The modern computer system has memory space divided into blocks of varying sizes. The operating system assigns these memory block to the different process demanding empty memory spaces. This memory is assigned in the main memory segment depending on the demanded memory and the empty memory slots present in the main memory.

Assigning memory blocks to the process is challenging as the primary memory is needed to be divided among the operating system, user process, and the operating system process. Therefore, the system uses different algorithms to allocate memory from the main memory segment. These algorithms are also known as the memory partitioning algorithms are broadly categorized under the following algorithms:

- First Fit
- Best Fit
- Worst Fit

Example of Best Fit Method

The best fit memory allocation method the memory is traversed until a suitable empty block is found. In this method, the memory wastage is minimal as the approach allocates the memory blocks with minimum memory wastage.

In the given example, let us assume the jobs and the memory requirements as the following:

Job 1 90k
Job 2 20k
Job 3 50k
Job 4 200k

Let the free pace memory allocation blocks be:

Block 1 50k
Block 2 100k
Block 3 90k
Block 4 200k
Block 5 50k

50k 100k 90k 200k 50k

1. Job 1 (size – 90k)

• The memory is traversed searching for the memory space with minimum wastage.

• The first block is not assigned as it is smaller than the demanded memory. 50k (block size) < 90k (job size).

• The second block is also not allocated as it will cause memory wastage.

• Control moves to the third block which is then allocated as the process size and the memory sizes are equal.

• Job 1 is assigned to block 3 with no memory wastage.

2. Job 2 (size – 20k)

• The CPU searches the memory for an empty memory slot which is either equal to or more than 100k.

• The first block is selected as it is slightly greater than the process size.

• Block 1 is allocated to the second process.

• Waste in size is of 30k

3. Job 3 (size – 50k)

• The CPU searches the entire memory block to allocate free memory.

• The first block is already allocated to job 2. Hence it is skipped.

• The CPU moves to the second block but is greater in size than the demanded memory (50 k < 100 k).

• The third block is already occupied, and hence it is skipped.

• The CPU now moves to the fourth block which is also greater than the demanded size. Hence it is also skipped.

• Now the fifth block is accessed. In this case, the demanded memory and the available memory sizes are equal (50k = 50k).

• Hence the block five are allocated to job 3.

4. Job 4 (size- 200k)

• The CPU searches the memory blocks for allocation.

• It skips the first block as it is already occupied.

• The second block in the memory is skipped as it is smaller than the demanded memory.

• The third memory block is also skipped as it is already occupied.

• CPU then moves to the fourth memory block which is finally allocated as its size exactly meets the demanded memory size of the process (200k = 200k).

• Block four is allocated to the memory which no memory wastage.

5. All the processes are assigned memory blocks.
6. Block second remains unoccupied at the end of the memory allocation procedure.
Memory allocation at the end of the algorithm can be seen as follows.

Job 2 (20k) Job 1 (90k) Job 4 (200k) Job 3 (50k)

50k 100k 90k 200k 50k

Unused memory
Block 1= 30 K
Block 2= Nil
Block 3= 0 K
Block 4= 0 K
Block 5= 0 K

Hence, the method involves the least amount of wastage memory and hence is known as the Best Fit method.

Example of First Fit Method

In this method, whichever partition has the memory space equal or more than the demanded memory by the process, is allocated to the process at the first attempt during memory traversing. This method continues until all the processes are allocated free memory blocks or no correct free memory block is left that can be assigned to a process.

In the given example, let us assume the jobs and the memory requirements as the following:

Job 1 90k
Job 2 20k
Job 3 50k
Job 4 10k

Let the free pace memory allocation blocks be:

Block 1 50k
Block 2 100k

Block 3 90k

Block 4 200k

Block 5 50k

50k 100k 90k 200k 50k

1.Job 1 (size – 90k)

The job cannot fit in the first block as 50k (block size) < 90k (job size).

• Goes to block 2 as can fit in block 2 as 100K > 90K (Size of the block more than the size of the process).

• The job is assigned to the second block.

• 10K remains empty in this block now.

2. Job 2 (size 20K)

• The job can easily fit in the first block 20K < 100k•Job is assigned to the first block.

• 30k remains empty in this block now

3.Job 3 (size 50 K)

• The next empty memory block is the third block with condition 50k < 90k.

• The job is thus allocated to the third block.

• 40k remains empty in this block.

