

Write a program to simulate Memory placement strategies – best fit, first fit, next fit and worst fit.

```
class memoryManagement_variable:
```

```
    def __init__(self):
```

```
        self.memoryBlocks = [int(i) for i in input("Enter sizes of the memory blocks: ").split(" ")]
```

```
        self.numProcesses = int(input("Enter number of processes: "))
```

```
        self.processSizes = [int(i) for i in input("Enter the size of processes: ").split(" ")]
```

```
    ]
```

```
    self.allocatedBlock = [-1] * self.numProcesses
```

```
    self.blockSizeRem = [self.memoryBlocks[i] for i in range(self.numProcesses)]
```

```
    def printTable(self):
```

```
        print('%-10s%-15s%-15s%-15s%-15s' %("Process","Process Size","Block Number","Block Size","Unused Memory"))
```

```
        for i in range(self.numProcesses):
```

```
            if self.allocatedBlock[i] != -1:
```

```
                print('%-10i%-15i%-15i%-15i%-15i' %(i+1,self.processSizes[i],self.allocatedBlock[i] + 1,self.memoryBlocks[self.allocatedBlock[i]],self.blockSizeRem[self.allocatedBlock[i]]))
```

```
            else:
```

```
                print('%-10i%-15i%-15s%-15s%-15s' %(i+1,self.processSizes[i],"N/A","-","-"))
```

```
        print("\n")
```

```
class memoryManagement_fixed:
```

```
    def __init__(self):
```

```
        self.memoryBlocks = [int(i) for i in input("Enter sizes of the memory blocks: ").split(" ")]
```

```
        self.numProcesses = int(input("Enter number of processes: "))
```

```
        self.processSizes = [int(i) for i in input("Enter the size of processes: ").split(" ")]
```

```
    ]
```

```
self.allocatedBlock = [-1] * self.numProcesses  
self.memoryAllocated = [False] * self.numProcesses
```

```
def printTable(self):  
    print('%-10s%-15s%-15s%-15s' %("Process","Process Size","Block Number","Block Size"))  
    for i in range(self.numProcesses):  
        if self.allocatedBlock[i] != -1:  
            print('%-10i%-15i%-15i%-15i' %(i+1,self.processSizes[i],self.allocatedBlock[i] +  
1,self.memoryBlocks[self.allocatedBlock[i]]))  
        else:  
            print('%-10i%-15i%-15s%-15s' %(i+1,self.processSizes[i],"N/A","-"))  
    print("\n")
```

```
class bestFitVariable(memoryManagement_variable):  
    def __init__(self):  
        memoryManagement_variable.__init__(self)  
  
    def execute(self):  
  
        for i in range(self.numProcesses):  
            bestBlock = -1  
            for block in range(len(self.memoryBlocks)):  
                if self.blockSizeRem[block] >= self.processSizes[i]:  
                    if bestBlock == -1:  
                        bestBlock = block  
                    elif self.blockSizeRem[bestBlock] > self.blockSizeRem[block]:  
                        bestBlock = block
```

```

        if bestBlock != -1:

            self.allocatedBlock[i] = bestBlock

            self.blockSizeRem[bestBlock] -= self.processSizes[i]

    print("\nBEST FIT - Variable Size of Memory Block:")

    self.printTable()

```

```

class bestFitFixed(memoryManagement_fixed):

    def __init__(self):
        super().__init__()

    def execute(self):

        for i in range(self.numProcesses):

            bestBlock = -1

            for block in range(len(self.memoryBlocks)):

                if self.memoryBlocks[block] >= self.processSizes[i] and self.memoryAllocated[block] == False:

                    if bestBlock == -1:

                        bestBlock = block

                    elif self.memoryBlocks[bestBlock] > self.memoryBlocks[block]:

                        bestBlock = block

            if bestBlock != -1:

                self.allocatedBlock[i] = bestBlock

                self.memoryAllocated[bestBlock] = True

    print("\nBEST FIT - Fixed size of memory block:")

    self.printTable()

```

```

class worstFitVariable(memoryManagement_variable):

    def __init__(self):
        super().__init__()

