19131A05P1 V SATYA SIVA LALITHA GAYATHRI BODA CSE4

Os week 1

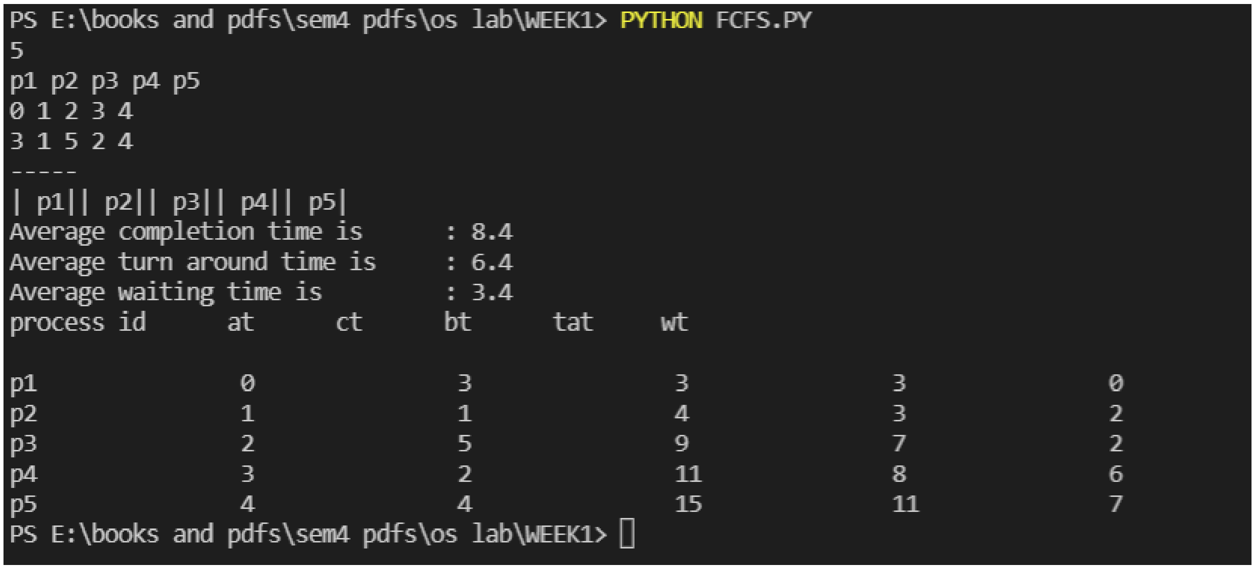
FCFS:

PROGRAM :

```
n=int(input())
p=list(map(str,input().split()))
at=list(map(int,input().split()))
bt=list(map(int,input().split()))
at1=at.copy()
gt=[]
ct=[0]*n
tat=[0]*n
wt=[0]*n


k=min(at)
ind=at.index(k)
gt.append(p[ind])
ct[ind]=bt[ind]
tat[ind]=ct[ind]-at[ind]
wt[ind]=tat[ind]-bt[ind]
at[ind]=99999
pre=ind
i=0
while i<n-1:
    r=min(at)
    rind=at.index(r)
    kk=ct[pre]
    kkk=at[rind]
    if(at[rind]<=ct[pre]):
        ct[rind]=ct[pre]+bt[rind]
    elif(at[rind]>ct[pre]):
        ct[rind]=at[rind]+bt[rind]
    gt.append(p[rind])
    tat[rind]=ct[rind]-at[rind]
    wt[rind]=tat[rind]-bt[rind]
    pre=rind
    at[rind]=999999
    i+=1

print("-----")
```

```
for i in range(0,n):


    print("|",gt[i],end="|")

print()
print('Average completion time is \t:',sum(ct)/n)
print('Average turn around time is \t:',sum(tat)/n)
print('Average waiting time is \t:',sum(wt)/n)
print("process id\tat\tct\tbt\ttat\twt\n")
for i in range(0,n):

print(p[i],"\t\t",at1[i],"\t\t",bt[i],"\t\t",ct[i],"\t\t",tat[i],"\t\t",wt[i])
```

Output:



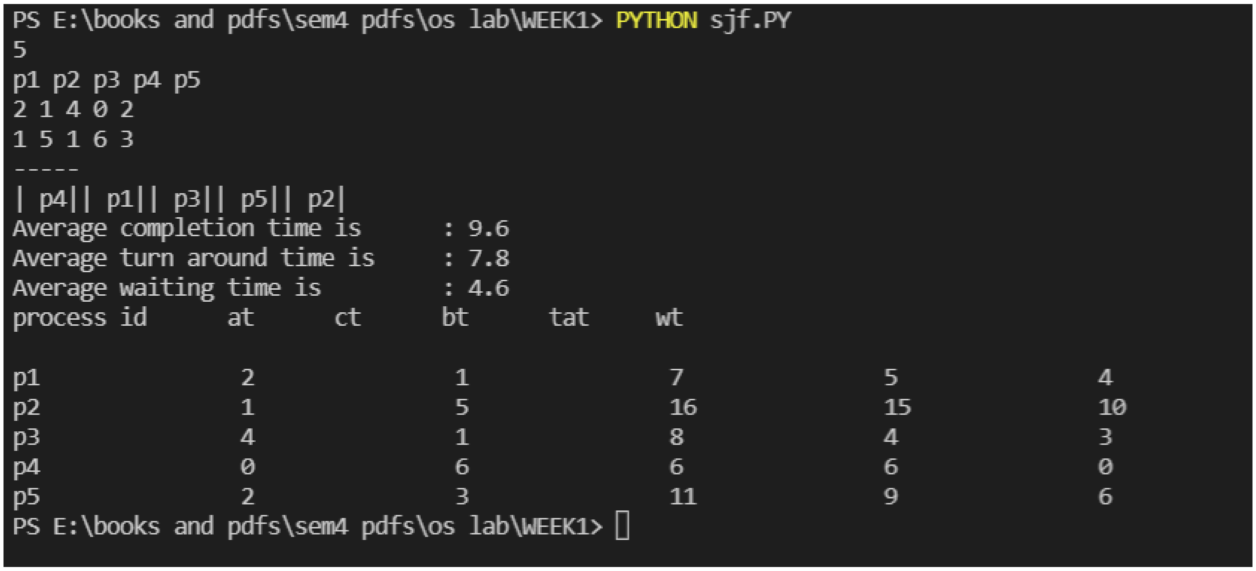
```
PS E:\books and pdfs\sem4 pdfs\os lab\WEEK1> PYTHON FCFS.PY
5
p1 p2 p3 p4 p5
0 1 2 3 4
3 1 5 2 4
-----
| p1|| p2|| p3|| p4|| p5|
Average completion time is      : 8.4
Average turn around time is     : 6.4
Average waiting time is         : 3.4
process id      at      ct      bt      tat     wt

p1               0               3               3               3               0
p2               1               1               4               3               2
p3               2               5               9               7               2
p4               3               2               11              8               6
p5               4               4               15              11              7
PS E:\books and pdfs\sem4 pdfs\os lab\WEEK1> 
```

SJF:

