Lab No. 4 **Date: 2025/08/14**

Title: Write a program to display the process allocation for user input free blocks and incoming process using Best Fit Allocation.

Memory allocation is the process of assigning memory blocks to processes from the available free memory space. When multiple free blocks are available, different strategies can be used to decide which block to allocate to a process.

The Best Fit Allocation Algorithm searches for the smallest available block that is large enough to accommodate the process. It helps to minimize unused memory (internal fragmentation) by using the most appropriately sized block. However, it can lead to external fragmentation because small leftover blocks may not be usable later.

Steps in Best Fit Allocation:

a. For each process, scan the list of free memory blocks.

b. Choose the smallest block that is greater than or equal to the process size.

c. Allocate the process to that block and reduce the available block size.

d. If no suitable block is found, the process remains unallocated.

Programming Language: C

IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;

void bestFit(int blockSize[], int originalBlockSize[], int m, int processSize[], int n)
{
  int allocation[n];    // Stores block index for each process (-1 if not allocated)
  int holeAfterAlloc[n]; // Stores hole size immediately after allocation
  // Initialize arrays
  for (int i = 0; i < n; i++)
  {
    allocation[i] = -1;
    holeAfterAlloc[i] = -1;
  }
  // Allocate processes
  for (int i = 0; i < n; i++)
  {
    int bestIdx = -1;
    for (int j = 0; j < m; j++)
    {
      if (blockSize[j] >= processSize[i])
      {
        if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])
          bestIdx = j;
      }
    }
    if (bestIdx != -1)
    {
      allocation[i] = bestIdx;
      blockSize[bestIdx] -= processSize[i];
      holeAfterAlloc[i] = blockSize[bestIdx]; // store immediate hole
    }
  }
  // Showing output results
  cout << "\nProcess No.\tProcess Size\tBlock No.\tBlock Size\tHole Size\n";

  for (int i = 0; i < n; i++)
  {
    cout << setw(6) << i + 1 << "\t\t"
       << setw(6) << processSize[i] << "\t\t";
    if (allocation[i] != -1)
    {
      int b = allocation[i];
      cout << setw(5) << b + 1 << "\t\t" << setw(6) << originalBlockSize[b] << "\t\t" << setw(6) <<
    holeAfterAlloc[i] << "\n";
    }
    else
    {
      cout << setw(5) << "NA" << "\t\t" << setw(6) << "-" << "\t\t" << setw(6) << "-" << "\n";
    }
```

```
    }
  }

  int main()
  {
    cout << "Compiled by:- Jonash Chataut\n";
    cout << "<---- Best fit allocation ---->\n";
    int m, n;
    cout << "Enter number of memory blocks: ";
    cin >> m;
    int blockSize[m], originalBlockSize[m];
    cout << "Enter sizes of the " << m << " blocks:\n";
    for (int i = 0; i < m; i++)
    {
      cin >> blockSize[i];
      originalBlockSize[i] = blockSize[i];
    }
    cout << "Enter number of processes: ";
    cin >> n;
    int processSize[n];
    cout << "Enter sizes of the " << n << " processes:\n";
    for (int i = 0; i < n; i++)
      cin >> processSize[i];
    bestFit(blockSize, originalBlockSize, m, processSize, n);
    return 0;
  }
```

Output:

```
C:\csit\fourth_sem_jonash\OS
Compiled by:- Jonash Chataut
<---- Best fit allocation ---->
Enter number of memory blocks: 5
Enter sizes of the 5 blocks:
100 500 200 300 600
Enter number of processes: 4
Enter sizes of the 4 processes:
212 417 112 426

Process No.     Process Size    Block No.       Block Size      Hole Size
     1              212             4               300             88
     2              417             2               500             83
     3              112             3               200             88
     4              426             5               600            174

