4. Write a C/Python program to calculate average waiting time and Turnaround Time of n processes with First Come First Serve (FCFS) CPU scheduling algorithm.

```
n = int(input("No. of processes: "))
bt = list(map(int, input("Burst times: ").split()))
at = list(map(int, input("Arrival times: ").split()))

wt = [0]*n
tat = [0]*n
ct = [0]*n
p = sorted(range(n), key=lambda i: at[i])

t = 0
for i in p:
    t = max(t, at[i]) + bt[i]
    ct[i] = t
    tat[i] = ct[i] - at[i]
    wt[i] = tat[i] - bt[i]

print("\nPID\tAT\tBT\tWT\tTAT")
for i in range(n):
    print(i+1, "\t", at[i], "\t", bt[i], "\t", wt[i], "\t", tat[i])
print("\nAvg WT:", sum(wt)/n, " Avg TAT:", sum(tat)/n)
```

```
No. of processes: 3
Burst times: 2 3 4
Arrival times: 0 1 2 3

PID     AT      BT      WT      TAT
1        0       2       0       2
2        1       3       1       4
3        2       4       3       7

Avg WT: 1.3333333333333333  Avg TAT: 4.333333333333333
PS C:\Users\Th3\Desktop\test>
```

5.Write a C/Python program to calculate average waiting time and Turnaround Time of n processes with Shortest Job First (SJF) CPU scheduling algorithm.(non preemptive)

```
n = int(input("No. of processes: "))
bt = list(map(int, input("Burst times: ").split()))
at = list(map(int, input("Arrival times: ").split()))

wt = [0]*n; tat = [0]*n; done=[0]*n
t = 0; done_cnt = 0
```

```
while done_cnt < n:
    idx = -1
    for i in range(n):
        if not done[i] and at[i] <= t:
            if idx == -1 or bt[i] < bt[idx]:
                idx = i
    if idx == -1: t += 1; continue
    t += bt[idx]; tat[idx] = t - at[idx]; wt[idx] = tat[idx] - bt[idx]
    done[idx] = 1; done_cnt += 1

print("\nPID\tAT\tBT\tWT\tTAT")
for i in range(n):
    print(i+1,"\t",at[i],"\t",bt[i],"\t",wt[i],"\t",tat[i])
print("\nAvg WT:", sum(wt)/n, " Avg TAT:", sum(tat)/n)
```

```
No. of processes: 3
Burst times: 3 2 1
Arrival times: 2 3 1

PID     AT      BT      WT      TAT
1        2       3       0       3
2        3       2       2       4
3        1       1       0       1

Avg WT: 0.6666666666666666  Avg TAT: 2.6666666666666665
```

6.Write a C/Python program to calculate average waiting time and Turnaround Time of n processes with Priority CPU scheduling algorithm.(non preemptive)

```
n = int(input("No. of processes: "))
bt = list(map(int, input("Burst times: ").split()))
at = list(map(int, input("Arrival times: ").split()))
pr = list(map(int, input("Priorities: ").split()))

wt = [0]*n; tat = [0]*n; done=[0]*n
t = 0; done_cnt = 0

while done_cnt < n:
    idx = -1
    for i in range(n):
        if not done[i] and at[i] <= t:
            if idx == -1 or pr[i] < pr[idx]:
                idx = i
    if idx == -1: t += 1; continue
    t += bt[idx]; tat[idx] = t - at[idx]; wt[idx] = tat[idx] - bt[idx]
    done[idx]=1; done_cnt+=1
```

```
print("\nPID\tAT\tBT\tPR\tWT\tTAT")
for i in range(n):
    print(i+1,"\t",at[i],"\t",bt[i],"\t",pr[i],"\t",wt[i],"\t",tat[i])
print("\nAvg WT:", sum(wt)/n, " Avg TAT:", sum(tat)/n)

print("Turnaround Times:", tat)
print("Average Waiting Time:", sum(wt)/n)
print("Average Turnaround Time:", sum(tat)/n)
```

```
No. of processes: 3
Burst times: 3 2 3
Arrival times: 2 3 2
Priorities: 3 1 2

PID     AT      BT      PR      WT      TAT
1        2       3       3       5       8
2        3       2       1       2       4
3        2       3       2       0       3

Avg WT: 2.3333333333333335  Avg TAT: 5.0
```