PROGRAM:

```
n=int(input())
p=list(map(str,input().split()))
at=list(map(int,input().split()))


bt=list(map(int,input().split()))
#shortest job
bt1=bt.copy()
k=min(at)
ind=at.index(k)
gt=[]
tat=[0]*n
wt=[0]*n
ct=[0]*n
gt.append(p[ind])


ct[ind]=bt[ind]
tat[ind]=ct[ind]-at[ind]
wt[ind]=tat[ind]-bt[ind]

bt[ind]=999999
pre=ind
i=1
while i<n:
    r=min(bt)
    rind=bt.index(r)
    if at[rind]<ct[pre]:
        ct[rind]=ct[pre]+bt[rind]
        pre=rind
        tat[rind]=ct[rind]-at[rind]
        wt[rind]=tat[rind]-bt[rind]
        gt.append(p[rind])
        # print(rind,at[rind],ct[pre],gt[rind])
        bt[rind]=999999
        i+=1
print("-----")
for i in range(0,n):
    print("|",gt[i],end="|")
print()
print('Average completion time is \t:',sum(ct)/n)
print('Average turn around time is \t:',sum(tat)/n)
print('Average waiting time is \t:',sum(wt)/n)
print("process id\tat\tct\tbt\ttat\twt\n")
for i in range(0,n):

print(p[i],"\t\t",at[i],"\t\t",bt1[i],"\t\t",ct[i],"\t\t",tat[i],"\t\t",wt[
i])
```

Output:



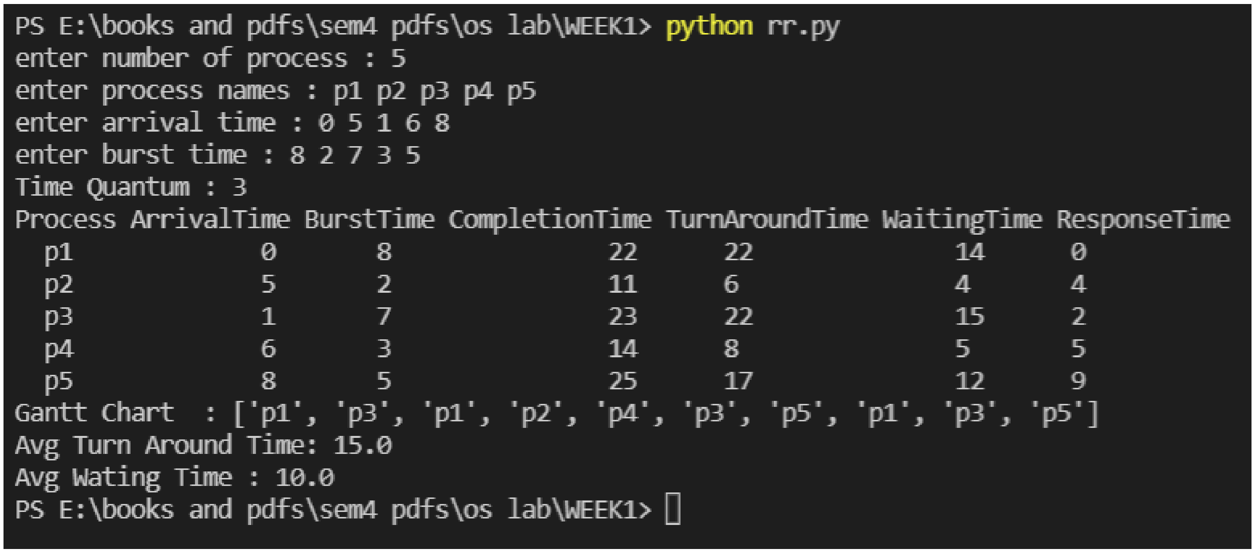
```
PS E:\books and pdfs\sem4 pdfs\os lab\WEEK1> PYTHON sjf.PY
5
p1 p2 p3 p4 p5
2 1 4 0 2
1 5 1 6 3
-----
| p4|| p1|| p3|| p5|| p2|
Average completion time is      : 9.6
Average turn around time is     : 7.8
Average waiting time is         : 4.6
process id      at      ct      bt      tat     wt

p1               2               1               7               5               4
p2               1               5               16              15              10
p3               4               1               8               4               3
p4               0               6               6               6               0
p5               2               3               11              9               6
PS E:\books and pdfs\sem4 pdfs\os lab\WEEK1> 
```

Round robbin:

Program:

```
n = int(input("enter number of process : "))
process = list(map(str, input("enter process names : ").split()))
Arrival_time = list(map(int, input("enter arrival time : ").split()))
Burst_time = list(map(int, input("enter burst time : ").split()))
t = int(input("Time Quantum : "))
Atl =  sorted(Arrival_time)
Bt1 = Burst_time.copy()
gantt_chart = []
ready_queue = []
completion_time = [0]*(n)
waiting_time = [0]*(n)
turn_around_time = [0]*(n)
response_time = [0]*n
val = cnt = flg = i = 0
s = sum(Burst_time)
while (max(completion_time)!=s):
   while(i<len(Atl) and cnt>=Atl[i]):
       ready_queue.append(Atl[i])
       i+=1
   if flg==1:
      ready_queue.append(Arrival_time[x])
   x = Arrival_time.index(ready_queue[0])
   if process[x] not in gantt_chart:
       response_time[x] = val-Arrival_time[x]
   gantt_chart.append(process[x])
   ready_queue.remove(Arrival_time[x])
   if Burst_time[x]<=t and Burst_time[x]!=0:
       completion_time[x] = Burst_time[x] + cnt
       turn_around_time[x] = completion_time[x]-Arrival_time[x]
       waiting_time[x] = turn_around_time[x]-Bt1[x]
       val += Burst_time[x]
       cnt +=Burst_time[x]
       Burst_time[x]=0
       flg=0
   else:
       Burst_time[x] = Burst_time[x]-t
       cnt+=t
       val = cnt
       flg=1
print("Process ArrivalTime BurstTime CompletionTime TurnAroundTime 
WaitingTime ResponseTime")
for i in range(0,len(process)):
   print(" ",process[i],"   \t",Arrival_time[i],"
\t",Bt1[i],"\t\t",completion_time[i],"
\t",turn_around_time[i],"\t\t",waiting_time[i],"\t",response_time[i])
print("Gantt Chart  :",gantt_chart)
print("Avg Turn Around Time:", round(sum(turn_around_time)/n,3))
print("Avg Wating Time :", round(sum(waiting_time)/n,3))
```

Output:



```
PS E:\books and pdfs\sem4 pdfs\os lab\WEEK1> python rr.py
enter number of process : 5
enter process names : p1 p2 p3 p4 p5
enter arrival time : 0 5 1 6 8
enter burst time : 8 2 7 3 5
Time Quantum : 3
Process ArrivalTime BurstTime CompletionTime TurnAroundTime WaitingTime ResponseTime
  p1             0       8               22      22              14      0
  p2             5       2               11      6               4       4
  p3             1       7               23      22              15      2
  p4             6       3               14      8               5       5
  p5             8       5               25      17              12      9
Gantt Chart  : ['p1', 'p3', 'p1', 'p2', 'p4', 'p3', 'p5', 'p1', 'p3', 'p5']
Avg Turn Around Time: 15.0
Avg Wating Time : 10.0
PS E:\books and pdfs\sem4 pdfs\os lab\WEEK1> 
```