```

```

def execute(self):
    for i in range(self.numProcesses):
        worstBlock = -1
        for block in range(len(self.memoryBlocks)):
            if self.blockSizeRem[block] >= self.processSizes[i]:
                if worstBlock == -1:
                    worstBlock = block
                elif self.blockSizeRem[worstBlock] < self.blockSizeRem[block]:
                    worstBlock = block

        if worstBlock != -1:
            self.allocatedBlock[i] = worstBlock
            self.blockSizeRem[worstBlock] -= self.processSizes[i]
    print("\nWORST FIT - Variable Size of Memory Block:")
    self.printTable()

```

```

class worstFitFixed(memoryManagement_fixed):
    def __init__(self):
        super().__init__()

    def execute(self):
        for i in range(self.numProcesses):
            worstBlock = -1
            for block in range(len(self.memoryBlocks)):
                if self.memoryBlocks[block] >= self.processSizes[i] and self.memoryAllocated[block] == False:
                    if worstBlock == -1:
                        worstBlock = block
                    elif self.memoryBlocks[worstBlock] < self.memoryBlocks[block]:
                        worstBlock = block

```

```

        if worstBlock != -1:
            self.allocatedBlock[i] = worstBlock
            self.memoryAllocated[worstBlock] = True
        print("\nWORST FIT - Fixed Size of Memory Block:")
        self.printTable()

```

```

class firstFitVariable(memoryManagement_variable):

```

```

    def __init__(self):
        super().__init__()

```

```

    def execute(self):
        for i in range(self.numProcesses):
            for block in range(len(self.memoryBlocks)):
                if self.blockSizeRem[block] >= self.processSizes[i]:
                    self.allocatedBlock[i] = block
                    self.blockSizeRem[block] -= self.processSizes[i]
                    break
        print("\nFIRST FIT - Variable Size of Memory Block:")
        self.printTable()

```

```

class firstFitFixed(memoryManagement_fixed):

```

```

    def __init__(self):
        super().__init__()

```

```

    def execute(self):
        for i in range(self.numProcesses):
            for block in range(len(self.memoryBlocks)):
                if self.memoryBlocks[block] >= self.processSizes[i] and self.memoryAllocated[block] == False:

```

```
        self.allocatedBlock[i] = block

        self.memoryAllocated[block] = True

        break

    print("\nFIRST FIT - Fixed Size of Memory Block:")

    self.printTable()
```

```
class nextFitVariable(memoryManagement_variable):

    def __init__(self):

        super().__init__()

    def execute(self):

        j = 0

        for i in range(self.numProcesses):

            while j < len(self.memoryBlocks):

                if self.blockSizeRem[j] >= self.processSizes[i]:

                    self.allocatedBlock[i] = j

                    self.blockSizeRem[j] -= self.processSizes[i]

                    break

                j = (j + 1) % len(self.memoryBlocks)

        print("\nNEXT FIT - Variable Size of Memory Block")

        self.printTable()
```

```
class nextFitFixed(memoryManagement_fixed):

    def __init__(self):

        super().__init__()

    def execute(self):

        j = 0

        for i in range(self.numProcesses):

            while j < len(self.memoryBlocks):

                if self.memoryBlocks[j] >= self.processSizes[i] and self.memoryAllocated[j] == False:
```

```
        self.allocatedBlock[i] = j

        self.memoryAllocated[j] = True

        j = (j + 1) % len(self.memoryBlocks)

        break

    j = (j + 1) % len(self.memoryBlocks)

    print("\nNEXT FIT - Fixed Size of Memory Block")

    self.printTable()
```

```
while True:
```

```
    print("1>Best Fit",
          "2>Worst Fit",
          "3>First Fit",
          "4>Next Fit",
          "5>Exit",sep="\n",end="\n\n")
```

```
choice = int(input("Enter a choice: "))
```

```
if choice == 1:
```

```
    bf = bestFitVariable()
    bf.execute()
    bf = bestFitFixed()
    bf.execute()
```

```
elif choice == 2:
```

```
    wf = worstFitVariable()
    wf.execute()
    wf = worstFitFixed()
    wf.execute()
```

```
elif choice == 3:
```

```
    ff = firstFitVariable()
    ff.execute()
```

```
ff = firstFitFixed()
ff.execute()
elif choice == 4:
    nf = nextFitVariable()
    nf.execute()
    nf = nextFitFixed()
    nf.execute()
else:
    break
```