

Ex. No.: 10a)
Date: 9/4/25

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 Code:

```
def bestfit (brize, m, prize, w):  
    alloc = [-1] * w  
    for i in range(w):  
        bestidoc = -1  
        for j in range(m):  
            if brize[j] >= prize[i]:  
                if bestidoc == -1:  
                    bestidoc = j  
                elif brize[bestidoc] > brize[j]:  
                    bestidoc = j  
            if bestidoc != -1:  
                alloc[i] = bestidoc  
                brize[bestidoc] -= prize[i]  
    for i in range(w):  
        print (i+1, " ", prize[i], end = " ")  
        if (alloc[i] != -1):  
            print (alloc[i] + 1)
```


else:

print C "Not Allocated";

if __name__ == '__main__':

bsize = [100, 500, 200, 300, 600]

fsize = [212, 417, 312, 426]

m = len(bsize)

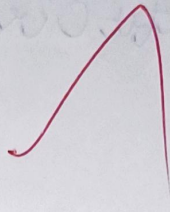
n = len(fsize)

bestfit C (bsize, m, fsize, n)



Output:-

Process No	Process Size	Block no
1	212	4
2	419	2
3	312	5
4	426	Not Allocated

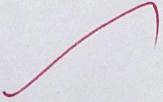


Sample Output:

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

Result:

Hence the Best Fit for the given processes is Implemented and Verified


alt

Ex. No.: 10b)
Date: 10/4/25

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 b[MAX], bo[MAX], bc, bs[MAX], be,  
alloc[MAX], fragment[MAX];
```

```
printf("Enter no of Memory Blocks: ");  
scanf("%d", &bc);
```

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

```
printf("Enter size of Each block: ");
```

```
for(int i=0; i<bc; i++)
```

```
{
```

```
printf("Block %d size: ", i+1);
```

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

```
bo[i] = bs[i];
```

```
}
```



```

printfC "In Enter no of files (max: 10):";
scanfC "%d", &bc);
printfC "Enter size of each file (in)";
forC int i=0; i<bc; i++)
{
    printfC "File %d size", i+1);
    scanfC "%d", &fs[i]);
    alloc[i] = -1;
    fragment[i] = 0;
}

```

```

}
forC int i=0; i<bc; i++)
{
    forC int j=0; j<bc; j++)
    {
        if (bs[j] >= fs[i])
        {
            alloc[i] = j;
            fragment[i] = bs[j] - fs[i];
            bs[j] -= fs[i];
            break;
        }
    }
}

```

```

}
printfC "In File No \t File size \t Block  
No \t Block size \t Fragment (in)";

```

```

forC int i=0; i<bc; i++)
{
    printfC "%d \t %d \t %d \t %d", i+1, fs[i],
    if (alloc[i] != -1)
    {
        int bno = alloc[i];
        printfC "%d \t %d \t %d \t %d",
        bno+1, bs[bno], fragment[i],
    }
}

```


else {

printf("Not Allocated (t-t-t-t-t)");

}

}
return 0;

}

Output :-

Enter no of memory Blocker :- 4

Block 1 size: 300

Block 2 size: 250

Block 3 size: 400

Block 4 size: 500

Enter no of files: 4

File 1 size: 212

File 2 size: 200

File 3 size: 312

File 4 size: 417

File No	File size	Block No	Block size	Fragment
1	212	2	250	38
2	200	1	300	100
3	312	3	400	88
4	417	4	500	83

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:

Hence the first fit algorithm is Imple-
mented and executed.