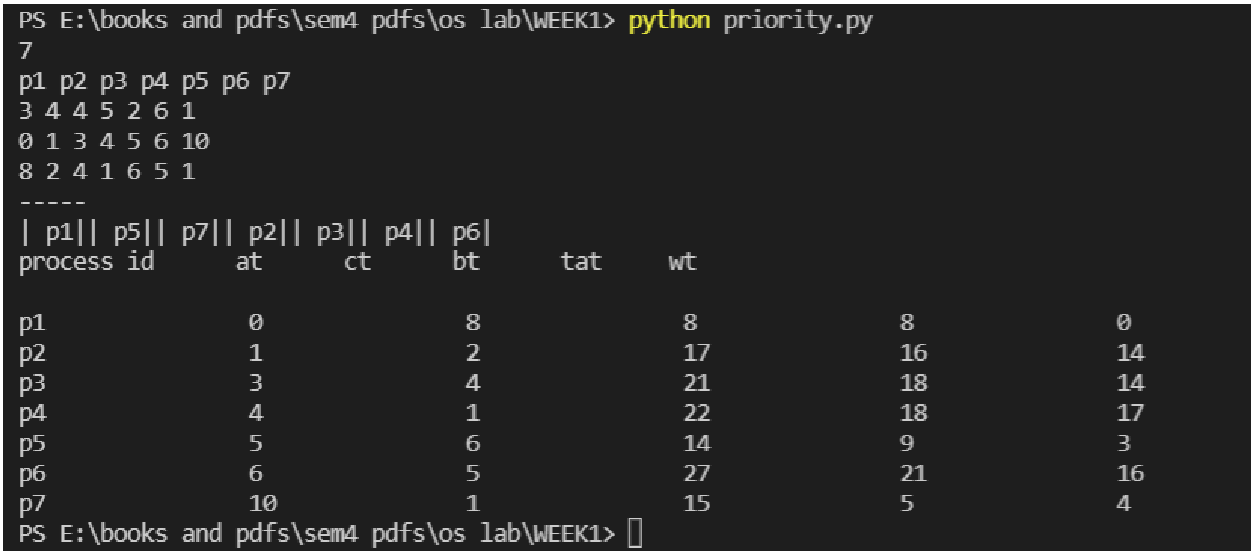
Priority scheduling:

Program:

```
n=int(input())
p=list(map(str,input().split()))
pr=list(map(int,input().split()))
at=list(map(int,input().split()))
bt=list(map(int,input().split()))
pr1=pr.copy()
gt=[]
```

```
ct=[0]*n
tat=[0]*n
wt=[0]*n

ind=at.index(min(at))
pr[ind]=999999
ct[ind]=bt[ind]
gt.append(p[ind])
tat[ind]=ct[ind]-at[ind]
wt[ind]=tat[ind]-bt[ind]

pre=ind

while ct[pre]!=sum(bt):
    rind=pr.index(min(pr))
    if at[rind]>ct[pre]:
        pr1[rind]=999999
        rind=pr1.index(min(pr1))
    ct[rind]=ct[pre]+bt[rind]
    pre=rind
    tat[rind]=ct[rind]-at[rind]
    wt[rind]=tat[rind]-bt[rind]
    pr[rind]=999999
    gt.append(p[rind])

print("-----")
for i in range(0,n):
    print("|",gt[i],end="|")
print()

print("process id\tat\tct\tbt\ttat\twt\n")
for i in range(0,n):

print(p[i],"\t\t",at[i],"\t\t",bt[i],"\t\t",ct[i],"\t\t",tat[i],"\t\t",wt[i
])
```

Output:



```
PS E:\books and pdfs\sem4 pdfs\os lab\WEEK1> python priority.py
7
p1 p2 p3 p4 p5 p6 p7
3 4 4 5 2 6 1
0 1 3 4 5 6 10
8 2 4 1 6 5 1
-----
| p1|| p5|| p7|| p2|| p3|| p4|| p6|
process id      at      ct      bt      tat     wt

p1              0               8               8               8               0
p2              1               2               17              16              14
p3              3               4               21              18              14
p4              4               1               22              18              17
p5              5               6               14              9               3
p6              6               5               27              21              16
p7              10              1               15              5               4
PS E:\books and pdfs\sem4 pdfs\os lab\WEEK1> 
```

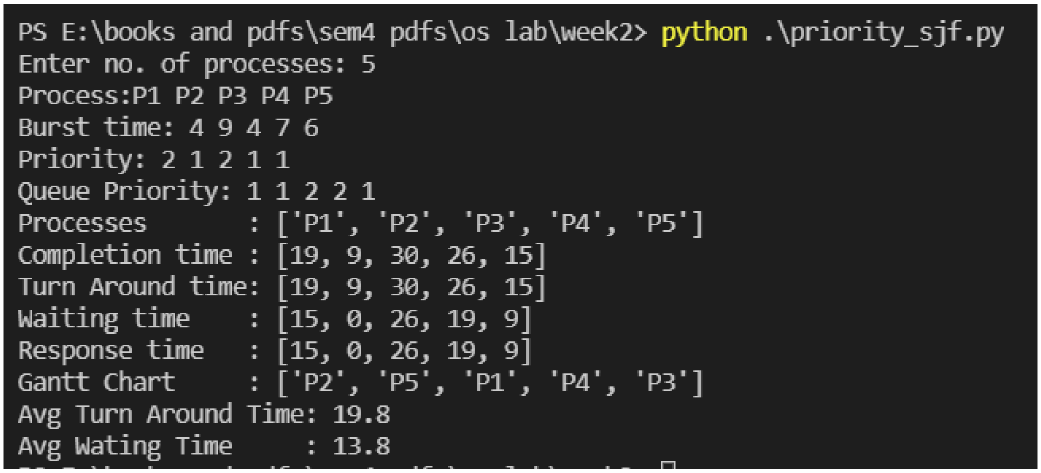
WEEK2

Priority sjf
Code:

```
n = int(input("Enter no. of processes: "))
process = list(map(str, input("Process:").split()))
Bursttime = list(map(int, input("Burst time: ").split()))
priority =list(map(int, input("Priority: ").split()))
Queuepriority =list(map(int, input("Queue Priority: ").split()))

#assuming arrival for all processes is 0
#Q1 = Priority, non preemptive
#Q2 = SJF, non preemptive
gantchart = []
comptime = [0]*n
waitingtime = [0]*n
tat = [0]*n
rt = [0]*n
```

```
val = 0
hold = []

for i in range(n):
    ind = priority.index(min(priority))

    if Queuepriority[ind]!= min(Queuepriority):
        hold.append(ind)
    else:
        gantchart = gantchart + [(process[ind])]
        comptime[ind] = val + Bursttime[ind]
        tat[ind] = comptime[ind]-0 #At[ind]
        waitingtime[ind] = tat[ind]-Bursttime[ind]
        rt[ind] = val- 0 #At[ind]
        val += Bursttime[ind]
    priority[ind]=99999

for i in hold:
    gantchart = gantchart + [(process[i])]
    comptime[i] = val + Bursttime[i]
    tat[i] = comptime[i]-0 #At[ind]
    waitingtime[i] = tat[i]-Bursttime[i]
    rt[i] = val- 0 #At[ind]
    val += Bursttime[i]

print("Processes       :", process)
print("Completion time :",comptime)
print("Turn Around time:",tat)
print("Waiting time    :",waitingtime)
print("Response time   :",rt)
print("Gantt Chart     :",gantchart)
print("Avg Turn Around Time:", round(sum(tat)/n,2))
print("Avg Wating Time     :", round(sum(waitingtime)/n,2))
```