4.Job 4 (10k)

• The first, second, and the third blocks in the memory are already occupied.

• The fourth job thus reaches the fourth empty memory block.

• The size of the block is greater than the size of the job (200k > 10 k).

• The memory block is assigned to the fourth job.

• 190k remains empty in this block.

5.Job 5 (100k)

• Memory blocks one to five are already occupied.

• Job 5 reaches the fifth memory block but cannot be assigned to it as the block is smaller than the required memory space.

• Thus, the job has to wait until a job is finished and an empty space is released.

Job 5 remains do not get a memory, and the last memory block remains unallocated.

The following diagrammatic representation shows memory allocation after the first fit method.

Job 2(20k) Job 1(90k) Job 3 (50k) Job 4 (10k)

50k 100k 90k 200k 50k

Unused memory

Block 1= 30 K

Block 2= 10 K

Block 3= 40 K

Block 4= 190 K

Block 5= 50 K

Example of Worst Fit Method

The worst fit approach is directly opposite to the best-fit approach. In this method, the CPU searches for a memory block which is greater to the memory in demand. In fact, it searches for an empty memory block which is exceptionally bigger than the demanded memory. The approach is known as the worst fit method as it causes the maximum amount of memory wastage in the memory.

In the given example, let us assume the jobs and the memory requirements as the following:

Job 1 10k

Job 2 20k

Job 3 30k

Job 4 70k

Let the free pace memory allocation blocks be:

Block 1 50k

Block 2 100k

Block 3 90k

Block 4 200k

Block 5 50k

50k 100k 90k 200k 50k

1. Job1 (Size – 10k)

• The CPU searches the empty memory slots which is greater than the demanded memory.

• It skips memory block one, two, and three to find the largest empty memory space.

• The CPU allocated memory block four which is of 200k and is the largest among all the slots.

• Job 1 is assigned to block four.

• The wasted memory is 190k.

2. Job 2 (size – 20k)

• The CPU searches for the largest empty block.

• It skips the first memory block and reaches to the second.

• As the second is the largest empty block available, it is allotted to the second job.

• The memory wastage, in this case, is 80k.

3. Job 3 (size- 30k)

• The CPU skips the first memory block as an empty block of 90k remains empty.

• It skips the second block as it is already occupied by job 2.

• CPU finds an empty block of size 90k and assigns it to job 3.

• The third block is assigned to job 3.

• The amount of wasted memory, in this case, is 60k.

4. Job 4 (size -70k)

• The CPU skips the first memory block as it is smaller than the demanded memory.

• It further skips the second, third and fourth blocks as they are already occupied.

• The CPU cannot allocate the fifth and the last block as it is smaller than the demanded memory.

• Job 4 remains unassigned.

5. Block one and six remain unoccupied, and Job 4 remains unassigned.

Memory allocation at the end of the algorithm can be seen as follows.

Job 2 (20k) Job 3 (30k) Job 1 (10k)

50k 100k 90k 200k 50k

Unused memory

Block 1= 50k

Block 2= 80k

Block 3= 60 K

Block 4= 190 K

Block 5= 50 K

Hence, the method involves a maximum amount of wastage memory and hence is known as the Worst Fit method.

Contiguous Memory Allocation Techniques

First Fit

In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

Best Fit

The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

Worst fit

In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

Program Code for First Fit

```
#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp;

static int bf[max],ff[max];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);
```

```
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files:-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
```

```
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragment");

for(i=1;i<=nf;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}
```

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_
```

Program Code for Best Fit

```
#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

static int bf[max],ff[max];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);
```

```
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files:-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;
lowest=temp;
}
}
}
```

```
frag[i]=lowest;

bf[ff[i]]=1;

lowest=10000;

}

printf("\nFile_no  \tFile_size  \tBlock_no  \tBlock_size  \tFragment");

for(i=1;i<=nf && ff[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}
```

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               3               4               3
2               4               1               5               1
3               7               2               8               1
```

Program Code for Worst Fit

```
#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;

static int bf[max],ff[max];
```

```
clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files:-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
```

```
}

}

}

frag[i]=highest;

bf[ff[i]]=1;

highest=0;

}

printf("\nFile_no  \tFile_size  \tBlock_no  \tBlock_size  \tFragment");

for(i=1;i<=nf;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}
```

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               4               10              9
2               4               2               8               4
3               7               0               0               0
```