7.Write a C/Python program to calculate average waiting time and Turnaround Time of n processes with Round Robin (RR) CPU

```
n = int(input("No. of processes: "))
bt = list(map(int, input("Burst times: ").split()))
at = list(map(int, input("Arrival times: ").split()))
q = int(input("Time quantum: "))

rt = bt.copy(); wt=[0]*n; tat=[0]*n; t=0; done=[0]*n
from collections import deque
qz = deque()

while any(rt):
    for i in range(n):
        if at[i] <= t and rt[i] > 0 and i not in qz: qz.append(i)
    if not qz: t += 1; continue
    i = qz.popleft(); ex = min(q, rt[i])
    rt[i] -= ex; t += ex
    for j in range(n):
        if at[j] <= t and rt[j] > 0 and j not in qz: qz.append(j)
    if rt[i]==0: tat[i]=t-at[i]; wt[i]=tat[i]-bt[i]
```

```
print("\nPID\tAT\tBT\tWT\tTAT")
for i in range(n):
    print(i+1,"\t",at[i],"\t",bt[i],"\t",wt[i],"\t",tat[i])
print("\nAvg WT:", sum(wt)/n, " Avg TAT:", sum(tat)/n)
```

```
No. of processes: 3
Burst times: 6 2 4
Arrival times: 0 2 3
Time quantum: 2

PID     AT      BT      WT      TAT
1        0       6       2       8
2        2       2       2       4
3        3       4       5       9

Avg WT: 3.0  Avg TAT: 7.0
```

10.Write a C/Python program on First In First Out (FIFO) Page Replacement algorithm.

```
n = int(input("Enter number of frames: "))
pages = list(map(int, input("Enter page reference string: ").split()))

frames = []
faults = 0
hits = 0

print("\nPage\tFrames\t\tPage Fault")
for p in pages:
    if p not in frames:
        faults += 1
        if len(frames) < n:
            frames.append(p)
        else:
            frames.pop(0)
            frames.append(p)
        pf = "Yes"
    else:
        hits += 1
        pf = "No"
    print(p, "\t", frames, "\t", pf)
```

```
hit_ratio = hits / len(pages)
print("\nTotal Page Faults:", faults)
print("Total Page Hits:", hits)
print("Hit Ratio: {:.2f}".format(hit_ratio))
```

```
Enter number of frames: 3
Enter page reference string: 0 1 2 3 0 1 2 3 1 2 3

Page    Frames          Page Fault
0        [0]     Yes
1        [0, 1]          Yes
2        [0, 1, 2]       Yes
3        [1, 2, 3]       Yes
0        [2, 3, 0]       Yes
1        [3, 0, 1]       Yes
2        [0, 1, 2]       Yes
3        [1, 2, 3]       Yes
1        [1, 2, 3]       No
2        [1, 2, 3]       No
3        [1, 2, 3]       No

Total Page Faults: 8
Total Page Hits: 3
Hit Ratio: 0.27
```

11.Write a C/Python program on Least Recently Used (LRU) Page Replacement algorithm.

```
n = int(input("Enter number of frames: "))
pages = list(map(int, input("Enter page reference string: ").split()))

frames = []
recent = []
faults = 0
hits = 0

print("\nPage\tFrames\t\tPage Fault")
for p in pages:
    if p not in frames:
        faults += 1
        if len(frames) < n:
            frames.append(p)
```

```
        else:
            lru = recent.pop(0)
            frames.remove(lru)
            frames.append(p)
        pf = "Yes"
    else:
        hits += 1
        pf = "No"
        recent.remove(p)
    recent.append(p)
    print(p, "\t", frames, "\t", pf)

hit_ratio = hits / len(pages)
print("\nTotal Page Faults:", faults)
print("Total Page Hits:", hits)
print("Hit Ratio: {:.2f}".format(hit_ratio))
```

```
Enter number of frames: 3
Enter page reference string: 0 1 2 3 0 1 2 3 1 2 3

Page    Frames          Page Fault
0        [0]     Yes
1        [0, 1]          Yes
2        [0, 1, 2]       Yes
3        [1, 2, 3]       Yes
0        [2, 3, 0]       Yes
1        [3, 0, 1]       Yes
2        [0, 1, 2]       Yes
3        [1, 2, 3]       Yes
1        [1, 2, 3]       No
2        [1, 2, 3]       No
3        [1, 2, 3]       No

Total Page Faults: 8
Total Page Hits: 3
Hit Ratio: 0.27
```

12.Write a C/Python program on sequential file allocation method.

```
n = int(input("Enter total number of blocks: "))
f = int(input("Enter number of files: "))

mem = [0]*n
files = []

for i in range(f):
    start = int(input(f"\nEnter starting block of file {i+1}: "))
    length = int(input("Enter length of file: "))

    if start + length > n or any(mem[start:start+length]):
        print("Cannot allocate (out of range or already used)")
        continue

    alloc = list(range(start, start + length))
    for j in alloc:
        mem[j] = 1
    files.append((i+1, length, alloc))

print("\nFile\tLength\tBlocks Allocated")
for fno, length, alloc in files:
    print(f"{fno}\t{length}\t{alloc}")
```

```
Enter total number of blocks: 100
Enter number of files: 3

Enter starting block of file 1: 4
Enter length of file: 20

Enter starting block of file 2: 25
Enter length of file: 10

Enter starting block of file 3: 50
Enter length of file: 46

File    Length  Blocks Allocated
1       20      [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
2       10      [25, 26, 27, 28, 29, 30, 31, 32, 33, 34]
3       46      [50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95]
```