Output:



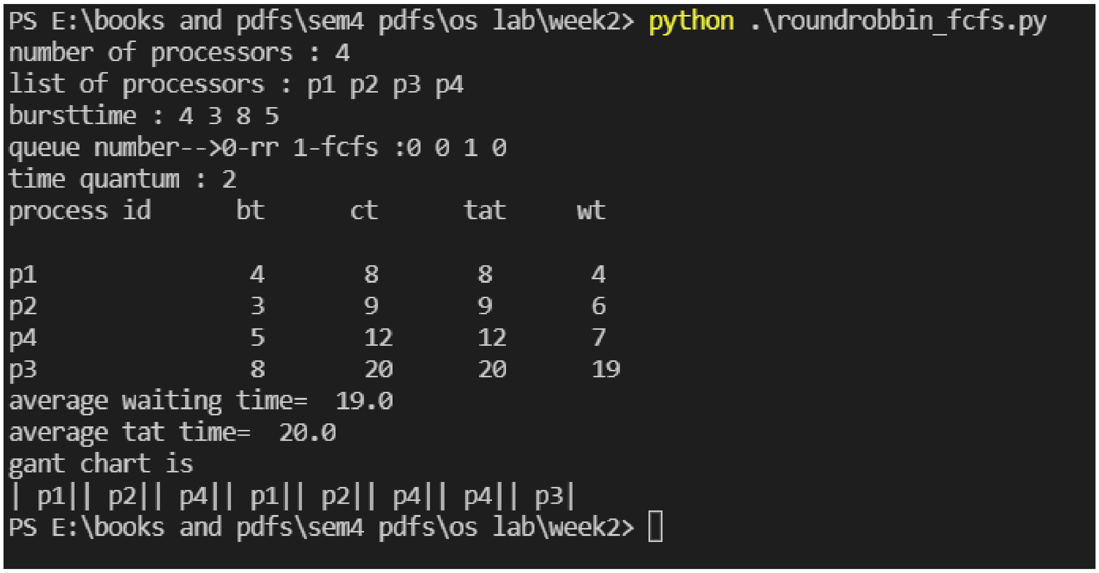
```
PS E:\books and pdfs\sem4 pdfs\os lab\week2> python .\priority_sjf.py
Enter no. of processes: 5
Process:P1 P2 P3 P4 P5
Burst time: 4 9 4 7 6
Priority: 2 1 2 1 1
Queue Priority: 1 1 2 2 1
Processes       : ['P1', 'P2', 'P3', 'P4', 'P5']
Completion time : [19, 9, 30, 26, 15]
Turn Around time: [19, 9, 30, 26, 15]
Waiting time    : [15, 0, 26, 19, 9]
Response time   : [15, 0, 26, 19, 9]
Gantt Chart     : ['P2', 'P5', 'P1', 'P4', 'P3']
Avg Turn Around Time: 19.8
Avg Wating Time     : 13.8
```

RR AND FCFS:

```
gt=[]

def calc_rr(p,bt):
    tq=int(input('time quantum : '))
    bt1=bt.copy()
    rq=[]
    global gt
    n=len(p)
    tat=[0]*n
    wt=[0]*n
    ct=[0]*n
    pre=0
    ind=pre
    i=0
    val=0
    ss=sum(bt)
    global y
    global z
    global x
    while(1):
        while(i<len(p) and bt[i]!=0):
            rq.append(p[i])
            i+=1
        if(len(rq)==0):
            break
        ele=rq[0]
        rq.remove(ele)
        gt.append(ele)
        ind=p.index(ele)
        if(bt[ind]<tq):
            val+=bt[ind]
            bt[ind]=bt[ind]-tq
        else:
            bt[ind]=bt[ind]-tq
            val+=tq
        if(bt[ind]<=0):
            ct[ind]=val
            wt[ind]=ct[ind]-bt1[ind]
            tat[ind]=wt[ind]+bt1[ind]
        if(bt[ind]>0):
            rq.append(p[ind])

    print("process id\tbt\tct\ttat\twt\n")
    for i in range(0,n):
        print(p[i],"\t\t",bt1[i],"\t",ct[i]," \t",tat[i],"\t",wt[i])

    z=ct[n-1]
    y=tat[n-1]
    x=wt[n-1]


def calc_fcfs(p,bt):
    n=len(p)
    global gt
    ct=[0]*n
    wt=[0]*n
    i=0
    global y
    global z
    global x
    while i<n:
        if i==0:
            ct[i]=z+bt[i]
            wt[i]=x+ct[i]-bt[i]
        else:
            ct[i]=ct[i-1]+bt[i]

            wt[i]=ct[i]-bt[i]
        gt.append(p[i])
        i+=1
    tat=ct.copy()
    for i in range(0,n):
        print(p[i],"\t\t",bt[i],"\t",ct[i]," \t",tat[i],"\t",wt[i])
    print("average waiting time= ",sum(wt)/n)
    print("average tat time= ",sum(tat)/n)
```

```
number=int(input("number of processors : "))
process=list(map(str,input("list of processors : ").split()))
bursttime=list(map(int,input("bursttime : ").split()))
queuenum=list(map(int,input("queue number-->0-rr 1-fcfs :").split()))

fcfsbt=[]
fcfs=[]
rr=[]
rrbt=[]
for i in range(len(queuenum)):
    if queuenum[i]==0:
        rr.append(process[i])
        rrbt.append(bursttime[i])
    elif(queuenum[i]==1):
        fcfs.append(process[i])
        fcfsbt.append(bursttime[i])

calc_rr(rr,rrbt)
calc_fcfs(fcfs,fcfsbt)
print("gant chart is")
for i in gt:
    print("|",i,end="|")
```

Output:

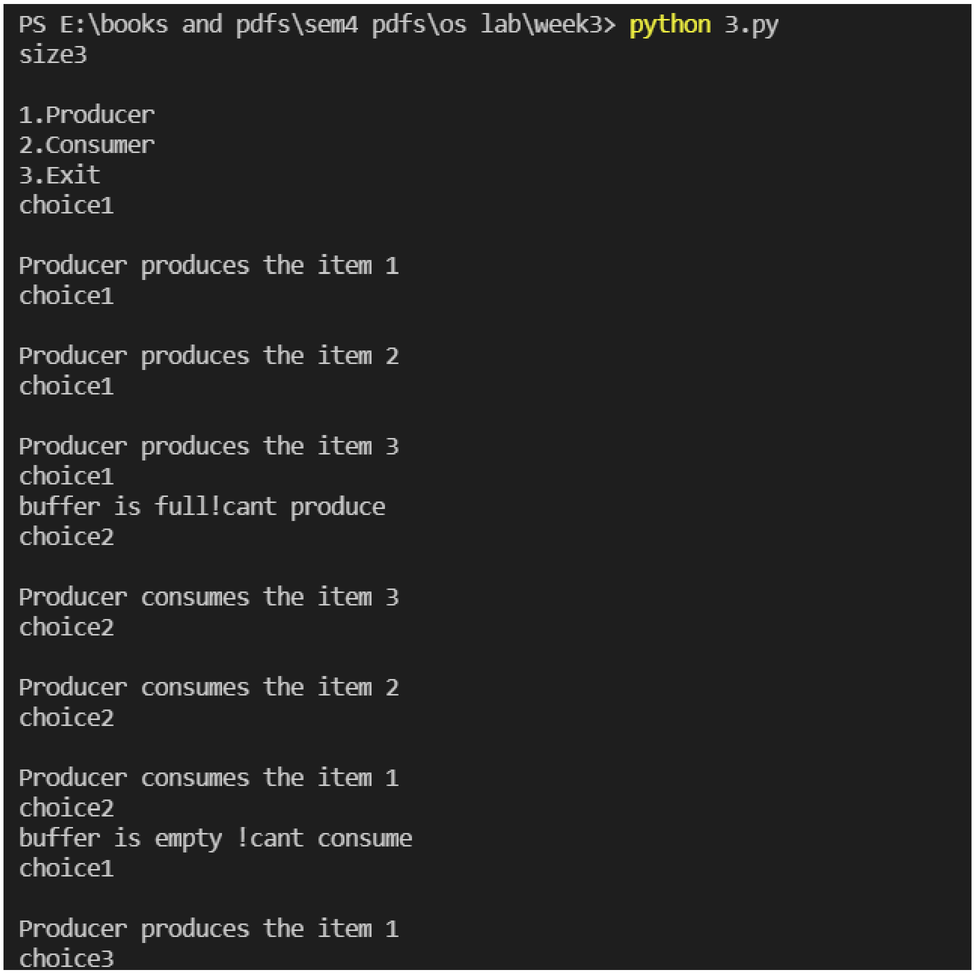


```
PS E:\books and pdfs\sem4 pdfs\os lab\week2> python .\roundrobbin_fcfs.py
number of processors : 4
list of processors : p1 p2 p3 p4
bursttime : 4 3 8 5
queue number-->0-rr 1-fcfs :0 0 1 0
time quantum : 2
process id      bt      ct      tat     wt

p1               4       8       8       4
p2               3       9       9       6
p4               5       12      12      7
p3               8       20      20      19
average waiting time=  19.0
average tat time=  20.0
gant chart is
| p1|| p2|| p4|| p1|| p2|| p4|| p4|| p3|
PS E:\books and pdfs\sem4 pdfs\os lab\week2>
```

Week 3:

```
k=int(input("size"))
mutex=1
full=0
empty=k
x=0
def wait(s):
    s-=1
    return s
def signal(s):
    s+=1
    return s

def producer():
    global mutex,full,empty,x
    mutex=wait(mutex)
    full=signal(full)
    empty=wait(empty)
    x+=1
    print("\nProducer produces the item",x)
    mutex=signal(mutex)


def consumer():
    global mutex,full,empty,x
    mutex=wait(mutex)
    full=wait(full)
    empty=signal(empty)
    print("\nProducer consumes the item",x)
    x-=1

    mutex=signal(mutex)


print("\n1.Producer\n2.Consumer\n3.Exit")
while(1):
    ch=int(input("choice"))
    if ch==1:
        if ((mutex==1) and (empty!=0)):
            producer()
        else:
            print("buffer is full!cant produce")
    elif ch==2:
        if((mutex==1) and (full!=0)):
            consumer()
        else:
            print("buffer is empty !cant consume")
    elif ch==3:
        exit(0)

    else:
        print("enter proper choice")
```

Output:



```
PS E:\books and pdfs\sem4 pdfs\os lab\week3> python 3.py
size3

1.Producer
2.Consumer
3.Exit
choice1

Producer produces the item 1
choice1

Producer produces the item 2
choice1

Producer produces the item 3
choice1
buffer is full!cant produce
choice2

Producer consumes the item 3
choice2

Producer consumes the item 2
choice2

Producer consumes the item 1
choice2
buffer is empty !cant consume
choice1

Producer produces the item 1
choice3
```

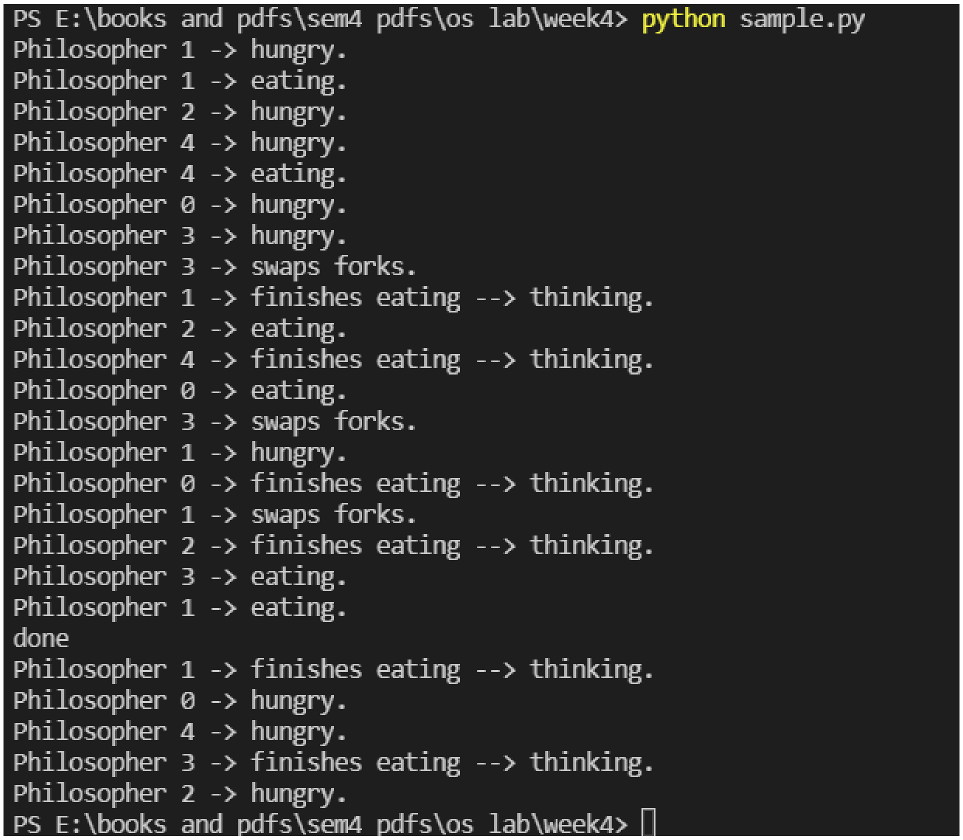
Week 4:

