CPU Process Scheduling Algorithms

First Come First Serve

Code :

```
#non-premptive

#process delcartioon
class Process :

    def __init__(self,id,at,bt):

        self.id = id
        self.at = at
        self.bt = bt

    def set_ct(self,ct):

        self.ct = ct
        self.tt = self.ct - self.at
        self.wt = self.tt - self.bt

    def __str__(self):
        return f'{self.id}\t{self.at}\t{self.bt}\t{self.ct}\t{self.tt}\t{self.wt}'



#input
n = int(input("No of proccess:"))
l = []

for i in range(n):
    print(f"Process {i+1}")
    at = int(input("Arrival Time:"))
    bt = int(input("Burst Time:"))
    l.append(Process(i+1,at,bt))
    print()

#sort based on arrival
l.sort(key=lambda x:(x.at,x.id))


#fcfs
final = []
ct = 0

for process in l:
    if ct < process.at :
        ct = process.at
    ct+=process.bt

    process.set_ct(ct)
    final.append(process)  #append process to final
```

```
final.sort(key=lambda x:x.id)
print('ID\tAT\tBT\tCT\tTAT\tWT')
for process in final :
    print(process)


#averages
avg_ct = 0
avg_tt = 0
avg_wt = 0

for i in final:
    avg_ct+=i.ct
    avg_tt+=i.tt
    avg_wt+=i.wt

print(f'Average CT : {avg_ct/n}')
print(f'Average TT : {avg_tt/n}')
print(f'Average WT : {avg_wt/n}')
```

Output :

```
No of proccess:3
Process 1
Arrival Time:0
Burst Time:24

Process 2
Arrival Time:1
Burst Time:3

Process 3
Arrival Time:2
Burst Time:4

ID      AT      BT      CT      TAT     WT
1       0       24      24      24      0
2       1       3       27      26      23
3       2       4       31      29      25
Average CT : 27.333333333333332
Average TT : 26.333333333333332
Average WT : 16.0
```

Longest Job First

Code:

```
#non-premptive

#process delcartioon
class Process :
```

```
    def __init__(self,id,at,bt):

        self.id = id
        self.at = at
        self.bt = bt

    def set_ct(self,ct):

        self.ct = ct
        self.tt = self.ct - self.at
        self.wt = self.tt - self.bt

    def __str__(self):
        return f'{self.id}\t{self.at}\t{self.bt}\t{self.ct}\t{self.tt}\t{self.wt}'

#Queue Implementation

class Queue:

    def __init__(self):
        self.q = []

    def push(self,x):
        self.q.append(x)
        self.q.sort(key=lambda x:(-x.bt,x.at,x.id))


    def popout(self):
        self.q.pop(0)

    def front(self):
        return self.q[0]

    def is_empty(self):
        return len(self.q)<=0


#input
n = int(input("No of proccess:"))
l = []

for i in range(n):
    print(f"Process {i+1}")
    at = int(input("Arrival Time:"))
    bt = int(input("Burst Time:"))
    l.append(Process(i+1,at,bt))
    print()

#sort based on arrival
l.sort(key=lambda x:x.at)


q=Queue()
final=[]

#intial step
time = l[0].at
while len(l) and l[0].at<=time:
```

```
        q.push(l[0])
        l.remove(l[0])


#ljf
while not q.is_empty():

    cur = q.front()
    q.popout()

    if time<cur.at :
        time = cur.at

    time += cur.bt
    cur.set_ct(time)
    final.append(cur)

    while len(l) and l[0].at<=time:
        q.push(l[0])
        l.remove(l[0])



final.sort(key=lambda x:x.id)
print('ID\tAT\tBT\tCT\tTAT\tWT')
for process in final :
    print(process)


#averages
avg_ct = 0
avg_tt = 0
avg_wt = 0

for i in final:
    avg_ct+=i.ct
    avg_tt+=i.tt
    avg_wt+=i.wt

print(f'Average CT : {avg_ct/n}')
print(f'Average TT : {avg_tt/n}')
print(f'Average WT : {avg_wt/n}')
```

