Ex. No: 10a Date: 7/3/25

BEST FIT

AIM:

To implement Best Fit memory allocation technique using Python.

ALGORITHM:

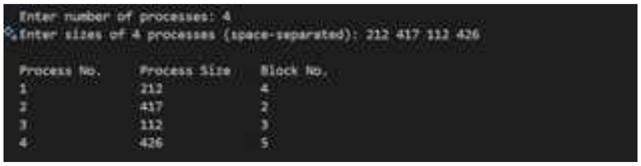
1. Input memory blocks and processes with sizes
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process
4. If found then assign it to the current process.
5. If not found then leave that process and keep checking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

PROGRAM:

```
def best_fit(blocks, processes): allocation = [-1] * len(processes)


for i in range(len(processes)): best_index = -1


for j in range(len(blocks)):
if blocks[j] >= processes[i]:
if best_index == -1 or blocks[j] < blocks[best_index]: best_index = j


if best_index != -1: allocation[i] = best_index
blocks[best_index] -= processes[i]
print("\nProcess No.\tProcess Size\tBlock No.") for i in range(len(processes)):
print(f"{i + 1}\t\t{processes[i]}\t\t{allocation[i] + 1 if allocation[i] != -1 else 'Not Allocated'}")


if _name______== "_main_":
num_blocks = int(input("Enter number of memory blocks: "))
```

```
blocks = list(map(int, input(f"Enter sizes of {num_blocks} memory blocks (space-separated): ").split()))


num_processes = int(input("\nEnter number of processes: "))

processes = list(map(int, input(f"Enter sizes of {num_processes} processes (space-separated): ").split()))
best_fit(blocks, processes)
```

OUTPUT:



```
Enter number of processes: 4
Enter sizes of 4 processes (space-separated): 212 417 112 426

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

RESULT:

Thus, the Best Fit Memory allocation technique is implemented successfully using Python.

Ex. No: 10b Date: 7/3/25

FIRST FIT

AIM:

To write a C program for the implementation of memory allocation methods for a fixed

partition using the first fit.

ALGORITHM:

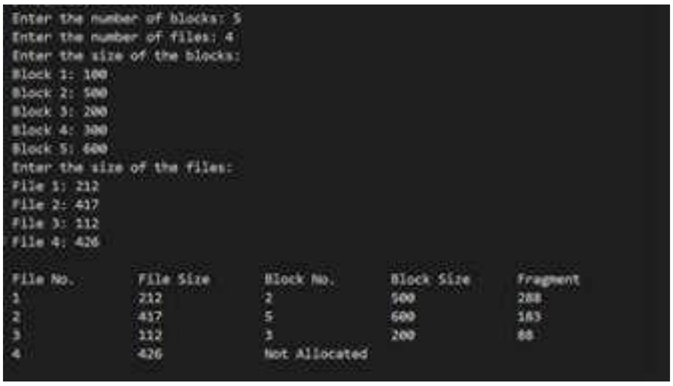
1. Define the max as 25.
2. Declare the variable frag[max],b[max],f[max],i,j, nb,nf, temp, highest=0, bf[max],ff[max].
3. Get the number of blocks, files, size of the blocks using a for loop.
4. In for loop check bf[j]!=1, if so temp=b[j]-f[i]
5. Check the highest.

PROGRAM:

```
#include <stdio.h> #define MAX 25
int main() {
int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp; static int bf[MAX], ff[MAX];
printf("Enter the number of blocks: "); scanf("%d", &nb);

printf("Enter the number of files: "); scanf("%d", &nf);
printf("Enter the size of the blocks:\n"); for (i = 0; i < nb; i++) {
printf("Block %d: ", i + 1);
scanf("%d", &b[i]);
}

printf("Enter the size of the files:\n"); for (i = 0; i < nf; i++) {
printf("File %d: ", i + 1);
```

```
scanf("%d",
&f[i]);

}

for (i = 0; i < nf;
i++) { for (j = 0; j
< nb; j++) {

if (bf[j] != 1) {

temp = b[j] -
f[i]; if (temp >=
0) {

ff[i] = j;

bf[j] = 1; frag[i]
= temp; break;

}
printf("\nFile No.\tFile Size\tBlock No.\tBlock Size\tFragment\n"); for (i = 0; i < nf; i++) {
if (bf[ff[i]] == 1)
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]); else
printf("%d\t\t%d\t\tNot Allocated\n", i + 1, f[i]);
}

return 0;
}
```

OUTPUT:



```
Enter the number of blocks: 5
Enter the number of files: 4
Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 426

File No.        File Size       Block No.       Block Size      Fragment
1               212             2               500             288
2               417             5               600             183
3               112             3               200             88
4               426             Not Allocated
```

RESULT:

Thus, the First Fit allocation technique is implemented successfully using C.