--------------------------------
Process exited after 21.16 seconds with return value 0
Press any key to continue . . .
```

Lab No. 5 **Date: 2025/08/14**

Title: Write a program to display the process allocation for user input free blocks and incoming process using Worst Fit Allocation.

Worst Fit is a dynamic memory allocation strategy used in operating systems for assigning free memory blocks to processes. In this method, the largest available memory block is allocated to the process requesting memory. The Worst Fit Allocation Algorithm assigns a process to the largest available memory block that is big enough to hold it. The idea is to leave larger leftover holes, which might fit future processes more effectively than small leftover fragments.

Algorithm Steps

For each process:

1. Search all available free memory blocks.
2. Select the block with the largest size that is greater than or equal to the process size.
3. Allocate the process to that block.
4. Reduce the size of that block (update the hole size).
5. If no block can fit the process, it remains unallocated.

Programming Language: C

IDE: Dev C++

Source Code:

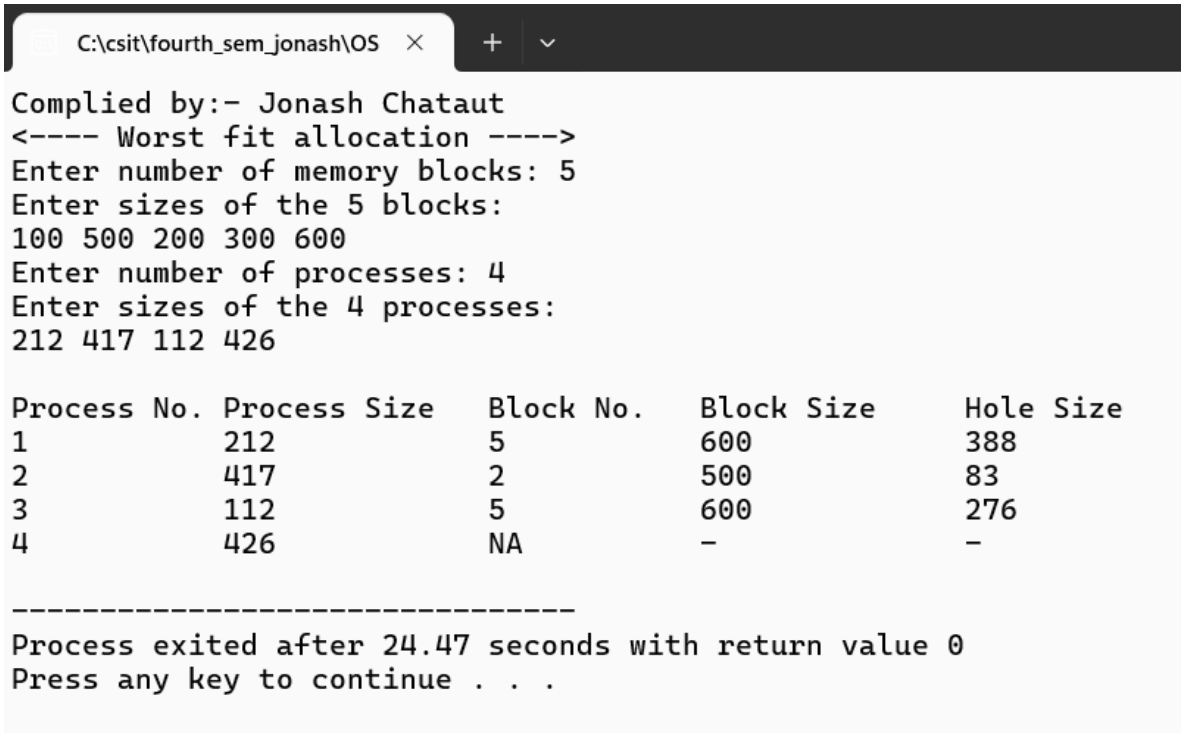
```
#include <iostream>
#include <iomanip>
using namespace std;

void worstFit(int blockSize[], int originalBlockSize[], int m, int processSize[], int n)
{
   int allocation[n];    // Stores block index for each process (-1 if not allocated)
   int holeAfterAlloc[n];  // Stores hole size right after allocation
   // Initialize arrays
   for (int i = 0; i < n; i++)
   {
      allocation[i] = -1;
      holeAfterAlloc[i] = -1;
   }
   // Allocate processes
   for (int i = 0; i < n; i++)
   {
      int worstIdx = -1;
      for (int j = 0; j < m; j++)
      {
         if (blockSize[j] >= processSize[i])
         {
            if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])
               worstIdx = j;
         }
      }
      if (worstIdx != -1)
      {
         allocation[i] = worstIdx;
         blockSize[worstIdx] -= processSize[i];
         holeAfterAlloc[i] = blockSize[worstIdx]; // store hole size immediately
      }
   }
   // Output results
   cout << "\n"
      << left << setw(12) << "Process No." << setw(15) << "Process Size" << setw(12) << "Block No." <<
     setw(15) << "Block Size" << "Hole Size\n";
   for (int i = 0; i < n; i++)
   {
      cout << left << setw(12) << i + 1
         << setw(15) << processSize[i];
      if (allocation[i] != -1)
      {
         int b = allocation[i];
         cout << setw(12) << b + 1 << setw(15) << originalBlockSize[b] << holeAfterAlloc[i] << "\n";
      }
      else
      {
         cout << setw(12) << "NA" << setw(15) << "-" << "-\n";
      }
```