Output:

```
Process 1
Arrival Time:1
Burst Time:1

Process 2
Arrival Time:2
Burst Time:4

Process 3
Arrival Time:3
Burst Time:6

Process 4
Arrival Time:2
Burst Time:3

ID      AT      BT      CT      TAT     WT
1       1       1       2       1       0
2       2       4       6       4       0
3       3       6       12      9       3
4       2       3       15      13      10
Average CT : 8.75
Average TT : 6.75
Average WT : 3.25
```

Shortest Job First

Code:

```
#non-premptive

#process delcartioon
class Process :

    def __init__(self,id,at,bt):

        self.id = id
        self.at = at
        self.bt = bt

    def set_ct(self,ct):

        self.ct = ct
        self.tt = self.ct - self.at
        self.wt = self.tt - self.bt

    def __str__(self):
        return f'{self.id}\t{self.at}\t{self.bt}\t{self.ct}\t{self.tt}\t{self.wt}'

#Queue Implementation

class Queue:

    def __init__(self):
        self.q = []
```

```
    def push(self,x):
        self.q.append(x)
        self.q.sort(key=lambda x:(x.bt,x.at,x.id))

    def popout(self):
        self.q.pop(0)

    def front(self):
        return self.q[0]

    def is_empty(self):
        return len(self.q)<=0


#input
n = int(input("No of proccess:"))
l = []

for i in range(n):
    print(f"Process {i+1}")
    at = int(input("Arrival Time:"))
    bt = int(input("Burst Time:"))
    l.append(Process(i+1,at,bt))
    print()

#sort based on arrival
l.sort(key=lambda x:x.at)


q=Queue()
final=[]

#intial step
time = l[0].at
while len(l) and l[0].at<=time:
        q.push(l[0])
        l.remove(l[0])


#sjf
while not q.is_empty():

    cur = q.front()
    q.popout()

    if time<cur.at :
        time = cur.at

    time += cur.bt
    cur.set_ct(time)
    final.append(cur)

    while len(l) and l[0].at<=time:
        q.push(l[0])
        l.remove(l[0])
```

```
final.sort(key=lambda x:x.id)
print('ID\tAT\tBT\tCT\tTAT\tWT')
for process in final :
    print(process)


#averages
avg_ct = 0
avg_tt = 0
avg_wt = 0

for i in final:
    avg_ct+=i.ct
    avg_tt+=i.tt
    avg_wt+=i.wt

print(f'Average CT : {avg_ct/n}')
print(f'Average TT : {avg_tt/n}')
print(f'Average WT : {avg_wt/n}')
```

Output:

```
Process 1
Arrival Time:0
Burst Time:6

Process 2
Arrival Time:0
Burst Time:8

Process 3
Arrival Time:0
Burst Time:7

Process 4
Arrival Time:0
Burst Time:3

ID      AT      BT      CT      TAT     WT
1       0       6       9       9       3
2       0       8       24      24      16
3       0       7       16      16      9
4       0       3       3       3       0
Average CT : 13.0
Average TT : 13.0
Average WT : 7.0
```

Highest Response Ratio Next

Code :

```
#non-premptive

#process delcartioon
class Process :

    def __init__(self,id,at,bt):

        self.id = id
        self.at = at
        self.bt = bt
        self.wt = 0

    def set_ct(self,ct):

        self.ct = ct
        self.tt = self.ct - self.at
        self.wt = self.tt - self.bt

    def __str__(self):
        return f'{self.id}\t{self.at}\t{self.bt}\t{self.ct}\t{self.tt}\t{self.wt}'


def rr(x):
     return (x.wt+x.bt)/x.bt

#queue implementation
class Queue:

    def __init__(self):
        self.q = []

    def push(self,x):
        self.q.append(x)

    def popout(self):
        self.q.pop(0)

    def front(self):
        return self.q[0]

    def is_empty(self):
        return len(self.q)<=0

    def update(self):
        self.q.sort(key=lambda x:(-rr(x),x.at,x.id))



#input
n = int(input("No of proccess:"))
l = []

for i in range(n):
    print(f"Process {i+1}")
    at = int(input("Arrival Time:"))
    bt = int(input("Burst Time:"))
```

```
    l.append(Process(i+1,at,bt))
    print()

#sort based on arrival
l.sort(key=lambda x:x.at)




#hrrn

q=Queue()
final=[]

#intial step
time = l[0].at
while len(l) and l[0].at<=time:
        q.push(l[0])
        l.remove(l[0])
q.update()


#hrrn
while not q.is_empty():

    cur = q.front()
    q.popout()

    if time<cur.at :
        time = cur.at

    time += cur.bt
    cur.set_ct(time)
    final.append(cur)

    while len(l) and l[0].at<=time:
        q.push(l[0])
        l.remove(l[0])

    #re-updation
    for process in q.q:
        process.wt = time - process.at
    q.update()




final.sort(key=lambda x:x.id)
print('ID\tAT\tBT\tCT\tTAT\tWT')
for process in final :
    print(process)


#averages
avg_ct = 0
avg_tt = 0
```

```
avg_wt = 0

for i in final:
    avg_ct+=i.ct
    avg_tt+=i.tt
    avg_wt+=i.wt

print(f'Average CT : {avg_ct/n}')
print(f'Average TT : {avg_tt/n}')
print(f'Average WT : {avg_wt/n}')
```

Output :

```
No of proccess:5
Process 1
Arrival Time:0
Burst Time:3

Process 2
Arrival Time:2
Burst Time:6

Process 3
Arrival Time:4
Burst Time:4

Process 4
Arrival Time:6
Burst Time:5

Process 5
Arrival Time:8
Burst Time:2

ID      AT      BT      CT      TAT     WT
1       0       3       3       3       0
2       2       6       9       7       1
3       4       4       13      9       5
4       6       5       20      14      9
5       8       2       15      7       5
Average CT : 12.0
Average TT : 8.0
Average WT : 4.0
```

Longest Remaining Job First

Code:

```
#premptive

#process delcartioon
class Process :
```

```
    def __init__(self,id,at,bt):

        self.id = id
        self.at = at
        self.bt = bt
        self.rt = bt

    def set_ct(self,ct):

        self.ct = ct
        self.tt = self.ct - self.at
        self.wt = self.tt - self.bt

    def __str__(self):
        return f'{self.id}\t{self.at}\t{self.bt}\t{self.ct}\t{self.tt}\t{self.wt}'

#queue implementation
class Queue:

    def __init__(self):
        self.q = []

    def push(self,x):
        self.q.append(x)
        self.update()

    def popout(self):
        self.q.pop(0)

    def front(self):
        if(len(self.q)):
            return self.q[0]
        else:
            return None

    def is_empty(self):
        return len(self.q)<=0

    def update(self):
        self.q.sort(key=lambda x:(-x.rt,x.at,x.id))


#input
n = int(input("No of proccess:"))
l = []

for i in range(n):
    print(f"Process {i+1}")
    at = int(input("Arrival Time:"))
    bt = int(input("Burst Time:"))
    l.append(Process(i+1,at,bt))
    print()

#sort based on arrival
l.sort(key=lambda x:x.at)
```

```
#lrjf

q=Queue()
final=[]

#intial step
time = l[0].at
while len(l) and l[0].at<=time:
        q.push(l[0])
        l.remove(l[0])



while not q.is_empty():

    cur = q.front()
    q.popout()

    if time<cur.at :
        time = cur.at

    time+=1
    cur.rt-=1

    if(cur.rt==0) :
        cur.set_ct(time)
        print("Time set")
        final.append(cur)
    else:
        q.push(cur)
        while len(l) and l[0].at<=time:
            q.push(l[0])
            l.remove(l[0])



final.sort(key=lambda x:x.id)
print('ID\tAT\tBT\tCT\tTAT\tWT')
for process in final :
    print(process)


#averages
avg_ct = 0
avg_tt = 0
avg_wt = 0

for i in final:
    avg_ct+=i.ct
    avg_tt+=i.tt
    avg_wt+=i.wt

print(f'Average CT : {avg_ct/n}')
print(f'Average TT : {avg_tt/n}')
print(f'Average WT : {avg_wt/n}')
```

Output:

```
No of proccess:4
Process 1
Arrival Time:1
Burst Time:2

Process 2
Arrival Time:2
Burst Time:4

Process 3
Arrival Time:3
Burst Time:6

Process 4
Arrival Time:4
Burst Time:8

Time set
Time set
Time set
Time set
ID      AT      BT      CT      TAT     WT
1       1       2       18      17      15
2       2       4       19      17      13
3       3       6       20      17      11
4       4       8       21      17      9
Average CT : 19.5
Average TT : 17.0
Average WT : 12.0
```

Shortest Remaining Job First

Code:

```
#premptive

#process delcartioon
class Process :

    def __init__(self,id,at,bt):

        self.id = id
        self.at = at
        self.bt = bt
        self.rt = bt

    def set_ct(self,ct):

        self.ct = ct
        self.tt = self.ct - self.at
```

```
        self.wt = self.tt - self.bt

    def __str__(self):
        return f'{self.id}\t{self.at}\t{self.bt}\t{self.ct}\t{self.tt}\t{self.wt}'

#queue implementation
class Queue:

    def __init__(self):
        self.q = []

    def push(self,x):
        self.q.append(x)
        self.update()

    def popout(self):
        self.q.pop(0)

    def front(self):
            return self.q[0]


    def is_empty(self):
        return len(self.q)<=0

    def update(self):
        self.q.sort(key=lambda x:(x.rt,x.at,x.id))

#input
n = int(input("No of proccess:"))
l = []

for i in range(n):
    print(f"Process {i+1}")
    at = int(input("Arrival Time:"))
    bt = int(input("Burst Time:"))
    l.append(Process(i+1,at,bt))
    print()

#sort based on arrival
l.sort(key=lambda x:x.at)


#srjf

q=Queue()
final=[]

#intial step
time = l[0].at
while len(l) and l[0].at<=time:
        q.push(l[0])
        l.remove(l[0])


while not q.is_empty():
```

```
    cur = q.front()
    q.popout()

    if time<cur.at :
        time = cur.at

    time+=1
    cur.rt-=1


    if(cur.rt==0) :
        cur.set_ct(time)
        final.append(cur)
    else:
        q.push(cur)
        while len(l) and l[0].at<=time:
            q.push(l[0])
            l.remove(l[0])



final.sort(key=lambda x:x.id)
print('ID\tAT\tBT\tCT\tTAT\tWT')
for process in final :
    print(process)


#averages
avg_ct = 0
avg_tt = 0
avg_wt = 0

for i in final:
    avg_ct+=i.ct
    avg_tt+=i.tt
    avg_wt+=i.wt

print(f'Average CT : {avg_ct/n}')
print(f'Average TT : {avg_tt/n}')
print(f'Average WT : {avg_wt/n}')
```

Output:

```
Process 1
Arrival Time:2
Burst Time:6

Process 2
Arrival Time:5
Burst Time:2

Process 3
Arrival Time:1
Burst Time:8

Process 4
Arrival Time:0
Burst Time:3

Process 5
Arrival Time:4
Burst Time:4

ID      AT      BT      CT      TAT     WT
1       2       6       15      13      7
2       5       2       7       2       0
3       1       8       23      22      14
4       0       3       3       3       0
5       4       4       10      6       2
Average CT : 11.6
Average TT : 9.2
Average WT : 4.6
```

Round Robin

Code:

```
#premptive
import queue
#process delcartioon
class Process :

    def __init__(self,id,at,bt):

        self.id = id
        self.at = at
        self.bt = bt
        self.rt = bt

    def set_ct(self,ct):

        self.ct = ct
        self.tt = self.ct - self.at
        self.wt = self.tt - self.bt

    def __str__(self):
        return f'{self.id}\t{self.at}\t{self.bt}\t{self.ct}\t{self.tt}\t{self.wt}'
```

```
q=queue.Queue()
time_quantumn = 3

#input
n = int(input("No of proccess:"))
l = []

for i in range(n):
    print(f"Process {i+1}")
    at = int(input("Arrival Time:"))
    bt = int(input("Burst Time:"))
    l.append(Process(i+1,at,bt))
    print()

#sort based on arrival
l.sort(key=lambda x:x.at)

final=[]

#intial step
time = l[0].at
while len(l) and l[0].at<=time:
        q.put(l[0])
        l.remove(l[0])


while not q.empty():

    cur = q.get()

    if time<cur.at :
        time = cur.at

    next_time = time + time_quantumn

    while time < next_time and cur.rt!=0 :
        time+=1
        cur.rt-=1
        while len(l) and l[0].at<=time:
            q.put(l[0])
            l.remove(l[0])

    if(not cur.rt):
        cur.set_ct(time)
        final.append(cur)
    else:
        q.put(cur)



final.sort(key=lambda x:x.id)
print('ID\tAT\tBT\tCT\tTAT\tWT')
for process in final :
    print(process)
```

```
#averages
avg_ct = 0
avg_tt = 0
avg_wt = 0

for i in final:
    avg_ct+=i.ct
    avg_tt+=i.tt
    avg_wt+=i.wt

print(f'Average CT : {avg_ct/n}')
print(f'Average TT : {avg_tt/n}')
print(f'Average WT : {avg_wt/n}')
```

Output :

```
No of proccess:4
Process 1
Arrival Time:0
Burst Time:5

Process 2
Arrival Time:1
Burst Time:4

Process 3
Arrival Time:2
Burst Time:2

Process 4
Arrival Time:4
Burst Time:1

ID      AT      BT      CT      TAT     WT
1       0       5       12      12      7
2       1       4       11      10      6
3       2       2       6       4       2
4       4       1       9       5       4
Average CT : 9.5
Average TT : 7.75
Average WT : 4.75
```

Priority Based

Code:

```
#premptive
#less the number higher the priority

#process delcartioon
class Process :
```

```
    def __init__(self,id,at,bt,prior):

        self.id = id
        self.at = at
        self.bt = bt
        self.rt = bt
        self.prior = prior

    def set_ct(self,ct):

        self.ct = ct
        self.tt = self.ct - self.at
        self.wt = self.tt - self.bt

    def __str__(self):
        return f'{self.id}\t{self.at}\t{self.bt}\t{self.ct}\t{self.tt}\t{self.wt}'

#queue implementation
class Queue:

    def __init__(self):
        self.q = []

    def push(self,x):
        self.q.append(x)
        self.update()

    def popout(self):
        self.q.pop(0)


    def front(self):
            return self.q[0]


    def is_empty(self):
        return len(self.q)<=0

    def update(self):
        self.q.sort(key=lambda x:(x.prior,x.at,x.id))



#input
n = int(input("No of proccess:"))
l = []

for i in range(n):
    print(f"Process {i+1}")
    at = int(input("Arrival Time:"))
    bt = int(input("Burst Time:"))
    prior = int(input("Enter the priority:"))
    l.append(Process(i+1,at,bt,prior))
    print()

#sort based on arrival
l.sort(key=lambda x:x.at)
```

```
#prior
q=Queue()
final=[]

#intial step
time = l[0].at
while len(l) and l[0].at<=time:
        q.push(l[0])
        l.remove(l[0])


while not q.is_empty():

    cur = q.front()
    q.popout()

    if time<cur.at :
        time = cur.at

    time+=1
    cur.rt-=1


    if(cur.rt==0) :
        cur.set_ct(time)
        final.append(cur)
    else:
        q.push(cur)
        while len(l) and l[0].at<=time:
            q.push(l[0])
            l.remove(l[0])


final.sort(key=lambda x:x.id)
print('ID\tAT\tBT\tCT\tTAT\tWT')
for process in final :
    print(process)


#averages
avg_ct = 0
avg_tt = 0
avg_wt = 0

for i in final:
    avg_ct+=i.ct
    avg_tt+=i.tt
    avg_wt+=i.wt

print(f'Average CT : {avg_ct/n}')
print(f'Average TT : {avg_tt/n}')
print(f'Average WT : {avg_wt/n}')
```

Output:

```
No of proccess:5
Process 1
Arrival Time:0
Burst Time:3
Enter the priority:3

Process 2
Arrival Time:1
Burst Time:4
Enter the priority:2

Process 3
Arrival Time:2
Burst Time:6
Enter the priority:4

Process 4
Arrival Time:3
Burst Time:4
Enter the priority:6

Process 5
Arrival Time:5
Burst Time:2
Enter the priority:10

ID      AT      BT      CT      TAT     WT
1       0       3       7       7       4
2       1       4       5       4       0
3       2       6       13      11      5
4       3       4       17      14      10
5       5       2       19      14      12
Average CT : 12.2
Average TT : 10.0
Average WT : 6.2
```