```
    }

    if (safe) {
        printf("The SAFE Sequence is\n");
        for (int i = 0; i < n; i++) {
            printf("P%d", safe_seq[i]);
            if (i != n-1) printf(" -> ");
        }
        printf("\n");
    } else {
        printf("No safe sequence exists.\n");
    }

    return 0;
}
```

Sample Output:

The SAFE Sequence is
P1 -> P3 -> P4 -> P0 -> P2

Result:

The Banker's Algorithm was successfully implemented to find a safe sequence, demonstrating the ability to avoid deadlocks and ensure system stability in resource allocation.

Ex. No.: 10a)
Date:

BEST FIT

Aim:

To implement Best Fit memory allocation technique using Python.

Algorithm:

1. Input memory blocks and processes with sizes
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process
4. If found then assign it to the current process.
5. If not found then leave that process and keep checking the further processes.

Program Code:

```
  def best_fit():
    # Get memory blocks
    memory_blocks = list(map(int, input("Enter memory block sizes (separated by space): ").split()))

    # Get process sizes
    process_sizes = list(map(int, input("Enter process sizes (separated by space): ").split()))

    allocation = [-1] * len(process_sizes)

    for i in range(len(process_sizes)):
        best_idx = -1
        for j in range(len(memory_blocks)):
            if memory_blocks[j] >= process_sizes[i]:
                if best_idx == -1 or memory_blocks[j] < memory_blocks[best_idx]:
                    best_idx = j

        if best_idx != -1:
            allocation[i] = best_idx
            memory_blocks[best_idx] -= process_sizes[i]

    print("\nProcess No.  Process Size  Block No.")
    for i in range(len(process_sizes)):
        print(f"{i+1}\t\t{process_sizes[i]}\t\t", end="")
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("Not Allocated")
```

```
# Run the program
print("Best Fit Memory Allocation")
best_fit()
```

Sample Output:

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

Result:

The **Best Fit memory allocation technique** was implemented using **Python**, showing effective **memory management** by **minimizing wasted space** and optimizing allocation.

Ex. No.: 10b)
Date:

FIRST FIT

Aim:

To write a C program for implementation memory allocation methods for fixed partition using first fit.

Algorithm:

1. Define the max as 25.
2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max]. 3: Get the number of blocks,files,size of the blocks using for loop.
4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
5: Check highest

Program Code:

```
#include <stdio.h>
#define max 25

int main() {
    int frag[max], b[max], f[max], bf[max] = {0}, ff[max];
    int nb, nf, i, j, temp;

    printf("Enter number of blocks: ");
    scanf("%d", &nb);
    printf("Enter size of each block:\n");
    for(i = 0; i < nb; i++) scanf("%d", &b[i]);

    printf("Enter number of files: ");
    scanf("%d", &nf);
    printf("Enter size of each file:\n");
    for(i = 0; i < nf; i++) scanf("%d", &f[i]);

    for(i = 0; i < nf; i++) {
        for(j = 0; j < nb; j++) {
            if(bf[j] != 1 && b[j] >= f[i]) {
                ff[i] = j;
                frag[i] = b[j] - f[i];
                bf[j] = 1;
                break;
            }
        }
    }
```

```
    printf("\nFile_no\tFile_size\tBlock_no\tBlock_size\tFragment");
    for(i = 0; i < nf; i++)
        printf("\n%d\t%d\t\t%d\t\t%d\t\t%d", i+1, f[i], ff[i]+1, b[ff[i]], frag[i]);

    return 0;
}
```

Sample Output:

```
Enter the number of blocks:4
Enter the number of files:3

Enter the size of the blocks:-
Block 1:5
Block 2:8
Block 3:4
Block 4:10
Enter the size of the files:-
File 1:1
File 2:4
File 3:7

File_no:        File_size :     Block_no:       Block_size:     Fragment
1               1               1               5               4
2               4               2               8               4
3               7               4               10              3
```

Result:

A **C program** was written to implement **memory allocation using fixed partitions with First Fit**, proving the feasibility of **static partitioning** and efficient allocation based on the **first available block**.