Title: Contiguous Memory Allocation

Aim:
Write a program to Simulate Dynamic memory allocation algorithms - First-fit, Best-fit, Worst-fit algorithms.

Procedure:
First Fit:
1. Start from the first memory block and allocate it to the process if it is large enough.
2. Move to the next memory block and repeat step 1 until a block is found that can accommodate the process or all blocks have been checked.
3. Repeat steps 1-2 for all processes.

Best Fit:
1. Find the smallest memory block that is large enough to accommodate the process.
2. Allocate the process to that memory block.
3. Repeat steps 1-2 for all processes.

Worst Fit:
1. Find the largest memory block that is large enough to accommodate the process.
2. Allocate the process to that memory block.
3. Repeat steps 1-2 for all processes.

Algorithm:
1. Start
2. Define the number of processes.
3. Define the number of memory blocks.

4. Create first_fit function.
   a. Initialize an Array to store the allocation of memory blocks to processes. Create an array called allocation [] to store the allocation status of each process.
   b. Initialize all allocations as -1 (not allocated).
   c. Loop through all processes, Loop through all memory blocks,
   d. If the memory block is large enough to accommodate the process
   e. Allocate the memory block to the process.
   f. Reduce the size of the memory block by the size of the process.
   g. Break out of the inner loop (memory block loop)
   h. Print the header for the table.
   i. Loop through all processes
   j. Print the process number and size.
   k. If the process was allocated a memory block, then Print the block number else print "Not Allocated".

5. Create Best_fit function.
   a. Initialize an Array to store the allocation of memory blocks to processes. Create an array called allocation [] to store the allocation status of each process.

   b. Loop through all processes
   c. Define the Variable to store the index of the smallest memory block that can accommodate the process
   d. Loop through all memory blocks
   e. If the memory block is large enough to accommodate the process
   f. If this is the first suitable memory block found then Set jmin (variable) to its index else if this memory block is smaller than the current smallest suitable block Set jmin to its index.
   g. If a suitable memory block was found Allocate it to the process
   h. Reduce its size by the size of the process
   i. Print the header for the table
   j. Loop through all processes
   k. Print the process number and size
   l. If the process was allocated a memory block then Print the block number else print "Not Allocated”.

6. Create Worst Fit function.
   a. Initialize an Array to store the allocation of memory blocks to processes. Create an array called allocation [] to store the allocation status of each process.
   b. Loop through all processes
   c. Define the Variable to store the index of the largest memory block that can accommodate the process.
   d. Loop through all blocks
   e. If this block is large enough and If this is first suitable block set jmax = j.
   f. If this block is larger than current largest suitable block set jmax = j.
   g. If a suitable block was found Allocate it to the process and Reduce its size by the size of the process
   h. Print the header for the table
   i. Loop through all processes
   j. Print the process number and size
   k. If the process was allocated a memory block then Print the block number else print "Not Allocated”.

7. Main Program
   a. Initialize Array to store the sizes of the memory blocks.
   b. Array to store the sizes of the processes.
   c. Call the first_fit function
   d. Create a copy of the memory array for use in best_fit
   e. Call the best_fit function
   f. Create a copy of the memory array for use in worst_fit
   g. Call the worst_fit function.

8. Stop

Programs (Not as Per above Procedure)

//FIRST FIT

```
#include <stdio.h>

void implimentFirstFit(int blockSize[], int blocks, int processSize[], int processes)
{
  // This will store the block id of the allocated block to a process
  int allocate[processes];
  int occupied[blocks];

  // initially assigning -1 to all allocation indexes
  // means nothing is allocated currently
  for(int i = 0; i < processes; i++)
        {
                allocate[i] = -1;
        }

        for(int i = 0; i < blocks; i++){
    occupied[i] = 0;
  }

  // take each process one by one and find
  // first block that can accomodate it
  for (int i = 0; i < processes; i++)
  {
    for (int j = 0; j < blocks; j++)
    {
    if (!occupied[j] && blockSize[j] >= processSize[i])
      {
        // allocate block j to p[i] process
        allocate[i] = j;
        occupied[j] = 1;

        break;
      }
    }
  }

  printf("\nProcess No.\tProcess Size\tBlock no.\n");
  for (int i = 0; i < processes; i++)
  {
    printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
    if (allocate[i] != -1)
      printf("%d\n",allocate[i] + 1);
    else
      printf("Not Allocated\n");
  }
}

void main()
{
  int blockSize[] = {30, 5, 10};
  int processSize[] = {10, 6, 9};
  int m = sizeof(blockSize)/sizeof(blockSize[0]);
  int n = sizeof(processSize)/sizeof(processSize[0]);

  implimentFirstFit(blockSize, m, processSize, n);
}

//BEST FIT
#include <stdio.h>

void implimentBestFit(int blockSize[], int blocks, int processSize[], int proccesses)
{
  // This will store the block id of the allocated block to a process
  int allocation[proccesses];
  int occupied[blocks];

  // initially assigning -1 to all allocation indexes
  // means nothing is allocated currently
  for(int i = 0; i < proccesses; i++){
    allocation[i] = -1;
  }

  for(int i = 0; i < blocks; i++){
    occupied[i] = 0;
  }

  // pick each process and find suitable blocks
  // according to its size ad assign to it
  for (int i = 0; i < proccesses; i++)
  {

    int indexPlaced = -1;
    for (int j = 0; j < blocks; j++) {
      if (blockSize[j] >= processSize[i] && !occupied[j])
      {
        // place it at the first block fit to accomodate process
        if (indexPlaced == -1)
          indexPlaced = j;

        // if any future block is smalller than the current block where
        // process is placed, change the block and thus indexPlaced
            // this reduces the wastage thus best fit
        else if (blockSize[j] < blockSize[indexPlaced])
          indexPlaced = j;
      }
    }

    // If we were successfully able to find block for the process
    if (indexPlaced != -1)
    {
      // allocate this block j to process p[i]
      allocation[i] = indexPlaced;

      // make the status of the block as occupied
      occupied[indexPlaced] = 1;
      }
   }

   printf("\nProcess No.\tProcess Size\tBlock no.\n");
   for (int i = 0; i < proccesses; i++)
   {
      printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
      if (allocation[i] != -1)
         printf("%d\n",allocation[i] + 1);
      else
         printf("Not Allocated\n");
   }
}

// Driver code
int main()
{
   int blockSize[] = {100, 50, 30, 120, 35};
   int processSize[] = {40, 10, 30, 60};
   int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
   int proccesses = sizeof(processSize)/sizeof(processSize[0]);

   implimentBestFit(blockSize, blocks, processSize, proccesses);

   return 0 ;
}
```

//WORST FIT

```
#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int
processSize[], int processes)
{
   // This will store the block id of the allocated block to a process
   int allocation[processes];
   int occupied[blocks];

   // initially assigning -1 to all allocation indexes
   // means nothing is allocated currently
   for(int i = 0; i < processes; i++){
      allocation[i] = -1;
   }

   for(int i = 0; i < blocks; i++){
      occupied[i] = 0;
   }

   // pick each process and find suitable blocks
   // according to its size ad assign to it
   for (int i=0; i < processes; i++)
   {
         int indexPlaced = -1;
         for(int j = 0; j < blocks; j++)
         {
            // if not occupied and block size is large enough
            if(blockSize[j] >= processSize[i] && !occupied[j])
        {
            // place it at the first block fit to accomodate process
            if (indexPlaced == -1)
                indexPlaced = j;

            // if any future block is larger than the current block
where
            // process is placed, change the block and thus
indexPlaced
            else if (blockSize[indexPlaced] < blockSize[j])
                indexPlaced = j;
        }
    }

    // If we were successfully able to find block for the process
    if (indexPlaced != -1)
    {
        // allocate this block j to process p[i]
        allocation[i] = indexPlaced;

        // make the status of the block as occupied
        occupied[indexPlaced] = 1;

        // Reduce available memory for the block
        blockSize[indexPlaced] -= processSize[i];
    }
  }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < processes; i++)
    {
        printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n",allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

// Driver code
int main()
{
    int blockSize[] = {100, 50, 30, 120, 35};
    int processSize[] = {40, 10, 30, 60};
    int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
    int processes = sizeof(processSize)/sizeof(processSize[0]);

    implimentWorstFit(blockSize, blocks, processSize, processes);

    return 0;
}
```