```
    }
}

int main()
{
    cout << "Complied by:- Jonash Chataut\n";
    cout << "<---- Worst fit allocation ---->\n";
    int m, n;
    cout << "Enter number of memory blocks: ";
    cin >> m;
    int blockSize[m], originalBlockSize[m];
    cout << "Enter sizes of the " << m << " blocks:\n";
    for (int i = 0; i < m; i++)
    {
        cin >> blockSize[i];
        originalBlockSize[i] = blockSize[i];
    }
    cout << "Enter number of processes: ";
    cin >> n;
    int processSize[n];
    cout << "Enter sizes of the " << n << " processes:\n";
    for (int i = 0; i < n; i++)
        cin >> processSize[i];
    worstFit(blockSize, originalBlockSize, m, processSize, n);
    return 0;
}
```

Output:



Madan Bhandari Memorial College

Lab No. 6 **Date: 2025/08/14**

Title: Write a program to display the process allocation for user input free blocks and incoming process using First Fit Allocation.

First fit allocation is a dynamic memory allocation technique in operating systems where a process is allocated to the first available memory block that is large enough to accommodate it. It is simple and quick but may lead to fragmentation.

Algorithm:

a. The operating system maintains a list of free memory blocks (holes).
b. When a process arrives, the OS searches the list from the beginning.
c. The process is allocated to the first block that is large enough.
d. If the block is larger than the process, the remaining portion of the block becomes a hole.
e. If no block is big enough, the process cannot be allocated (it must wait or be rejected).

Programming Language: C++

IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;

void firstFit(int blockSize[], int originalBlockSize[], int m, int processSize[], int n)
{
  int allocation[n];    // Stores block index for each process (-1 if not allocated)
  int holeAfterAlloc[n]; // Stores hole size immediately after allocation

  // Initialize arrays
  for (int i = 0; i < n; i++)
  {
    allocation[i] = -1;
    holeAfterAlloc[i] = -1;
  }

  // Allocate processes
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++) // First Fit: allocate in first suitable block
    {
      if (blockSize[j] >= processSize[i])
      {
        allocation[i] = j;
        blockSize[j] -= processSize[i];
        holeAfterAlloc[i] = blockSize[j];
        break; // Move to next process
      }
    }
  }

  // Showing output results
  cout << "\nProcess No.\tProcess Size\tBlock No.\tBlock Size\tHole Size\n";
  for (int i = 0; i < n; i++)
  {
    cout << setw(6) << i + 1 << "\t\t"
       << setw(6) << processSize[i] << "\t\t";
    if (allocation[i] != -1)
    {
      int b = allocation[i];
      cout << setw(5) << b + 1 << "\t\t"
         << setw(6) << originalBlockSize[b] << "\t\t"
         << setw(6) << holeAfterAlloc[i] << "\n";
    }
    else
    {
      cout << setw(5) << "NA" << "\t\t" << setw(6) << "-" << "\t\t"<< setw(6) << "-" << "\n";
    }
  }
}
```

```
int main()
{
   cout << "Compiled by:- Jonash Chataut\n";
   cout << "<---- First fit allocation ---->\n";
   int m, n;
   cout << "Enter number of memory blocks: ";
   cin >> m;

   int blockSize[m], originalBlockSize[m];
   cout << "Enter sizes of the " << m << " blocks:\n";
   for (int i = 0; i < m; i++)
   {
      cin >> blockSize[i];
      originalBlockSize[i] = blockSize[i];
   }

   cout << "Enter number of processes: ";
   cin >> n;
   int processSize[n];
   cout << "Enter sizes of the " << n << " processes:\n";
   for (int i = 0; i < n; i++)
      cin >> processSize[i];
   firstFit(blockSize, originalBlockSize, m, processSize, n);
   return 0;
}
```

Output: