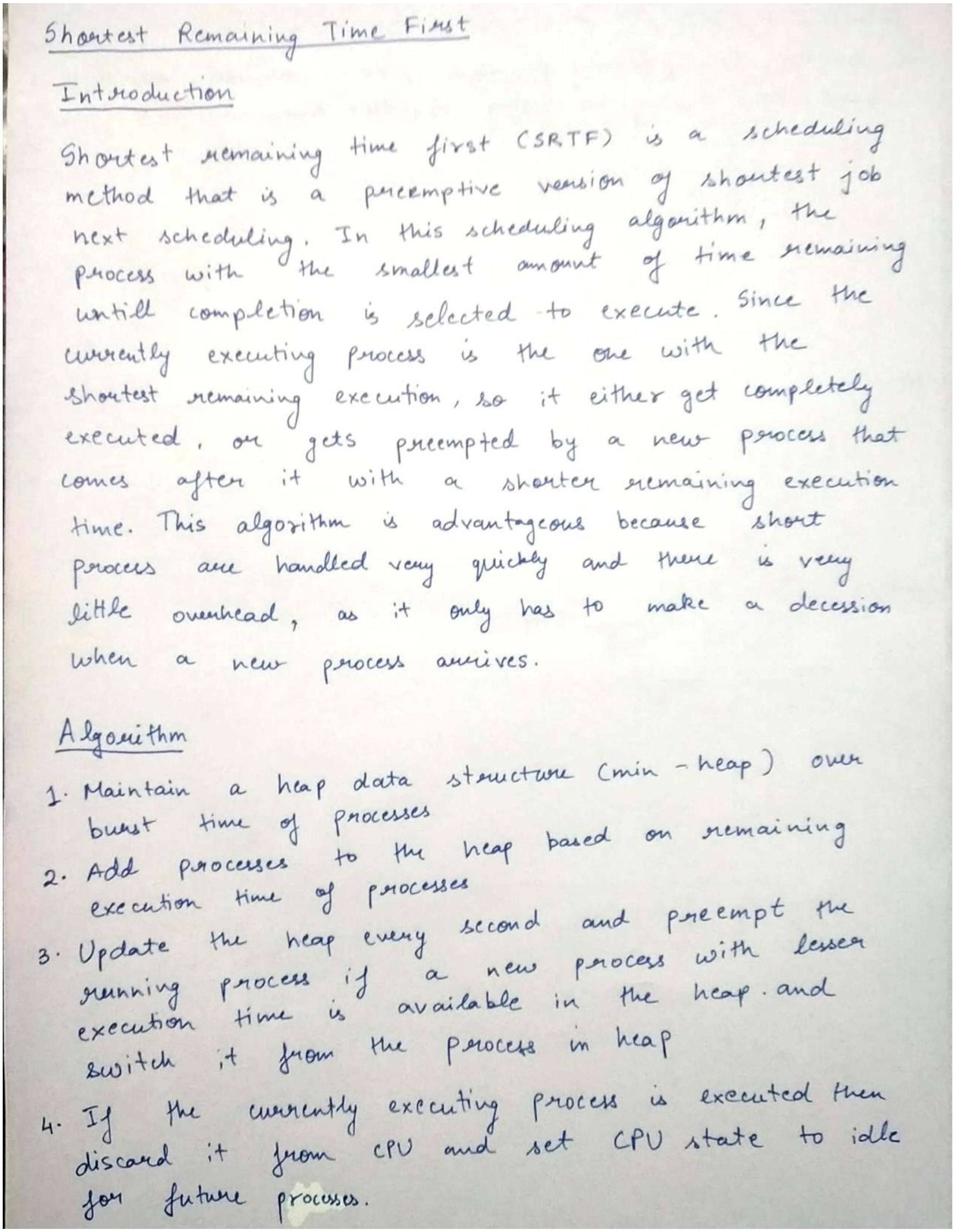
Experiment – 3



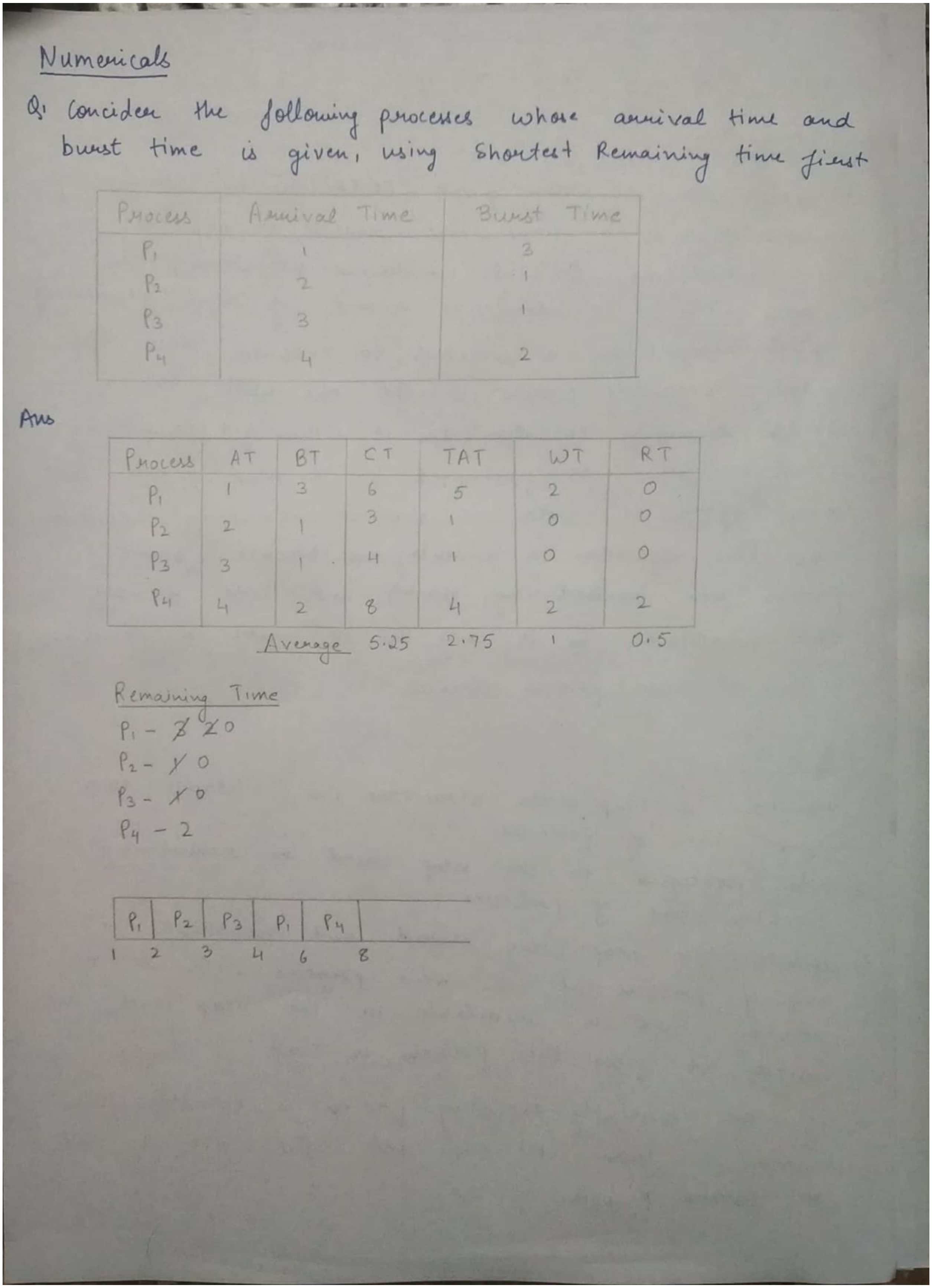
Shortest Remaining Time First

Introduction

Shortest remaining time first (SRTF) is a scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining untill completion is selected to execute. Since the currently executing process is the one with the shortest remaining execution, so it either get completely executed, or gets preempted by a new process that comes after it with a shorter remaining execution time. This algorithm is advantageous because short process are handled very quickly and there is very little overhead, as it only has to make a decession when a new process arrives.

Algorithm

1. Maintain a heap data structure (min-heap) over burst time of processes
2. Add processes to the heap based on remaining execution time of processes
3. Update the heap every second and preempt the running process if a new process with lesser execution time is available in the heap. and switch it from the process in heap
4. If the currently executing process is executed then discard it from CPU and set CPU state to idle for future processes.



Numericals

Q: Concider the following processes whose arrival time and burst time is given, using Shortest Remaining time first

Process	Arrival Time	Burst Time
P₁	1	3
P₂	2	1
P₃	3	1
P₄	4	2

Ans

Process	AT	BT	CT	TAT	WT	RT
P₁	1	3	6	5	2	0
P₂	2	1	3	1	0	0
P₃	3	1	4	1	0	0
P₄	4	2	8	4	2	2
		Average	5.25	2.75	1	0.5

Remaining Time

P₁ - ~~3~~ ~~2~~ 0

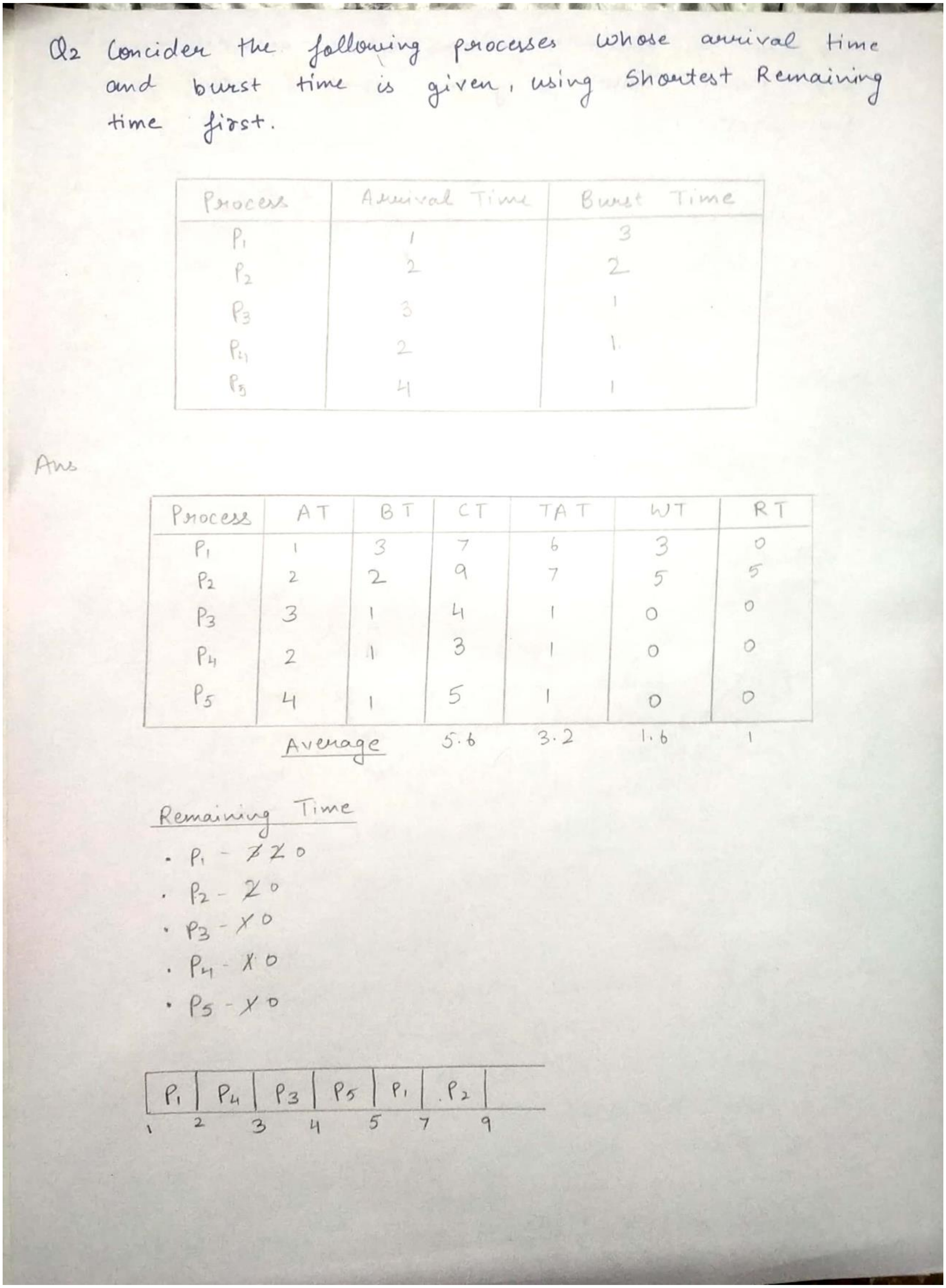
P₂ - ~~1~~ 0

P₃ - ~~1~~ 0

P₄ - 2

P₁	P₂	P₃	P₁	P₄

1 2 3 4 6 8



Q2 Concider the following processes whose arrival time and burst time is given, using Shortest Remaining time first.

Process	Arrival Time	Burst Time
P_1	1	3
P_2	2	2
P_3	3	1
P_4	2	1
P_5	4	1

Ans

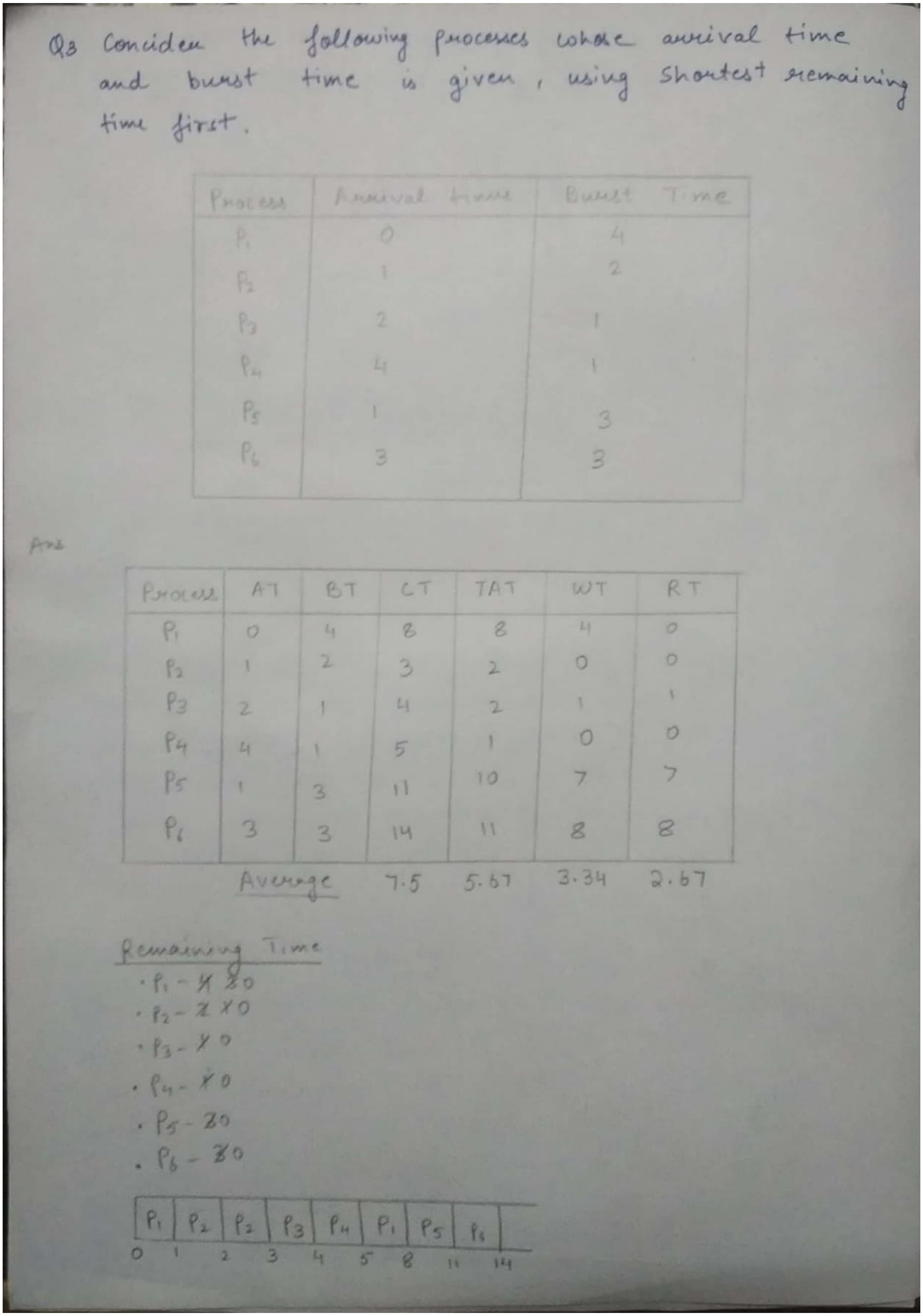
Process	AT	BT	CT	TAT	WT	RT
P_1	1	3	7	6	3	0
P_2	2	2	9	7	5	5
P_3	3	1	4	1	0	0
P_4	2	1	3	1	0	0
P_5	4	1	5	1	0	0
	Average		5.6	3.2	1.6	1

Remaining Time

- P_1 - ~~3~~ ~~2~~ 0
- P_2 - ~~2~~ 0
- P_3 - ~~1~~ 0
- P_4 - ~~1~~ 0
- P_5 - ~~1~~ 0

P_1	P_4	P_3	P_5	P_1	P_2

1 2 3 4 5 7 9



Q3 Concider the following processes whose arrival time and burst time is given, using Shortest remaining time first.

Process	Arrival time	Burst Time
P1	0	4
P2	1	2
P3	2	1
P4	4	1
P5	1	3
P6	3	3

Ans

Process	AT	BT	CT	TAT	WT	RT
P1	0	4	8	8	4	0
P2	1	2	3	2	0	0
P3	2	1	4	2	1	1
P4	4	1	5	1	0	0
P5	1	3	11	10	7	7
P6	3	3	14	11	8	8
	Average		7.5	5.67	3.34	2.67

Remaining Time

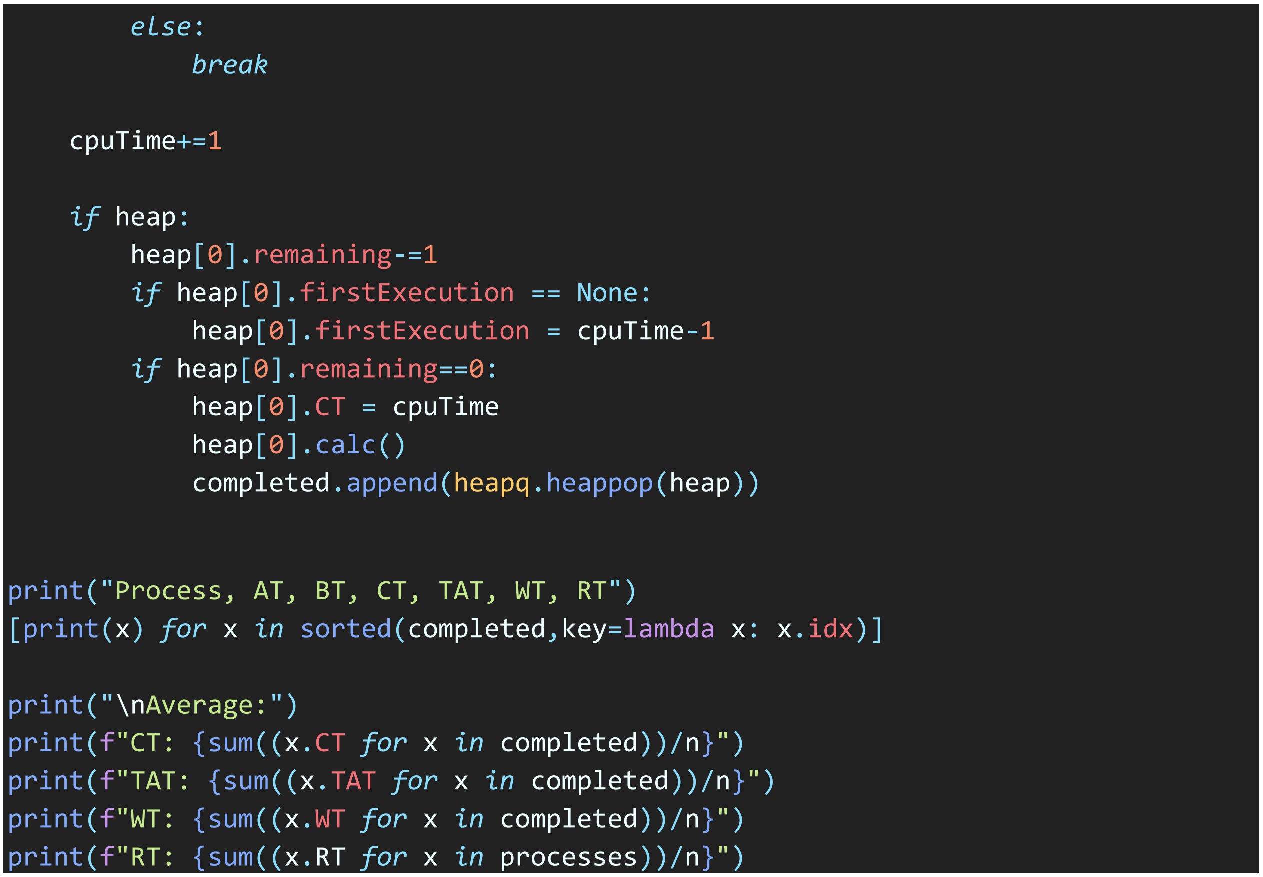
- P1 - ~~4~~ ~~3~~ 0
- P2 - ~~2~~ ~~1~~ 0
- P3 - ~~1~~ 0
- P4 - ~~1~~ 0
- P5 - ~~3~~ 0
- P6 - ~~3~~ 0

P1	P2	P2	P3	P4	P1	P5	P6

0 1 2 3 4 5 8 11 14

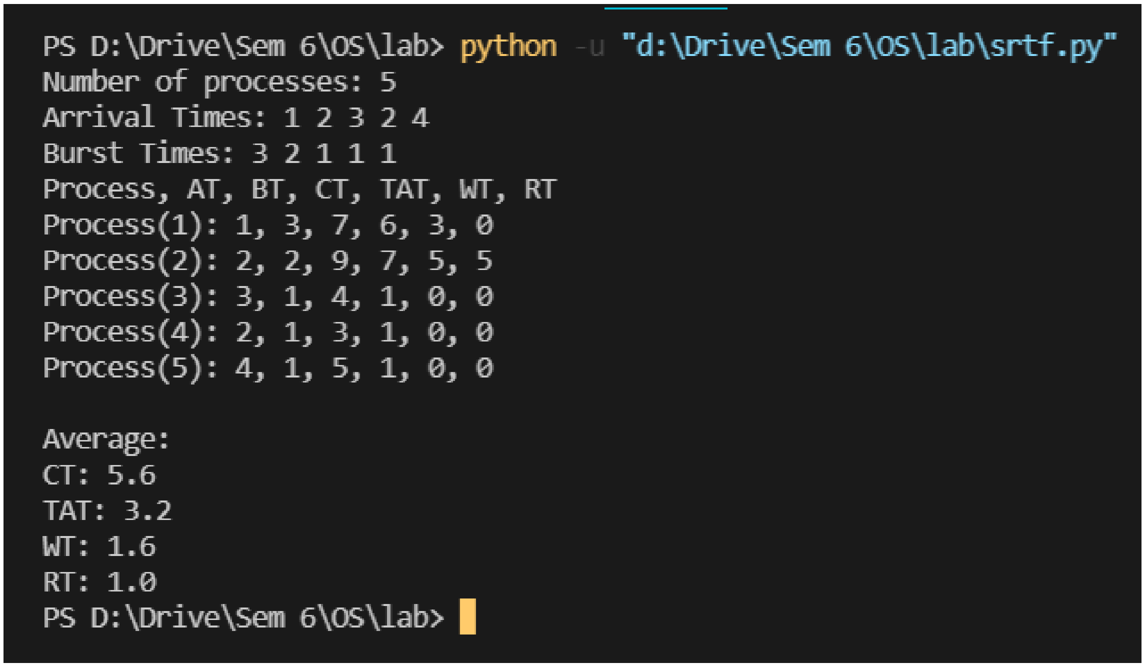
Code

```
import heapq

class Process:
    def __init__(self, idx, AT, BT,) -> None:
        self.idx = idx
        self.AT = AT
        self.BT = BT
        self.remaining = BT
        self.CT = None
        self.firstExecution = None

    def calc(self):
        self.TAT = self.CT - self.AT
        self.WT = self.TAT - self.BT
        self.RT = self.firstExecution - self.AT

    def __lt__(self,other):
        if self.remaining==other.remaining:
            return self.AT<other.AT
        return self.remaining<other.remaining

    def __repr__(self) -> str:
        return f"Process({self.idx}): {self.AT}, {self.BT}, {self.CT}, {self.TAT}, {self.WT}, {self.RT}"

n = int(input("Number of processes: "))
arrivalTime = list(map(int, input("Arrival Times: ").split()))
burstTime = list(map(int, input("Burst Times: ").split()))

processes = sorted([Process(x+1,arrivalTime[x],burstTime[x]) for x in range(len(arrivalTime))],key=lambda x:x.AT)
processID = 0
heap = []
completed = []
cpuTime = processes[0].AT

while len(completed)<n:
    print(cpuTime,completed,heap,"\n")
    for p in range(processID,n):
        p = processes[p]
        if(p.AT<=cpuTime):
            heapq.heappush(heap,p)
            processID+=1
```



```
        else:
            break

    cpuTime+=1

    if heap:
        heap[0].remaining-=1
        if heap[0].firstExecution == None:
            heap[0].firstExecution = cpuTime-1
        if heap[0].remaining==0:
            heap[0].CT = cpuTime
            heap[0].calc()
            completed.append(heapq.heappop(heap))


print("Process, AT, BT, CT, TAT, WT, RT")
[print(x) for x in sorted(completed,key=lambda x: x.idx)]

print("\nAverage:")
print(f"CT: {sum((x.CT for x in completed))/n}")
print(f"TAT: {sum((x.TAT for x in completed))/n}")
print(f"WT: {sum((x.WT for x in completed))/n}")
print(f"RT: {sum((x.RT for x in processes))/n}")
```

Output



```
PS D:\Drive\Sem 6\OS\lab> python -u "d:\Drive\Sem 6\OS\lab\srtf.py"
Number of processes: 5
Arrival Times: 1 2 3 2 4
Burst Times: 3 2 1 1 1
Process, AT, BT, CT, TAT, WT, RT
Process(1): 1, 3, 7, 6, 3, 0
Process(2): 2, 2, 9, 7, 5, 5
Process(3): 3, 1, 4, 1, 0, 0
Process(4): 2, 1, 3, 1, 0, 0
Process(5): 4, 1, 5, 1, 0, 0

Average:
CT: 5.6
TAT: 3.2
WT: 1.6
RT: 1.0
PS D:\Drive\Sem 6\OS\lab>
```