Code:

```
import threading
import random
import time

class Philosopher(threading.Thread):
    running = True

    def __init__(self, index, forkOnLeft, forkOnRight):
        threading.Thread.__init__(self)
        self.index = index
        self.forkOnLeft = forkOnLeft
        self.forkOnRight = forkOnRight
    def run(self):
        while self.running: #true
```

```
            time.sleep(random.uniform(2, 5))
            print('Philosopher %s => hungry.' % self.index)
            self.dine()
    def dine(self):
        fork1, fork2 = self.forkOnLeft, self.forkOnRight
        while self.running:
            fork1.acquire()
            is_free = fork2.acquire(False)
            if is_free: break
            fork1.release()

            print('Philosopher %s => swaps forks.' % self.index)
            fork1, fork2 = fork2, fork1
        else:
            return
        self.dining()
        fork2.release()
        fork1.release()
    def dining(self):
        print('Philosopher %s => eating. ' % self.index)
        time.sleep(random.uniform(1, 5))
        print('Philosopher %s => finishes eating => thinking.' %
self.index)
forks = [threading.Semaphore() for n in range(5)]
philosophers = [Philosopher(i, forks[i % 5], forks[(i + 1) % 5]) for i in
range(5)]
Philosopher.running = True
for p in philosophers:
    p.start()
time.sleep(10)
Philosopher.running = False
print("done")
```

Output:



```
PS E:\books and pdfs\sem4 pdfs\os lab\week4> python sample.py
Philosopher 1 -> hungry.
Philosopher 1 -> eating.
Philosopher 2 -> hungry.
Philosopher 4 -> hungry.
Philosopher 4 -> eating.
Philosopher 0 -> hungry.
Philosopher 3 -> hungry.
Philosopher 3 -> swaps forks.
Philosopher 1 -> finishes eating --> thinking.
Philosopher 2 -> eating.
Philosopher 4 -> finishes eating --> thinking.
Philosopher 0 -> eating.
Philosopher 3 -> swaps forks.
Philosopher 1 -> hungry.
Philosopher 0 -> finishes eating --> thinking.
Philosopher 1 -> swaps forks.
Philosopher 2 -> finishes eating --> thinking.
Philosopher 3 -> eating.
Philosopher 1 -> eating.
done
Philosopher 1 -> finishes eating --> thinking.
Philosopher 0 -> hungry.
Philosopher 4 -> hungry.
Philosopher 3 -> finishes eating --> thinking.
Philosopher 2 -> hungry.
PS E:\books and pdfs\sem4 pdfs\os lab\week4>
```

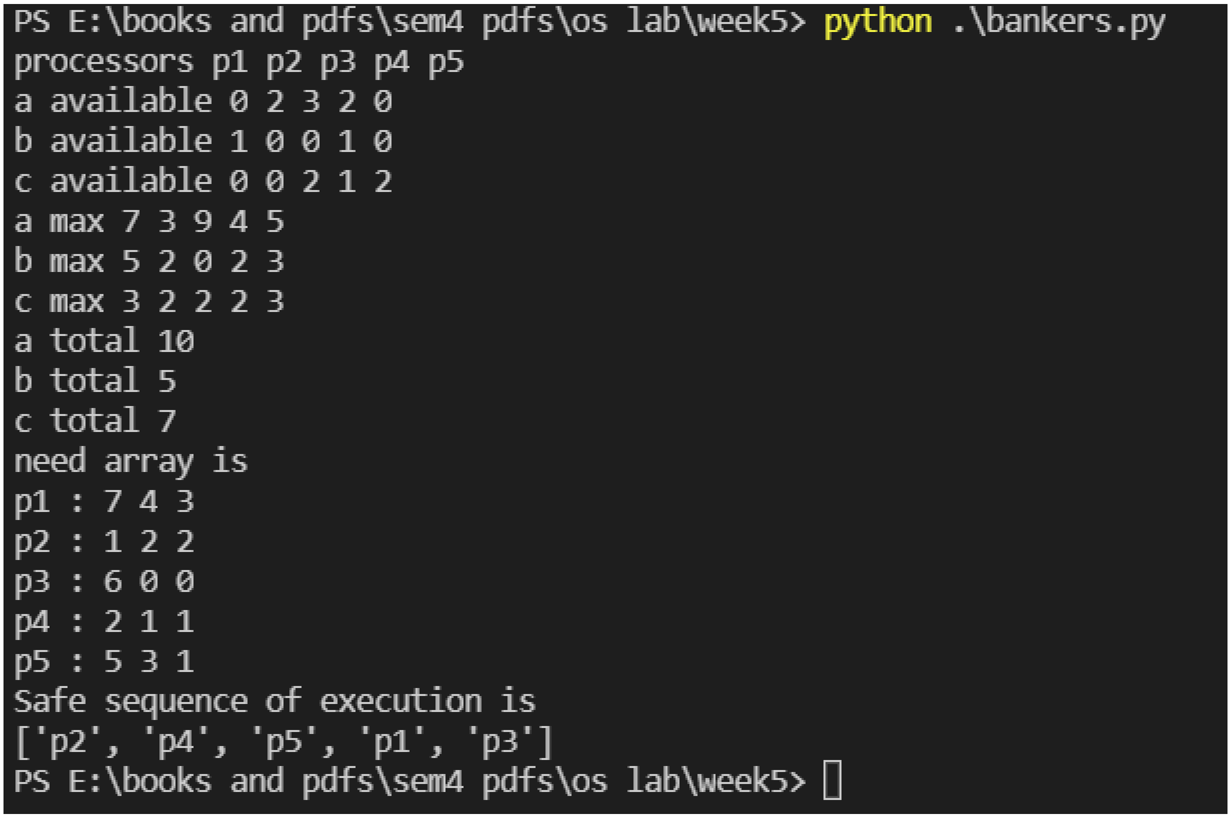
Week 5:

Code:

```
p=list(input('processors ').split())
p1=p.copy()
A=list(map(int,input('a available ').split()))
B=list(map(int,input('b available ').split()))
C=list(map(int,input('c available ').split()))
A_max=list(map(int,input('a max ').split()))
B_max=list(map(int,input('b max ').split()))
C_max=list(map(int,input('c max ').split()))
A_tot=int(input('a total '))
B_tot=int(input('b total '))
C_tot=int(input('c total '))
Aavail=A_tot-sum(A)
Bavail=B_tot-sum(B)
Cavail=C_tot-sum(C)
A_need=[0]*len(p)
B_need=[0]*len(p)
```

```
C_need=[0]*len(p)
for i in range(len(p)):
    A_need[i]=A_max[i]-A[i]
    B_need[i]=B_max[i]-B[i]
    C_need[i]=C_max[i]-C[i]

i=c=0
seq=[]
while c!=len(p):
    if A_need[i]<=Aavail and B_need[i]<=Bavail and C_need[i]<=Cavail and
p[i]!='x':
        seq.append(p[i])
        p[i]='x'
        c+=1
        Aavail=Aavail+A[i]
        Bavail=Bavail+B[i]
        Cavail=Cavail+C[i]



    if i<len(p)-1:
        i+=1
    else:
        i=0

print('need array is ')
for i in range(len(p)):
    print(p1[i],":",A_need[i],B_need[i],C_need[i])
print("Safe sequence of execution is")
print(seq)
```

Output:



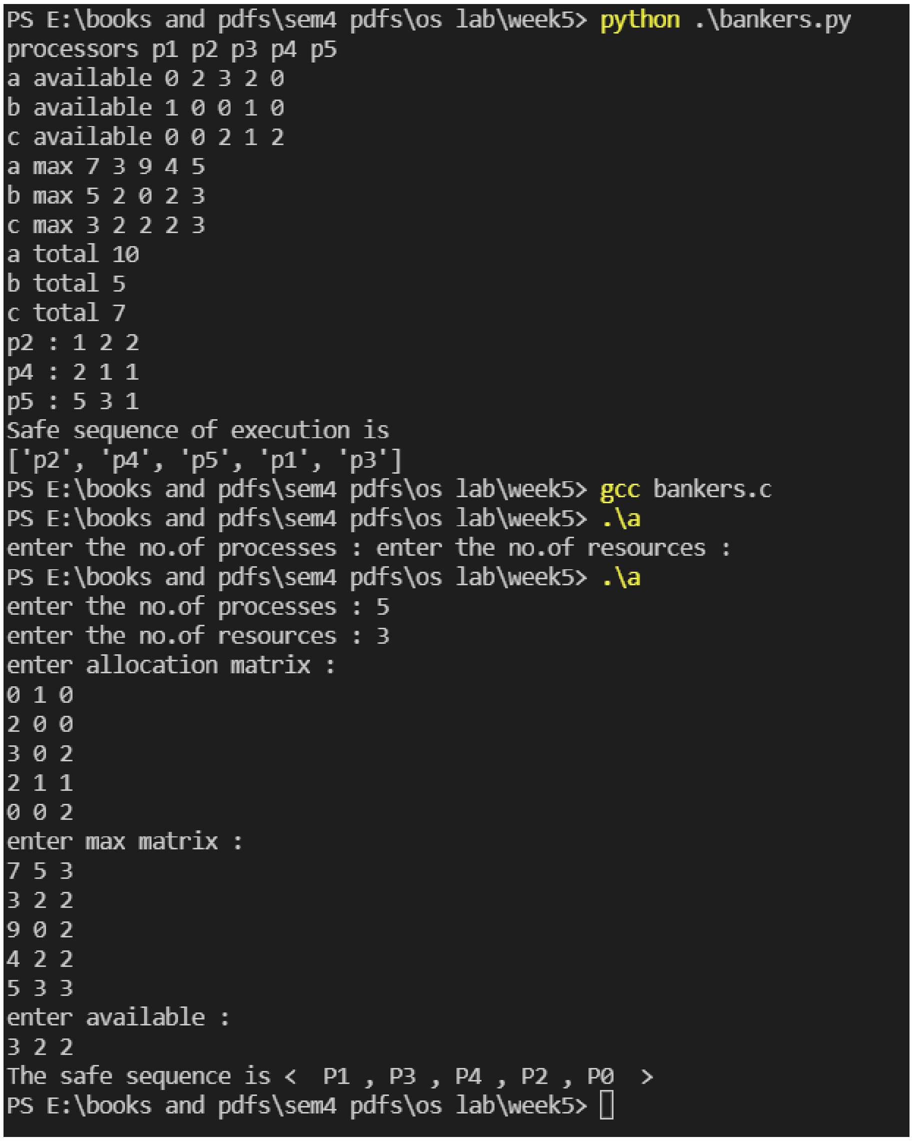
```
PS E:\books and pdfs\sem4 pdfs\os lab\week5> python .\bankers.py
processors p1 p2 p3 p4 p5
a available 0 2 3 2 0
b available 1 0 0 1 0
c available 0 0 2 1 2
a max 7 3 9 4 5
b max 5 2 0 2 3
c max 3 2 2 2 3
a total 10
b total 5
c total 7
need array is
p1 : 7 4 3
p2 : 1 2 2
p3 : 6 0 0
p4 : 2 1 1
p5 : 5 3 1
Safe sequence of execution is
['p2', 'p4', 'p5', 'p1', 'p3']
PS E:\books and pdfs\sem4 pdfs\os lab\week5>
```

Code2:

```
#include<stdio.h>
void main()
{
    int n,r,i,j,k,y=0,m=0;
    printf("enter the no.of processes : ");
    scanf("%d",&n);
    printf("enter the no.of resources : ");
    scanf("%d",&r);
    int
allocation[n][r],maximum[n][r],available[r],need_matrix[n][r],f1[n],sequenc
e[n];
    printf("enter allocation matrix : \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
           scanf("%d",&allocation[i][j]);
    }
    printf("enter max matrix : \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
           scanf("%d",&maximum[i][j]);
    }
    printf("enter available : \n");
    for(i=0;i<r;i++)
       scanf("%d",&available[i]);
    for(k=0;k<n;k++)
       f1[k]=0;
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
           need_matrix[i][j] = maximum[i][j] - allocation[i][j];
    }
    for(k=0;k<n;k++)
    {
        for(i=0;i<n;i++)
        {
            if(f1[i]==0)
            {
                int flag = 0;
                for(j=0;j<r;j++)
                {
                    if(need_matrix[i][j] > available[j])
                    {
                        flag = 1;
                        break;
                    }
                }
                if(flag == 0)
                {
                    sequence[m++]=i;
                    for(y=0;y<r;y++)
                        available[y] += allocation[i][y];
                    f1[i]=1;
                }
            }
        }
    }
    printf("The safe sequence is < ");
```

```
     for(i=0;i<n-1;i++)
        printf(" P%d ,",sequence[i]);
     printf(" P%d  >\n",sequence[n-1]);
 }
```

Output:



```
PS E:\books and pdfs\sem4 pdfs\os lab\week5> python .\bankers.py
processors p1 p2 p3 p4 p5
a available 0 2 3 2 0
b available 1 0 0 1 0
c available 0 0 2 1 2
a max 7 3 9 4 5
b max 5 2 0 2 3
c max 3 2 2 2 3
a total 10
b total 5
c total 7
p2 : 1 2 2
p4 : 2 1 1
p5 : 5 3 1
Safe sequence of execution is
['p2', 'p4', 'p5', 'p1', 'p3']
PS E:\books and pdfs\sem4 pdfs\os lab\week5> gcc bankers.c
PS E:\books and pdfs\sem4 pdfs\os lab\week5> .\a
enter the no.of processes : enter the no.of resources :
PS E:\books and pdfs\sem4 pdfs\os lab\week5> .\a
enter the no.of processes : 5
enter the no.of resources : 3
enter allocation matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
enter max matrix :
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3
enter available :
3 2 2
The safe sequence is <  P1 , P3 , P4 , P2 , P0  >
PS E:\books and pdfs\sem4 pdfs\os lab\week5> 
```

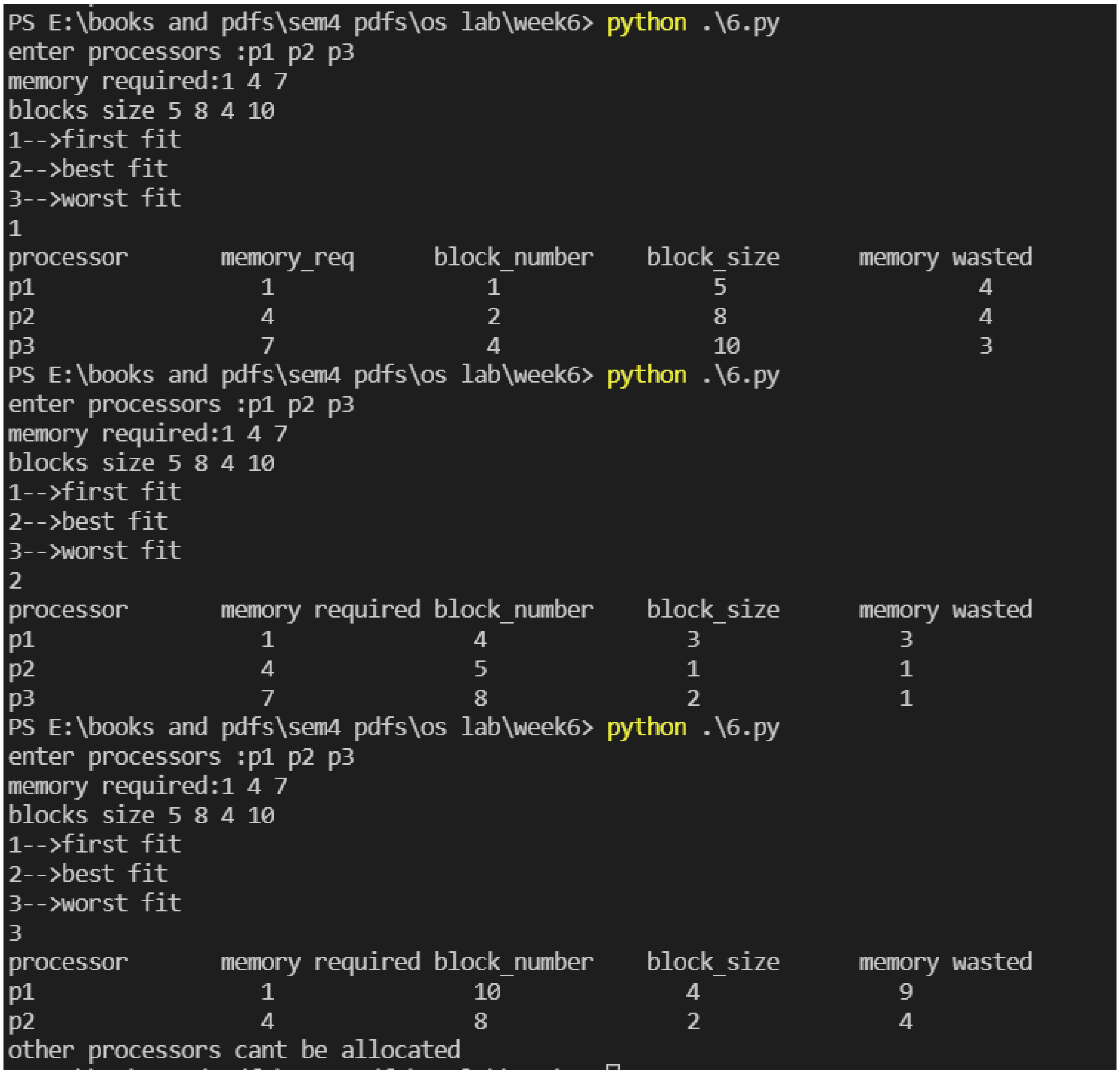
Week 6

Code:

```
def first_fit(p,mem_required,block_size):
    for i in range(len(p)):
        for j in range(len(block_size)):
            if(mem_required[i]<=block_size[j]):
                print(p[i],"\t\t  ",mem_required[i],"\t\t   ",j+1,"\t\t
",block_size[j],"   \t\t",block_size[j]-mem_required[i])
                block_size[j]=0
                break

def best_fit(p,mem_required,block_size):
    for i in range(len(p)):
        for j in range(len(block_size)):
            li=[]
            for k in range(len(block_size)):
                if mem_required[i]<=block_size[k]:
                    li.append([block_size[k],k])
            if li==[]:
                return

            li.sort()
            print(p[i],"\t\t  ",mem_required[i],"\t\t  ",li[0][0],"\t\t
",li[0][1]+1,"\t\t  ",li[0][0]-mem_required[i])
            block_size[li[0][1]]=0
            break

def worst_fit(p,mem_required,block_size):
    for i in range(len(p)):
        for j in range(len(block_size)):
            li=[]
            for k in range(len(block_size)):
                if mem_required[i]<=block_size[k]:
                    li.append([block_size[k],k])
            if li==[]:
                print("other processors cant be allocated")
                return

            li.sort()
            print(p[i],"\t\t  ",mem_required[i],"\t\t  ",li[len(li)-1][0],"\t\t
",li[len(li)-1][1]+1,"\t\t  ",li[len(li)-1][0]-mem_required[i])
            block_size[li[len(li)-1][1]]=0
            break

p=list(map(str,input('enter processors :').split()))
```

```
mem_required=list(map(int,input('memory required:').split()))
block_size=list(map(int,input('blocks size ').split()))
ch=int(input('1-->first fit \n2-->best fit \n3-->worst fit\n'))
if ch==1:
    print("processor\tmemory_req\tblock_number\tblock_size\tmemory wasted")
    first_fit(p,mem_required,block_size)
elif ch==2:
    print("processor\tmemory required\tblock_number\tblock_size\tmemory 
wasted")
    best_fit(p,mem_required,block_size)
elif ch==3:
    print("processor\tmemory required\tblock_number\tblock_size\tmemory 
wasted")
    worst_fit(p,mem_required,block_size)
```

Output:



```
PS E:\books and pdfs\sem4 pdfs\os lab\week6> python .\6.py
enter processors :p1 p2 p3
memory required:1 4 7
blocks size 5 8 4 10
1-->first fit
2-->best fit
3-->worst fit
1
processor       memory_req      block_number    block_size      memory wasted
p1                 1                1                5                   4
p2                 4                2                8                   4
p3                 7                4                10                  3
PS E:\books and pdfs\sem4 pdfs\os lab\week6> python .\6.py
enter processors :p1 p2 p3
memory required:1 4 7
blocks size 5 8 4 10
1-->first fit
2-->best fit
3-->worst fit
2
processor       memory required block_number    block_size      memory wasted
p1                 1               4               3               3
p2                 4               5               1               1
p3                 7               8               2               1
PS E:\books and pdfs\sem4 pdfs\os lab\week6> python .\6.py
enter processors :p1 p2 p3
memory required:1 4 7
blocks size 5 8 4 10
1-->first fit
2-->best fit
3-->worst fit
3
processor       memory required block_number    block_size      memory wasted
p1                 1               10              4               9
p2                 4               8               2               4
other processors cant be allocated
```