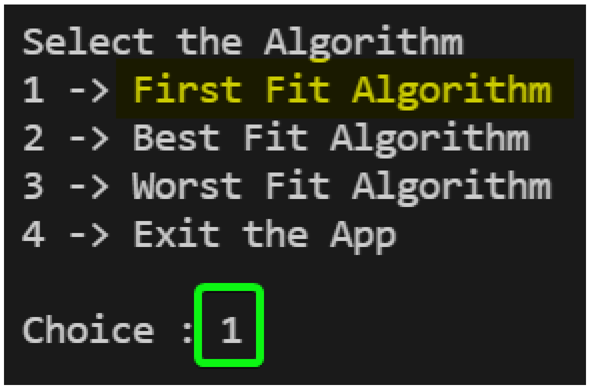
Step 1: Adding Memory Partitions



```
Enter No. of Partitions in Memory : 5
Starting and Ending Address of partition 1 : 1 100
Starting and Ending Address of partition 2 : 1001 1500
Starting and Ending Address of partition 3 : 2001 2200
Starting and Ending Address of partition 4 : 3001 3300
Starting and Ending Address of partition 5 : 4001 4600
```

Start	End	Size(in KB)
1	100	100
1001	1500	500
2001	2200	200
3001	3300	300
4001	4600	600

Step 2: Selecting an Algorithm



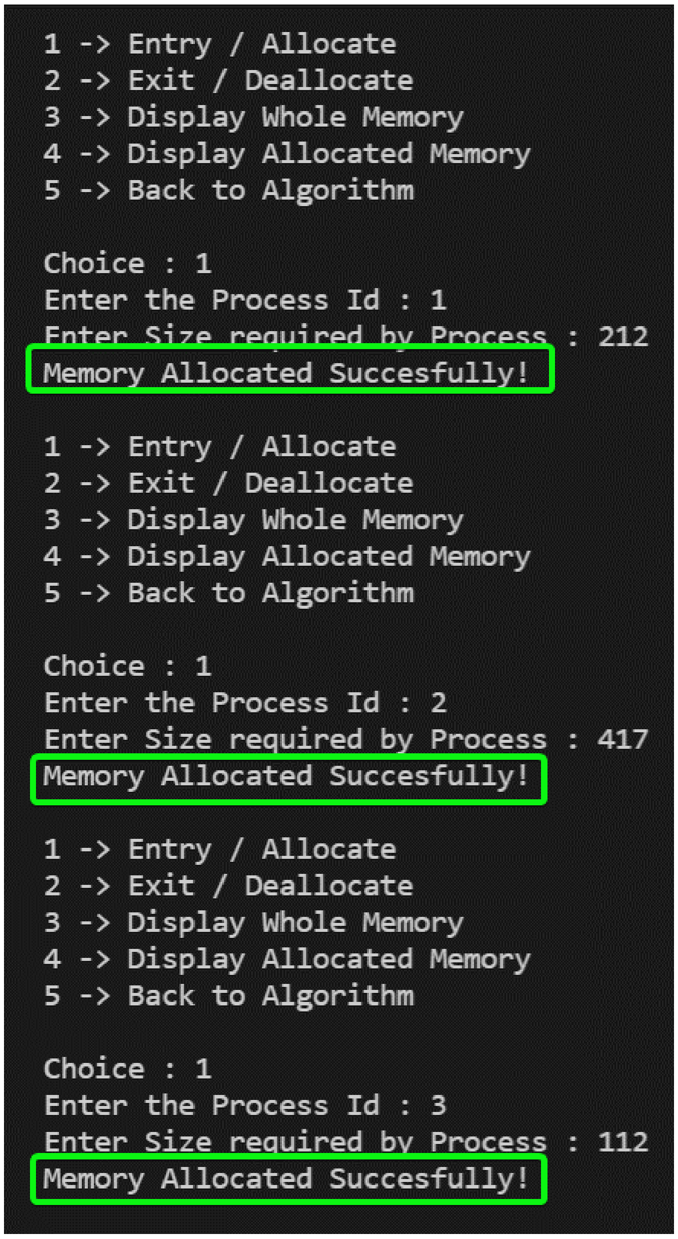
```
Select the Algorithm
1 -> First Fit Algorithm
2 -> Best Fit Algorithm
3 -> Worst Fit Algorithm
4 -> Exit the App

Choice : 1
```

Step 3: **First Fit** Algorithm Selected

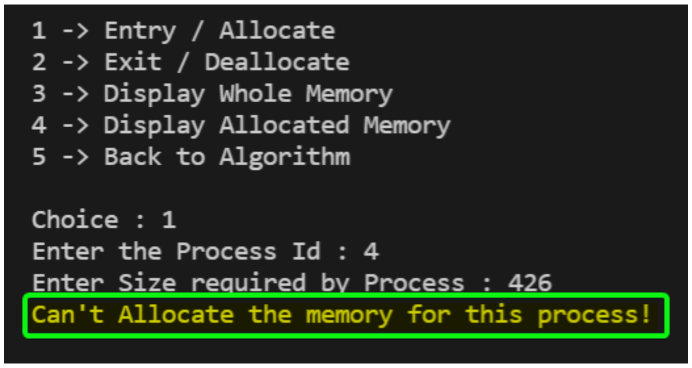
Required Size (in KB)	**Partition Size (in KB) used**	**New Partition created (if any)**
212	500	288KB partition created
417	600	183KB partition created
112	288	176KB partition created
426	-	-

Note: Using First Fit algorithm we **cannot allocate** process 4th which required 426KB of memory.

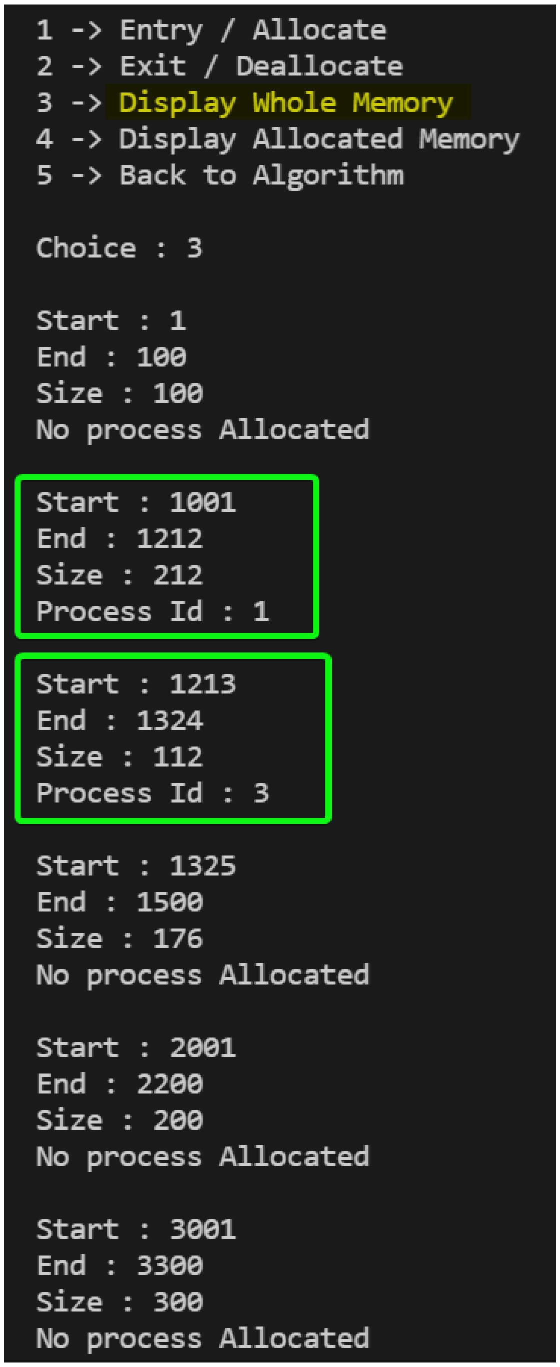
The same is displayed here in our program:

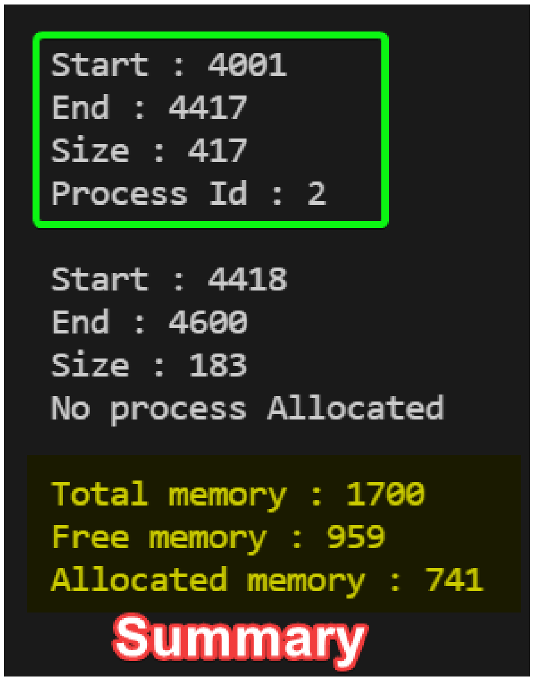


```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the Process Id : 1
Enter Size required by Process : 212
Memory Allocated Succesfully!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the Process Id : 2
Enter Size required by Process : 417
Memory Allocated Succesfully!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the Process Id : 3
Enter Size required by Process : 112
Memory Allocated Succesfully!
```

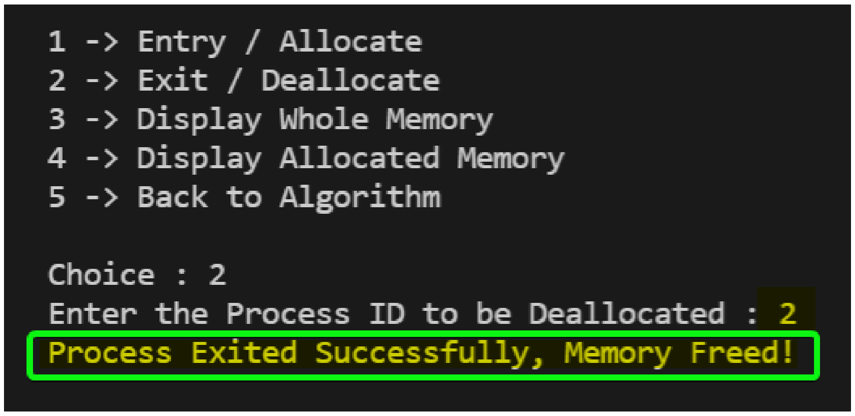

```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the Process Id : 4
Enter Size required by Process : 426
Can't Allocate the memory for this process!
```

Our Memory would be as below after allocating:


```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 3

Start : 1
End : 100
Size : 100
No process Allocated

Start : 1001
End : 1212
Size : 212
Process Id : 1

Start : 1213
End : 1324
Size : 112
Process Id : 3

Start : 1325
End : 1500
Size : 176
No process Allocated

Start : 2001
End : 2200
Size : 200
No process Allocated

Start : 3001
End : 3300
Size : 300
No process Allocated
```

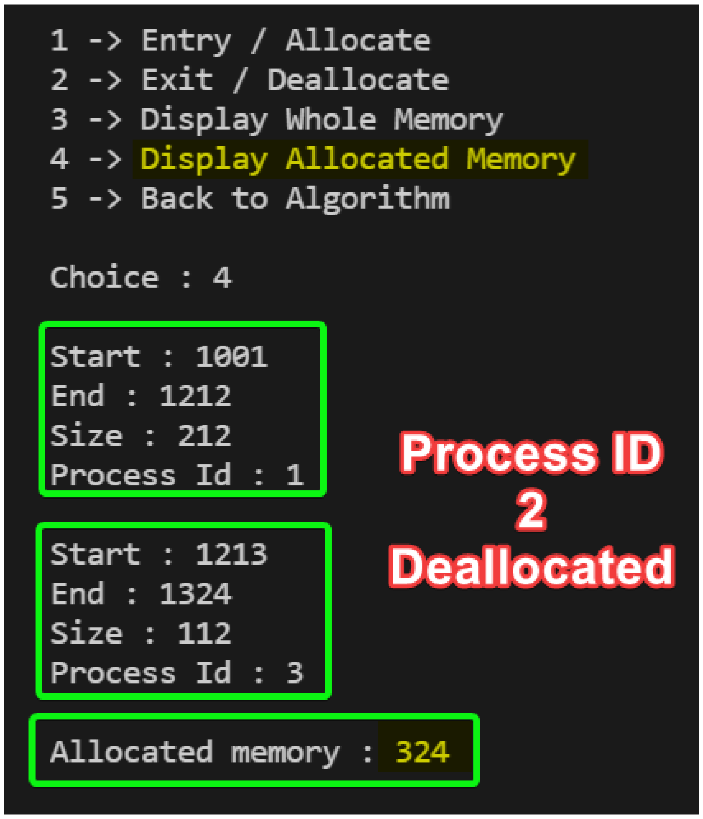

```
Start : 4001
End : 4417
Size : 417
Process Id : 2

Start : 4418
End : 4600
Size : 183
No process Allocated

Total memory : 1700
Free memory : 959
Allocated memory : 741
```
Summary

Step 4: Let's now deallocate 2nd process using 417KB of memory.



```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 2
Enter the Process ID to be Deallocated : 2
Process Exited Successfully, Memory Freed!
```

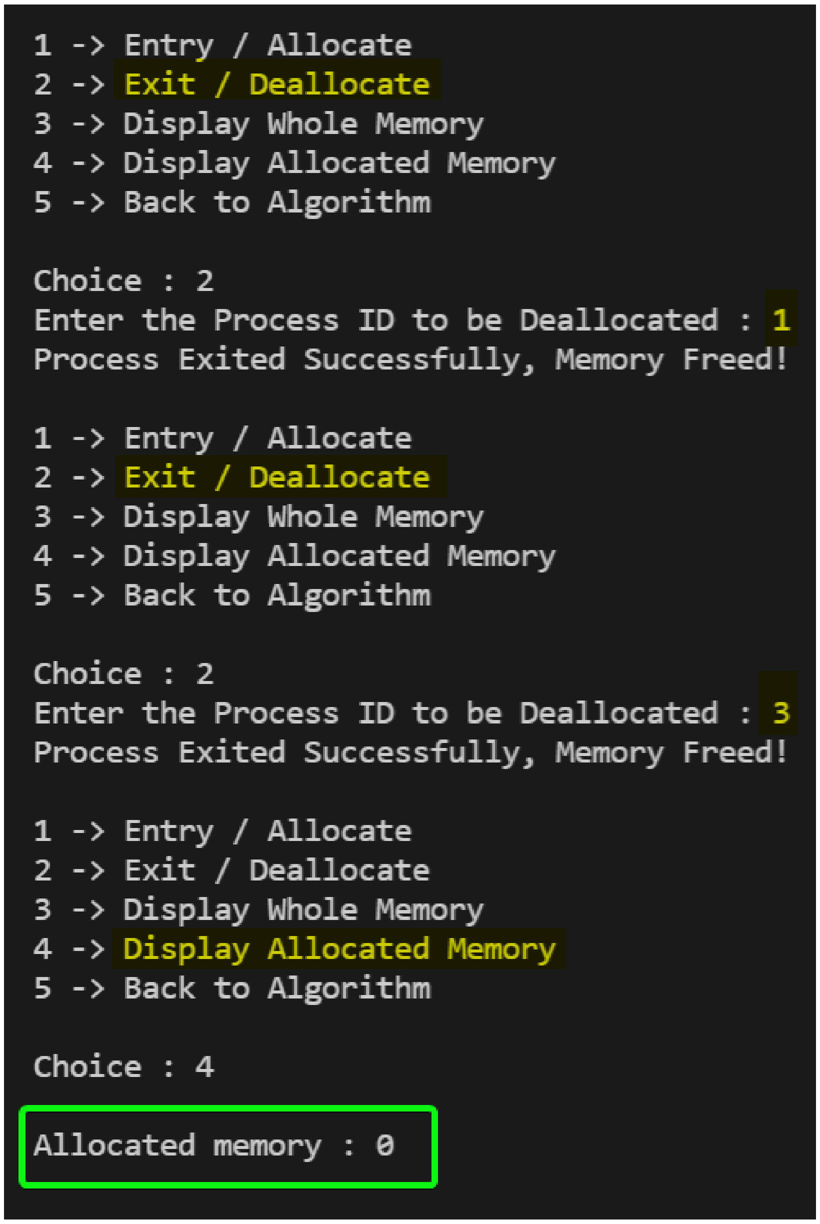
Let's check it from our memory!



```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 4

Start : 1001
End : 1212
Size : 212
Process Id : 1

Start : 1213
End : 1324
Size : 112
Process Id : 3

Allocated memory : 324
```

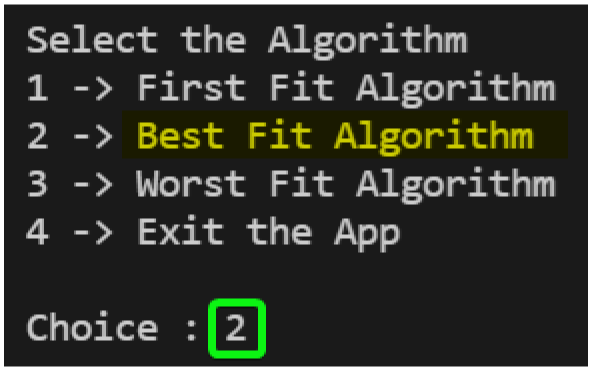
We can observe that <u>2nd process is no more running</u> and allocated memory is also reduced to 324KB.

<u>Let's De-allocate all the memory and try another algorithm</u>.



```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 2
Enter the Process ID to be Deallocated : 1
Process Exited Successfully, Memory Freed!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 2
Enter the Process ID to be Deallocated : 3
Process Exited Successfully, Memory Freed!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 4

Allocated memory : 0
```

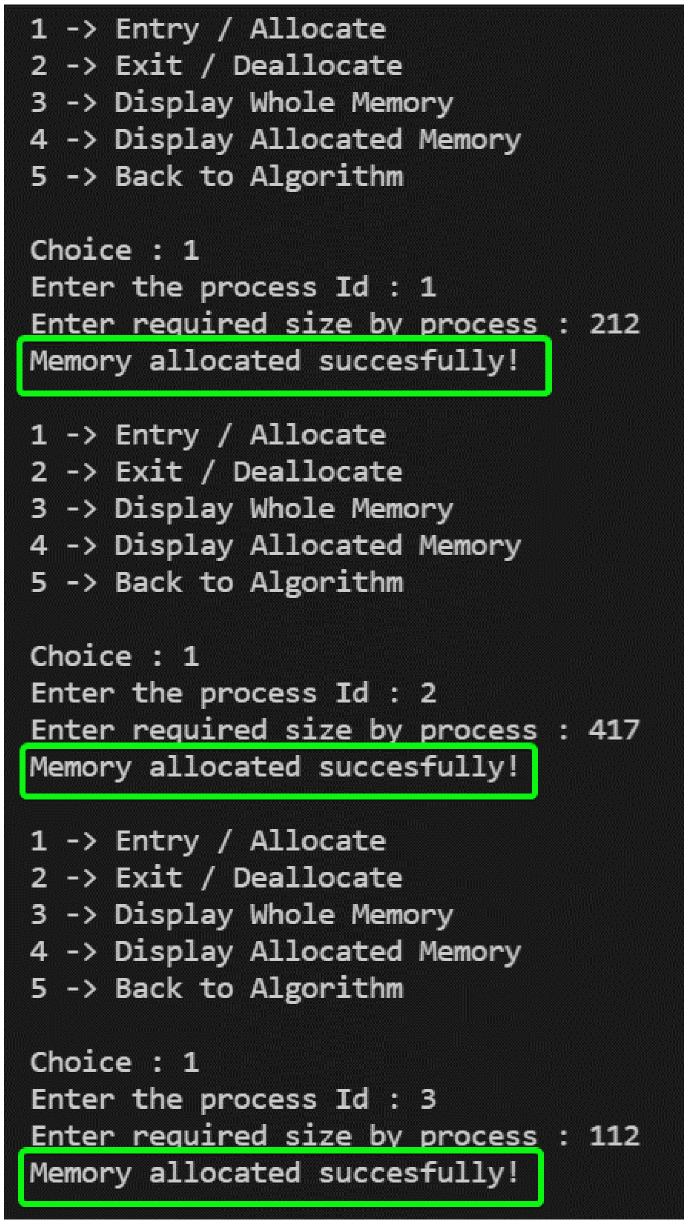
Best Fit Algorithm Selected



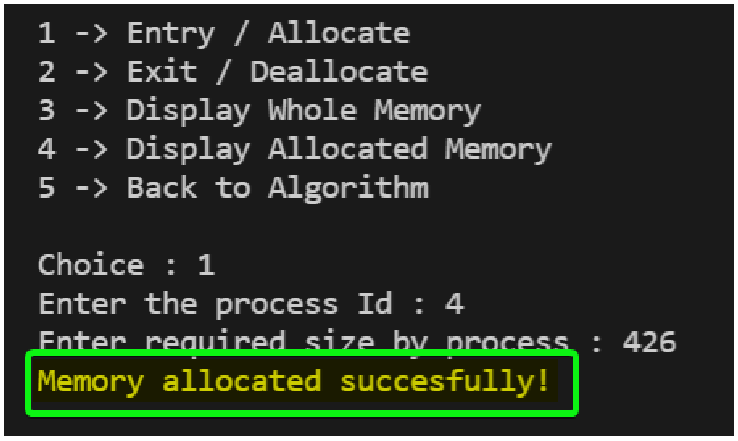
```
Select the Algorithm
1 -> First Fit Algorithm
2 -> Best Fit Algorithm
3 -> Worst Fit Algorithm
4 -> Exit the App

Choice : 2
```

Required Size (in KB)	Partition Size (in KB) used	New Partition created (if any)
212	300	88KB partition created
417	500	83KB partition created
112	200	88KB partition created
426	600	174KB partition created

Same can be observed below in our application:

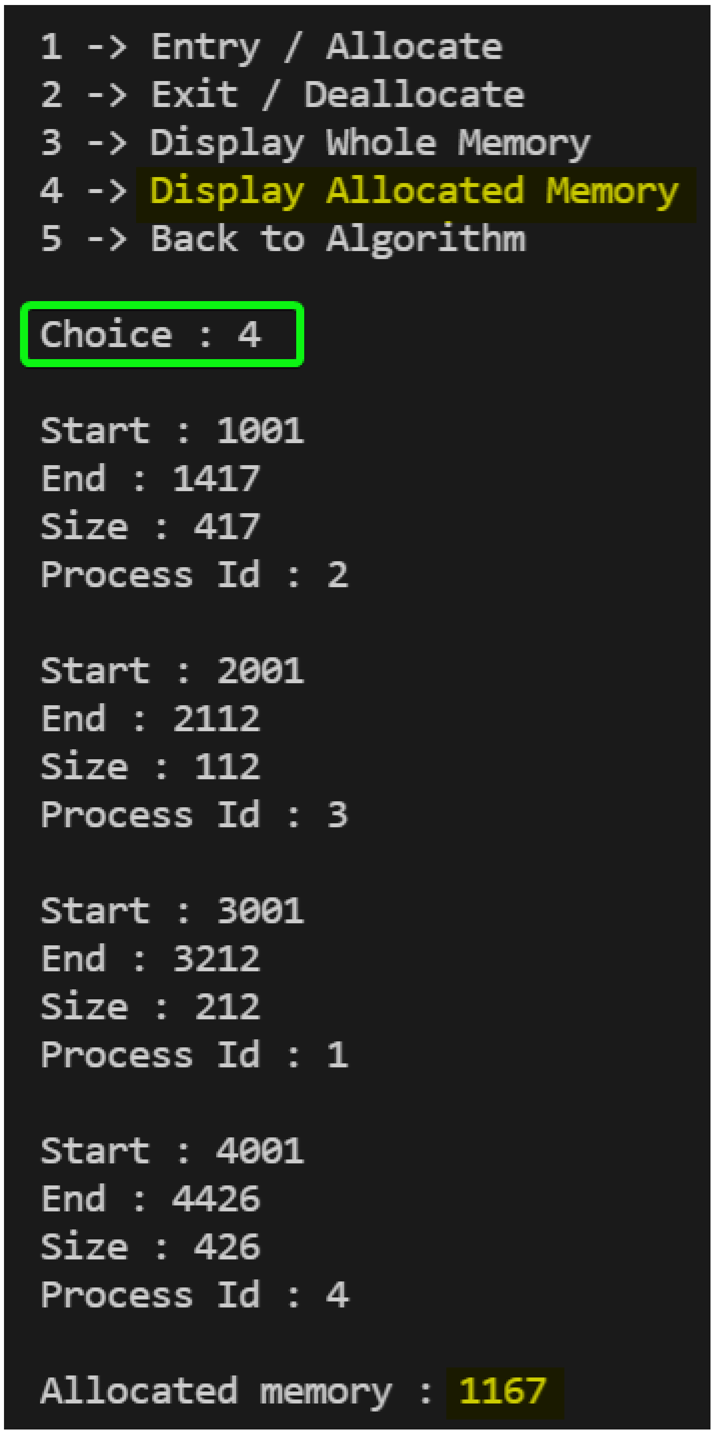


```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the process Id : 1
Enter required size by process : 212
Memory allocated succesfully!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the process Id : 2
Enter required size by process : 417
Memory allocated succesfully!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the process Id : 3
Enter required size by process : 112
Memory allocated succesfully!
```



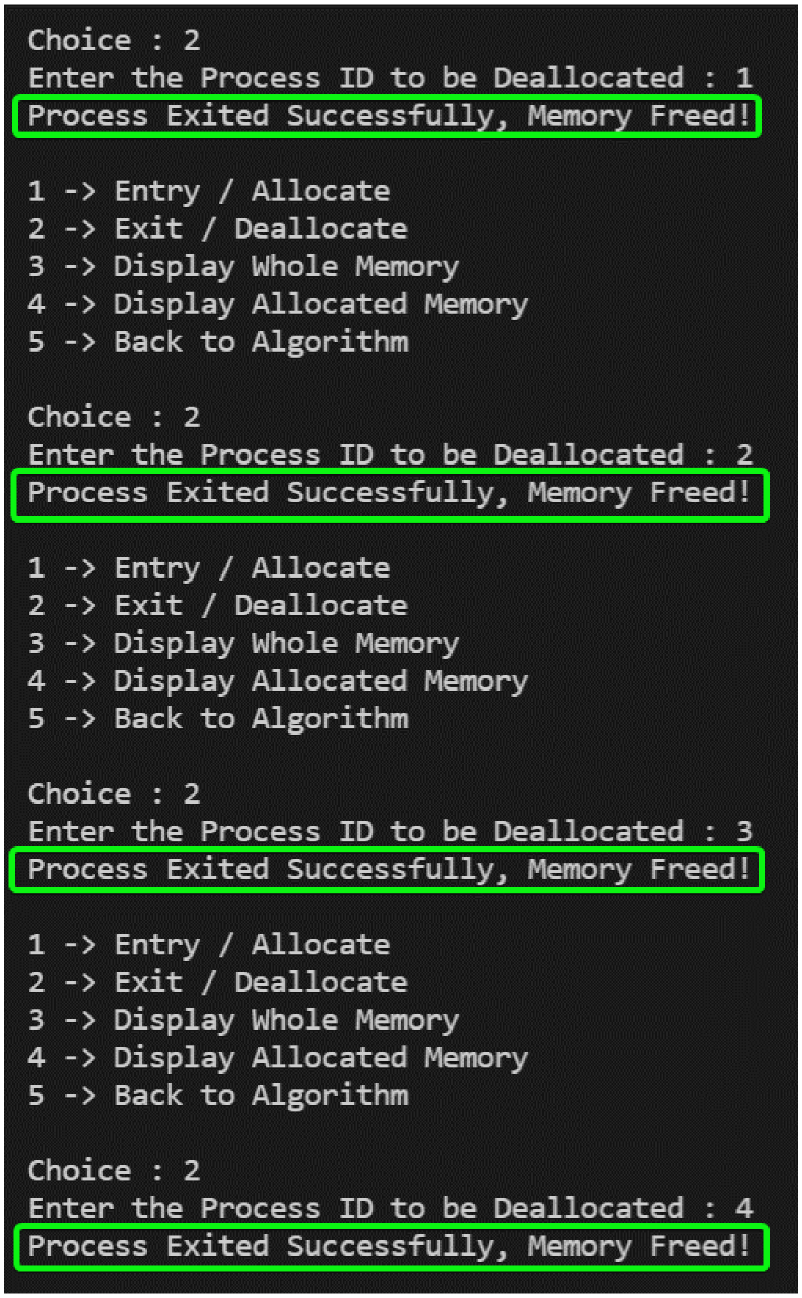
```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the process Id : 4
Enter required size by process : 426
Memory allocated succesfully!
```

The Allocated memory looks like:

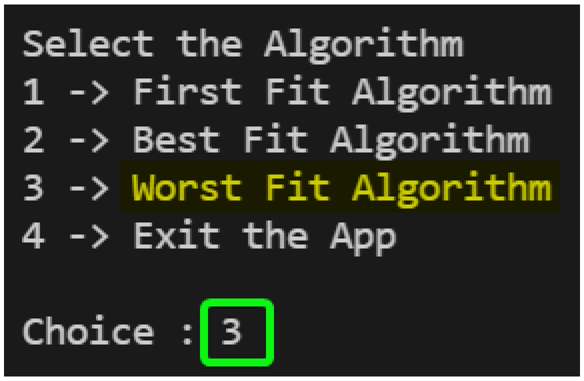


```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 4

Start : 1001
End : 1417
Size : 417
Process Id : 2

Start : 2001
End : 2112
Size : 112
Process Id : 3

Start : 3001
End : 3212
Size : 212
Process Id : 1

Start : 4001
End : 4426
Size : 426
Process Id : 4

Allocated memory : 1167
```

Let's De-allocate all memory and try Worst Fit Algorithm.

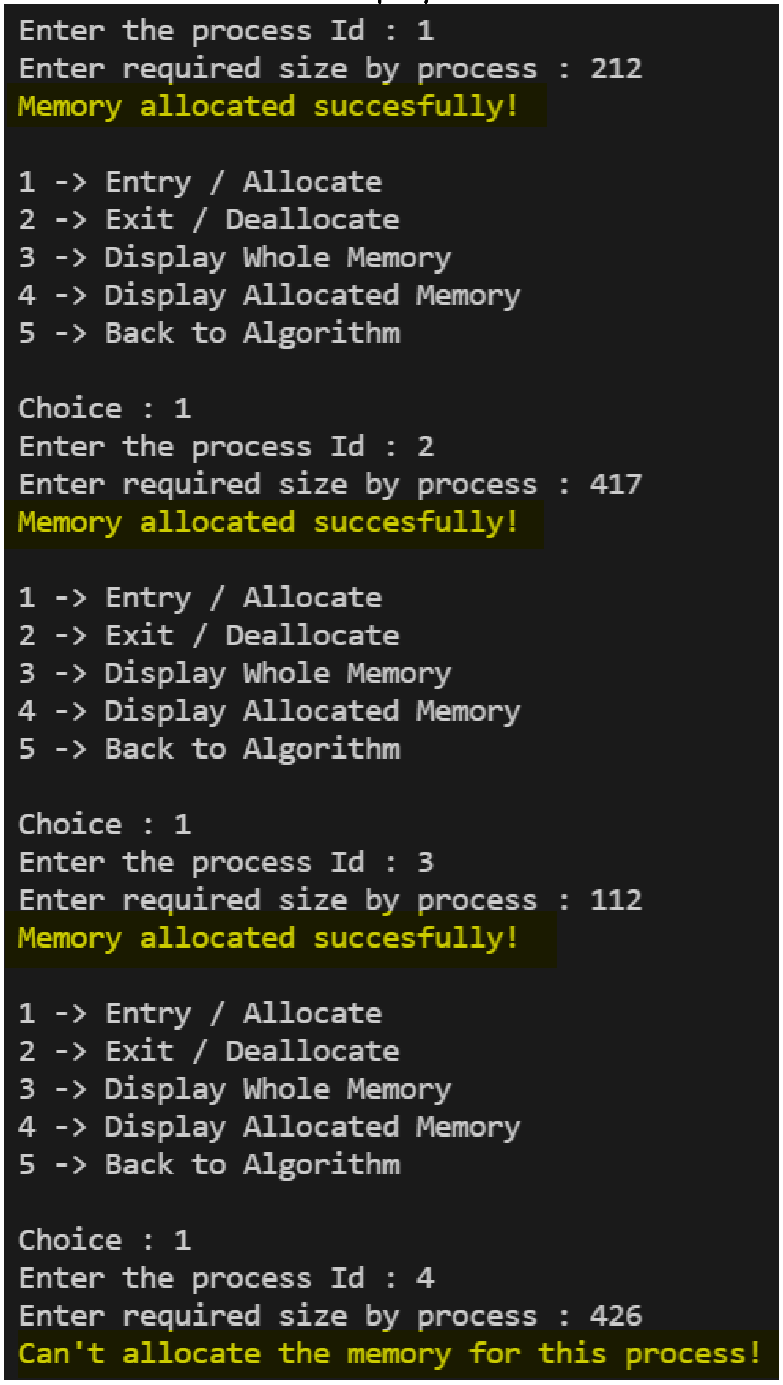

```
Choice : 2
Enter the Process ID to be Deallocated : 1
Process Exited Successfully, Memory Freed!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 2
Enter the Process ID to be Deallocated : 2
Process Exited Successfully, Memory Freed!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 2
Enter the Process ID to be Deallocated : 3
Process Exited Successfully, Memory Freed!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 2
Enter the Process ID to be Deallocated : 4
Process Exited Successfully, Memory Freed!
```

Worst Fit Algorithm Selected


```
Select the Algorithm
1 -> First Fit Algorithm
2 -> Best Fit Algorithm
3 -> Worst Fit Algorithm
4 -> Exit the App

Choice : 3
```

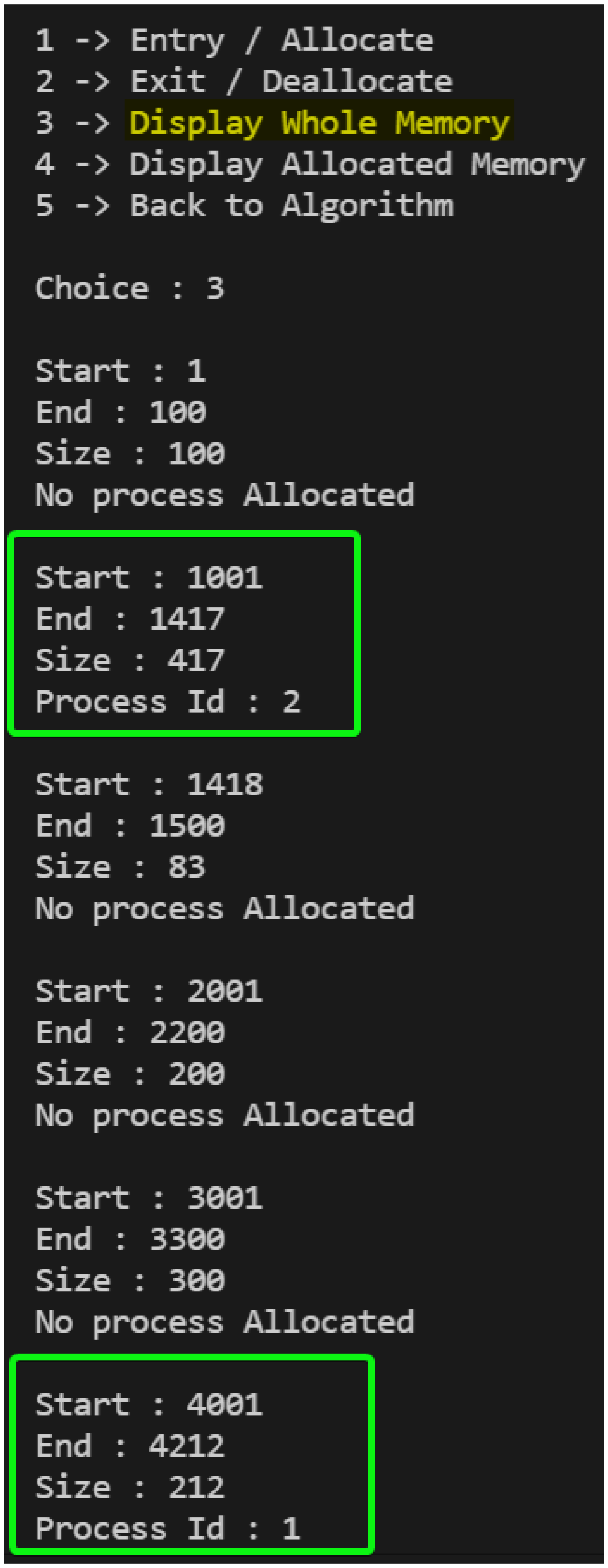
Required Size (in KB)	Partition Size (in KB) used	New Partition created (if any)
212	600	388KB partition created
417	500	83KB partition created
112	388	276KB partition created
426	-	-

We observe that 4th process **cannot** be allocated memory using Worst Fit Algorithm. Same is displayed below:

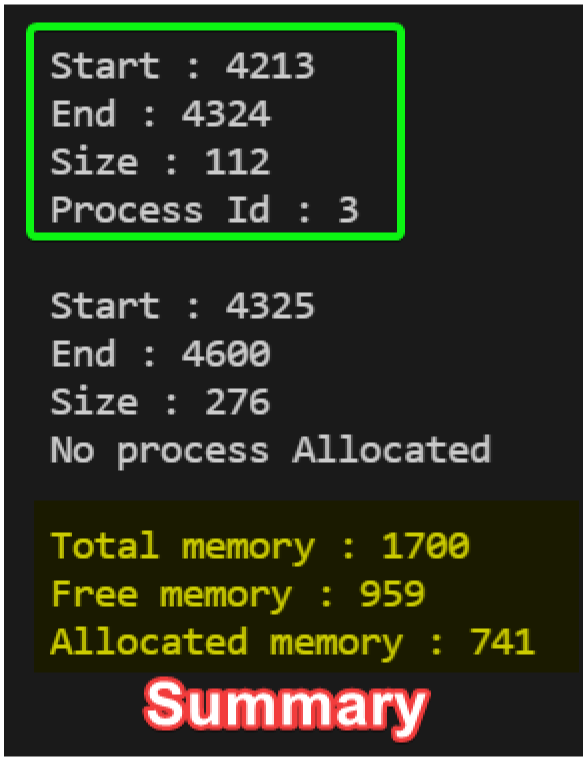


```
Enter the process Id : 1
Enter required size by process : 212
Memory allocated succesfully!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the process Id : 2
Enter required size by process : 417
Memory allocated succesfully!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the process Id : 3
Enter required size by process : 112
Memory allocated succesfully!

1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 1
Enter the process Id : 4
Enter required size by process : 426
Can't allocate the memory for this process!
```

Compete Memory Structure

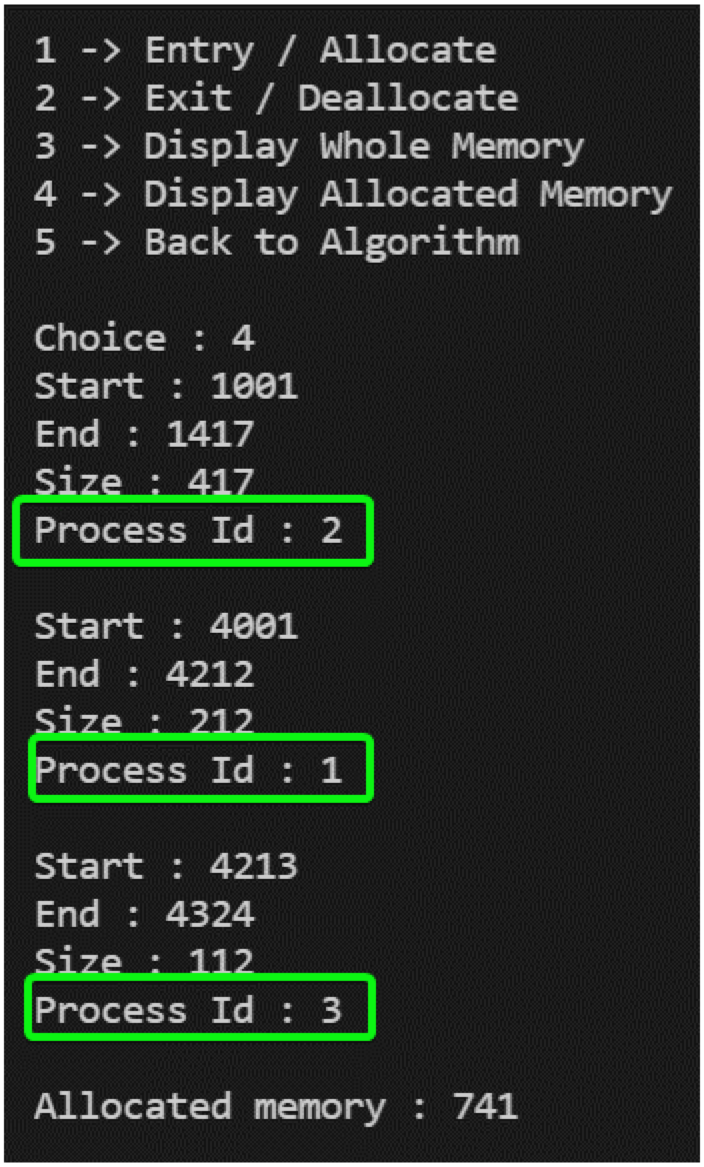


```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 3

Start : 1
End : 100
Size : 100
No process Allocated

Start : 1001
End : 1417
Size : 417
Process Id : 2

Start : 1418
End : 1500
Size : 83
No process Allocated

Start : 2001
End : 2200
Size : 200
No process Allocated

Start : 3001
End : 3300
Size : 300
No process Allocated

Start : 4001
End : 4212
Size : 212
Process Id : 1
```



```
Start : 4213
End : 4324
Size : 112
Process Id : 3

Start : 4325
End : 4600
Size : 276
No process Allocated

Total memory : 1700
Free memory : 959
Allocated memory : 741
```

Allocated Processes in Case of Worst Fit Algorithm



```
1 -> Entry / Allocate
2 -> Exit / Deallocate
3 -> Display Whole Memory
4 -> Display Allocated Memory
5 -> Back to Algorithm

Choice : 4
Start : 1001
End : 1417
Size : 417
Process Id : 2

Start : 4001
End : 4212
Size : 212
Process Id : 1

Start : 4213
End : 4324
Size : 112
Process Id : 3

Allocated memory : 741
```

- ✓ Therefore, we observe that our **Output in Tables [Expected]** is verified by our **Application [Program is Successful].**
- ✓ We also achieved our target of **De-allocating memory** and have taken care of merging contiguous free blocks into one single block.
- ✓ This result in Deleting more than one node from the free space list and changing the start and end address in the appropriate node.

2. Write a program that implements the following Page replacement algorithm.

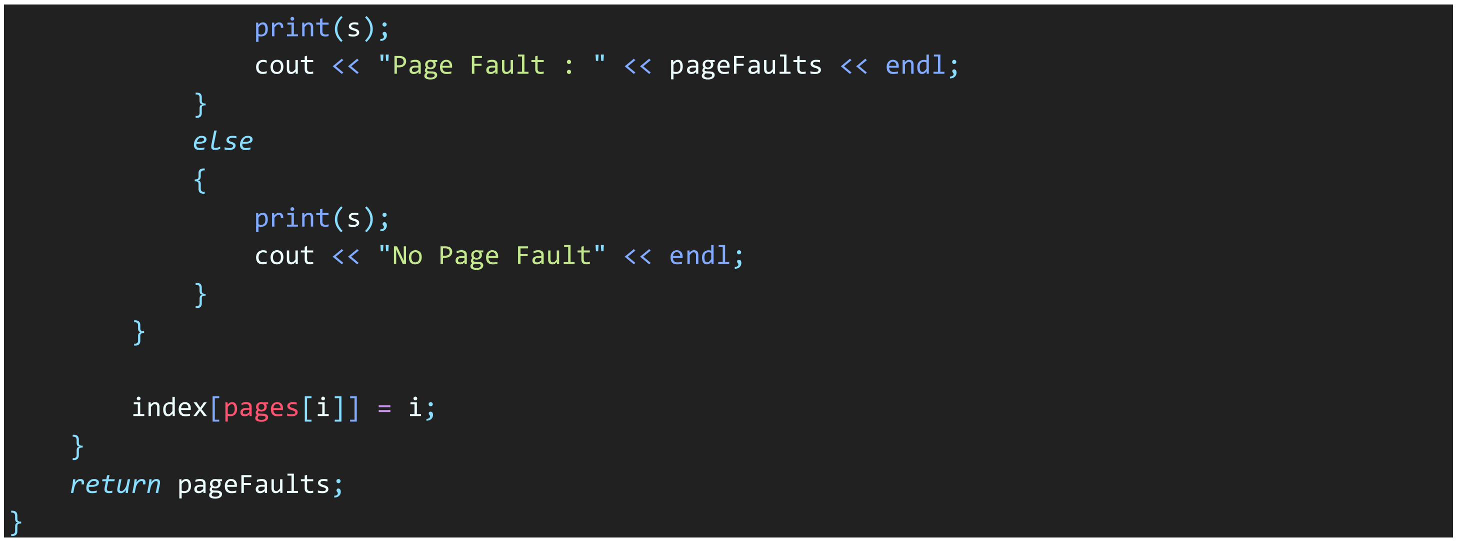
i) LRU (Least Recently Used)

In Least Recently Used (LRU) algorithm is a **Greedy algorithm** where the page to be replaced is least recently used. The idea is based on **locality of reference**, the least recently used page is not likely.

Code

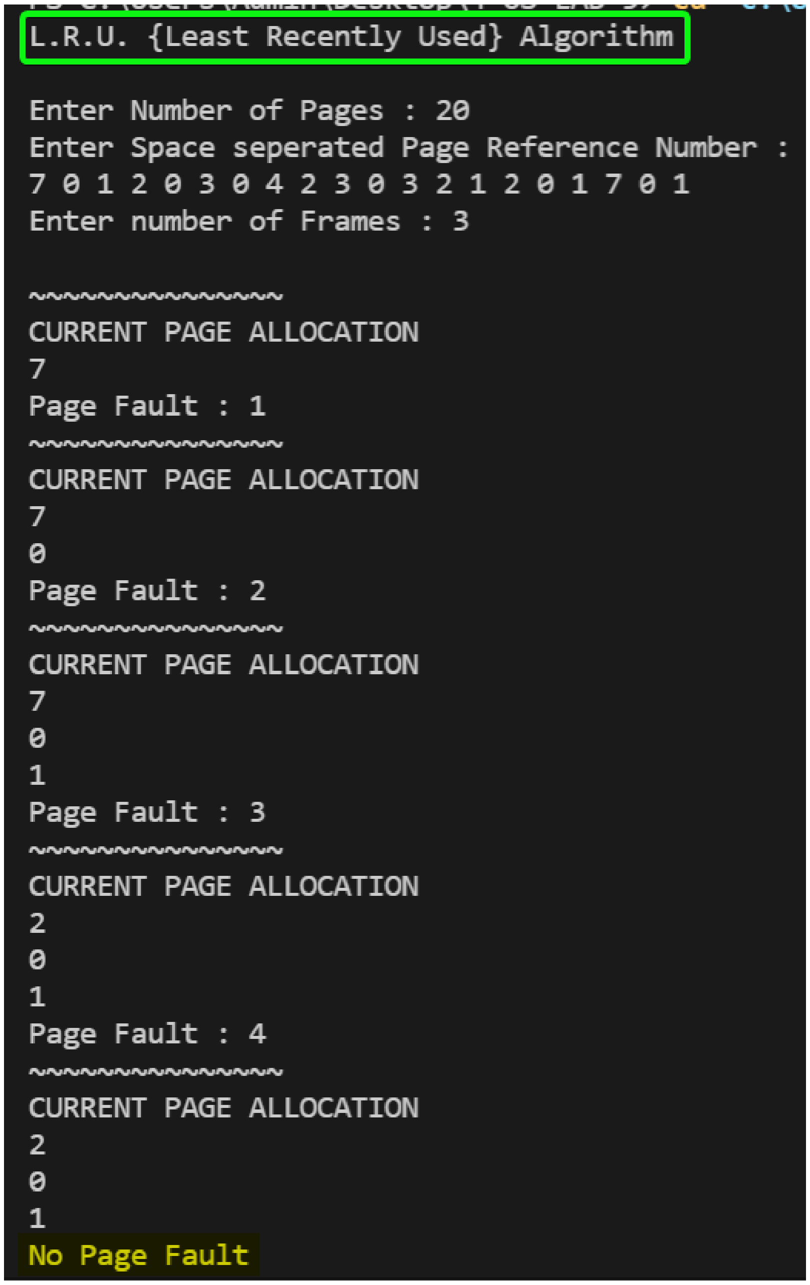
```cpp
#include <bits/stdc++.h>
using namespace std;

// F(x) to Print Vector
void print(vector<int> v);

// F(x) to Calculate the No Of Page Faults using LRU Algorithm
int LRU(int pages[], int n, int noFrames);

int main()
{
    cout << "L.R.U. {Least Recently Used} Algorithm\n\n";

    int n;
    cout << "Enter Number of Pages : ";
    cin >> n;

    int pages[n];
    cout << "Enter Space seperated Page Reference Number : \n";

    for (int i = 0; i < n; i++)
    {
        cin >> pages[i];
    }

    cout << "Enter number of Frames : ";
    int noFrames;
    cin >> noFrames;
    cout << endl;

    int ans = LRU(pages, n, noFrames);

    cout << "    " << endl;
    cout << "Total Page Faults : " << ans << endl;
}

// F(x) to Print Vector
void print(vector<int> v)
{
    cout << "~~~~~~~~~~~~~~~" << endl;
    cout << "CURRENT PAGE ALLOCATION" << endl;
    for (auto x : v)
```

```
        cout << x << endl;
};

// F(x) to Calculate the No Of Page Faults using LRU Algorithm
int LRU(int pages[], int n, int noFrames)
{
    vector<int> s;
    unordered_map<int, int> index;

    int pageFaults = 0;

    for (int i = 0; i < n; i++)
    {
        // If set holds less pages than capacity.
        if (s.size() < noFrames)
        {
            // If the Page is Not Found in Set
            if (find(s.begin(), s.end(), pages[i]) == s.end())
            {
                s.push_back(pages[i]);
                pageFaults++;
                print(s);
                cout << "Page Fault : " << pageFaults << endl;
            }
            // If [age is Found, No Page Fault Occurs
            else
            {
                print(s);
                cout << "No Page Fault" << endl;
            }
        }
        else
        {
            // Find the page in the set that was least recently used.

            // We find it using index array. We basically need to replace the page with minim
um index.
            if (find(s.begin(), s.end(), pages[i]) == s.end())
            {
                int lru = INT_MAX, val;
                for (int j = 0; j < s.size(); j++)
                {
                    if (index[s[j]] < lru)
                    {
                        lru = index[s[j]];
                        val = j;
                    }
                }
                s[val] = pages[i];
                pageFaults++;
```

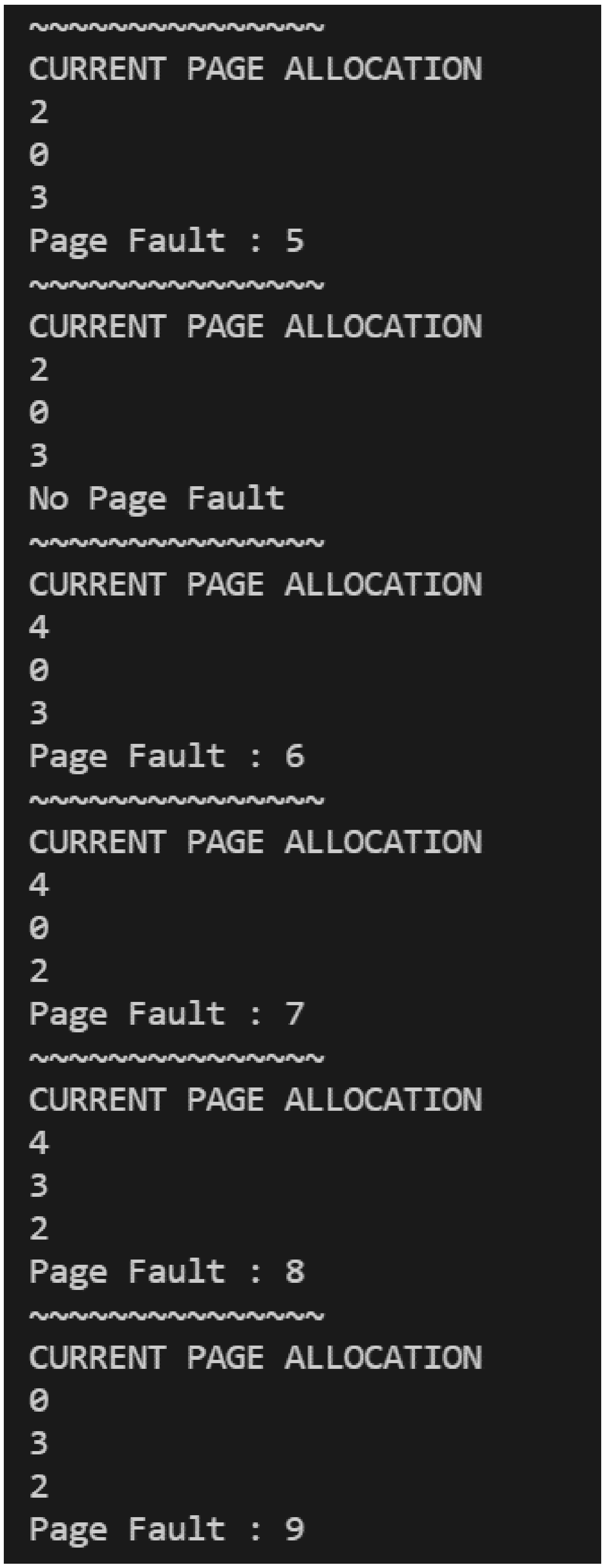


```cpp
                print(s);
                cout << "Page Fault : " << pageFaults << endl;
            }
            else
            {
                print(s);
                cout << "No Page Fault" << endl;
            }
        }

        index[pages[i]] = i;
    }
    return pageFaults;
}
```

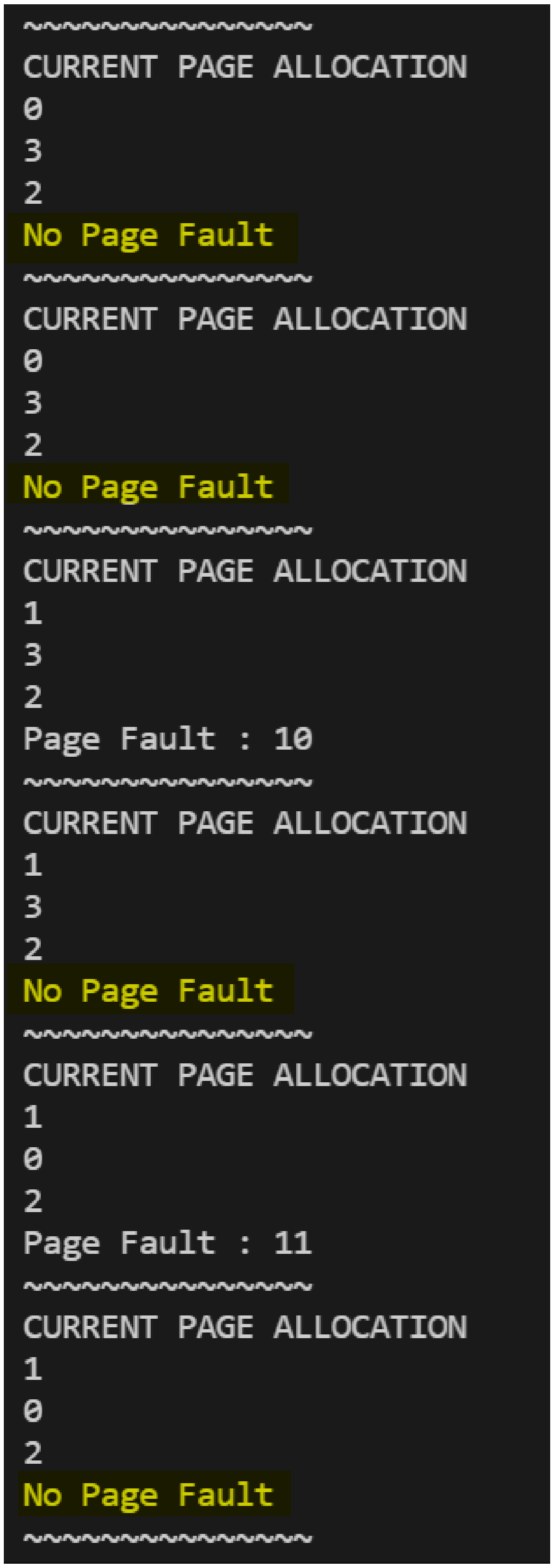
Output



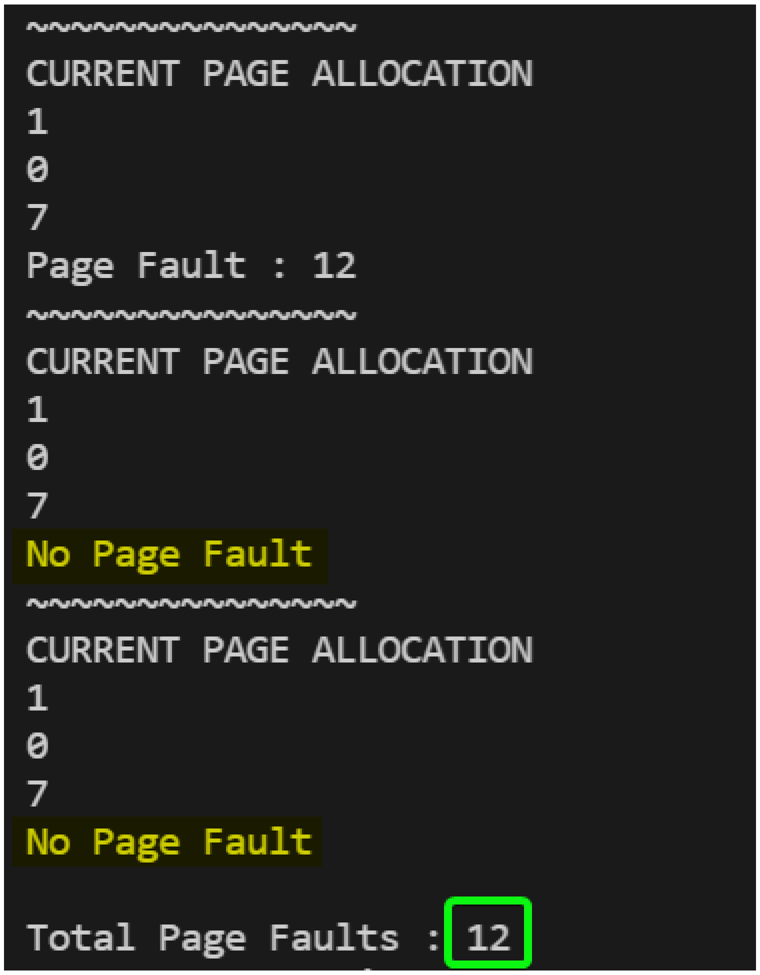
```
L.R.U. {Least Recently Used} Algorithm

Enter Number of Pages : 20
Enter Space seperated Page Reference Number :
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of Frames : 3

~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
Page Fault : 1
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
0
Page Fault : 2
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
0
1
Page Fault : 3
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
1
Page Fault : 4
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
1
No Page Fault
```



```
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
3
Page Fault : 5
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
3
No Page Fault
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
4
0
3
Page Fault : 6
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
4
0
2
Page Fault : 7
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
4
3
2
Page Fault : 8
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
0
3
2
Page Fault : 9
```



```
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
0
3
2
No Page Fault
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
0
3
2
No Page Fault
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
1
3
2
Page Fault : 10
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
1
3
2
No Page Fault
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
1
0
2
Page Fault : 11
~~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
1
0
2
No Page Fault
~~~~~~~~~~~~~~~~~~~
```



```
~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
1
0
7
Page Fault : 12
~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
1
0
7
No Page Fault
~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
1
0
7
No Page Fault

Total Page Faults : 12
```

ii) Optimal Page Replacement algorithm

In this algorithm, OS replaces the page that will **not be used** for the longest period of time in future.

Code

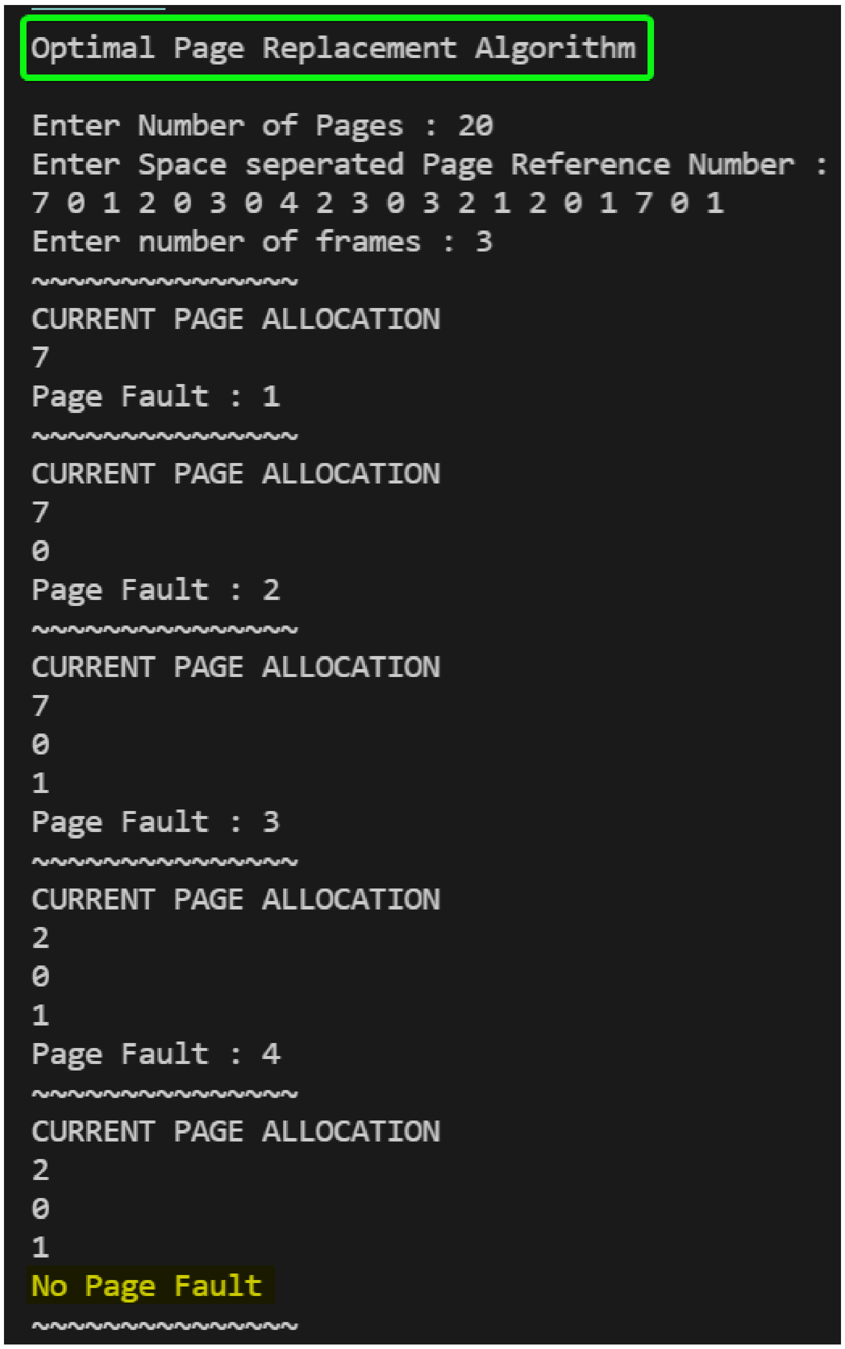
```cpp
#include <bits/stdc++.h>
using namespace std;

// F(x) to Print Vector
void print(vector<int> v);

// F(x) to Predict the Optimal Page that Would be Required in Future
int predictFuture(int pages[], vector<int> frames, int n, int index);

// F(x) to Calculate the No Of Page Faults using Optimal Algorithm
int OptimalPageReplacement(int pages[], int n, int noFrames);

int main()
{
    cout << "Optimal Page Replacement Algorithm\n\n";
```

```cpp
    int n;
    cout << "Enter Number of Pages : ";
    cin >> n;

    int pages[n];
    cout << "Enter Space seperated Page Reference Number : \n";

    for (int i = 0; i < n; i++)
        cin >> pages[i];

    cout << "Enter number of frames : ";
    int noFrames;
    cin >> noFrames;

    int ans = OptimalPageReplacement(pages, n, noFrames);

    cout << "   " << endl;
    cout << "Total Page Faults : " << ans << endl;

    return 0;
}

// F(x) to Print Vector
void print(vector<int> v)
{
    cout << "~~~~~~~~~~~~~~~" << endl;
    cout << "CURRENT PAGE ALLOCATION" << endl;
    for (auto x : v)
        cout << x << endl;
};

// F(x) to Predict the Optimal Page that Would be Required in Future
int predictFuture(int pages[], vector<int> frames, int n, int index)
{
    int res = -1;
    int far = index;

    for (int i = 0; i < frames.size(); i++)
    {
        int j;
        for (j = index; j < n; j++)
        {
            if (frames[i] == pages[j])
            {
                if (j > far)
                {
                    far = j;
                    res = i;
                }
```

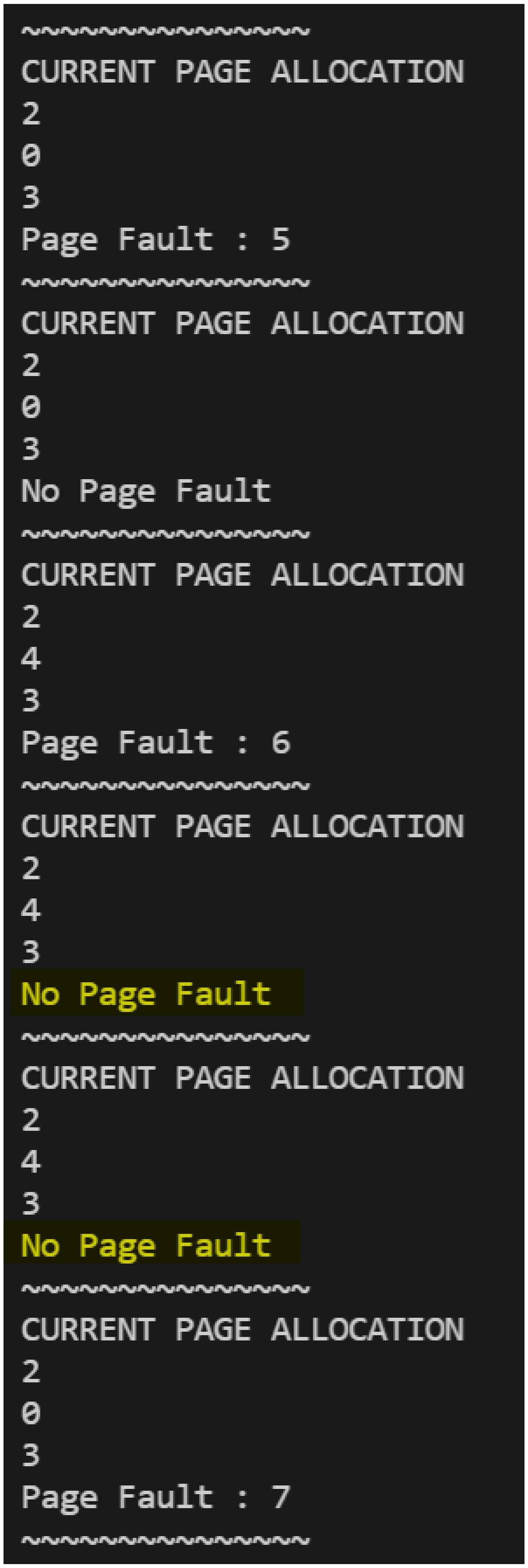
```
                break;
            }
        }
        if (j == n)
        {
            return i;
        }
    }
    if (res == -1)
    {
        return 0;
    }
    return res;
}

// F(x) to Calculate the No Of Page Faults using Optimal Algorithm
int OptimalPageReplacement(int pages[], int n, int noFrames)
{
    vector<int> frames;
    int pageFaults = 0;

    for (int i = 0; i < n; i++)
    {
        // If the Page is Found
        if (find(frames.begin(), frames.end(), pages[i]) != frames.end())
        {
            print(frames);
            cout << "No Page Fault" << endl;
        }
        // If set holds less pages than capacity.
        else if (frames.size() < noFrames)
        {
            frames.push_back(pages[i]);
            pageFaults++;
            print(frames);

            cout << "Page Fault : " << pageFaults << endl;
        }
        // Find the Optimal Page to be Replaced
        else
        {
            int index = predictFuture(pages, frames, n, i + 1);
            frames[index] = pages[i];
            pageFaults++;
            print(frames);
            cout << "Page Fault : " << pageFaults << endl;
        }
    }

    return pageFaults;
```

```
}
```

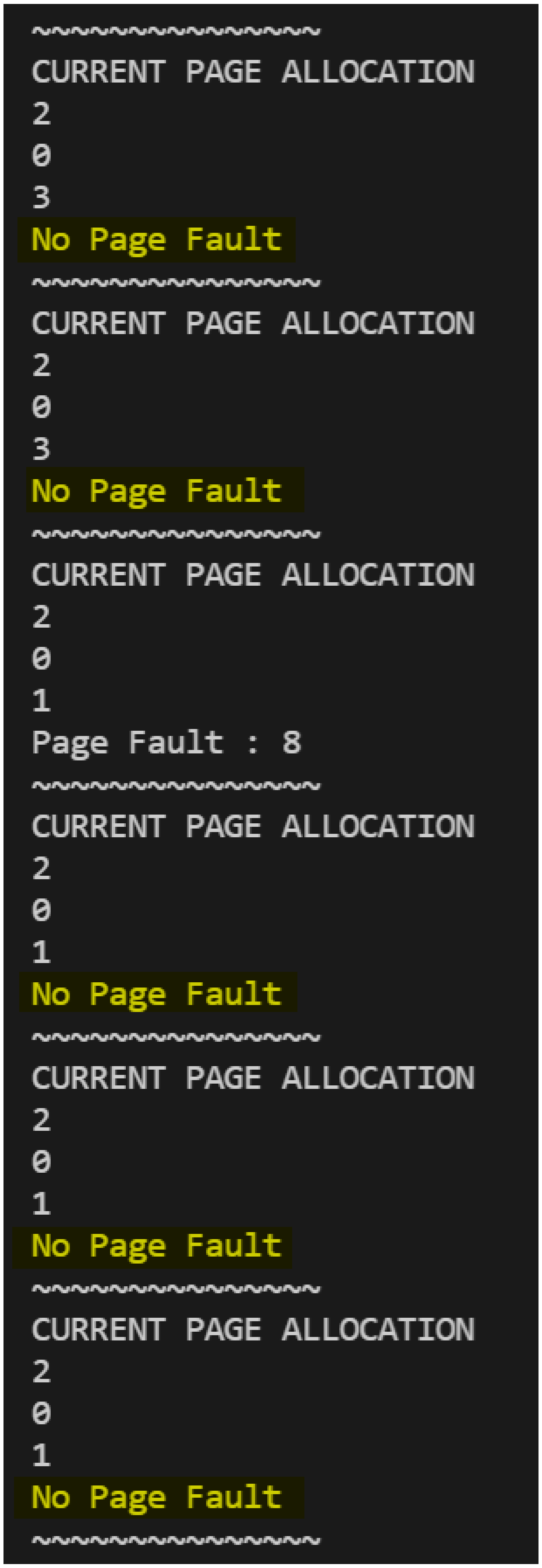
Output



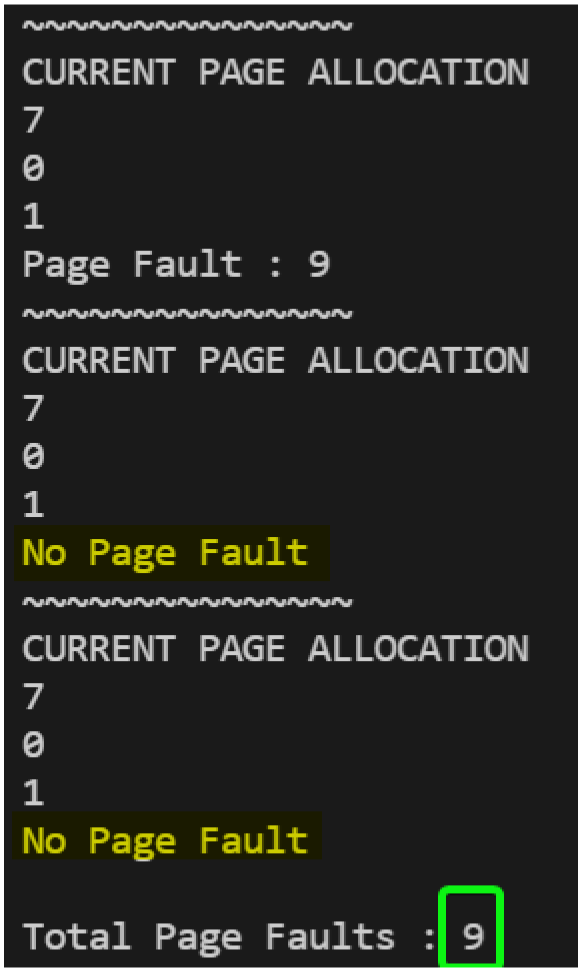
```
Optimal Page Replacement Algorithm

Enter Number of Pages : 20
Enter Space seperated Page Reference Number :
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames : 3
~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
Page Fault : 1
~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
0
Page Fault : 2
~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
0
1
Page Fault : 3
~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
1
Page Fault : 4
~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
1
No Page Fault
~~~~~~~~~~~~~~~
```



```
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
3
Page Fault : 5
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
3
No Page Fault
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
4
3
Page Fault : 6
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
4
3
No Page Fault
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
4
3
No Page Fault
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
3
Page Fault : 7
~~~~~~~~~~~~~~~~~
```



```
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
3
No Page Fault
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
3
No Page Fault
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
1
Page Fault : 8
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
1
No Page Fault
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
1
No Page Fault
~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
2
0
1
No Page Fault
~~~~~~~~~~~~~~~~~
```



```
~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
0
1
Page Fault : 9
~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
0
1
No Page Fault
~~~~~~~~~~~~~~~~~~
CURRENT PAGE ALLOCATION
7
0
1
No Page Fault

Total Page Faults : 9
```

Input Page Request	7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 (20 Pages)	
No. of frames	3	
Page Faults	**12** (LRU Algorithm)	**9** (Optimal Page Replacement Algorithm)

<u>SUBMITTED BY:</u>

U19CS012

BHAGYA VINOD RANA