Q1)

In the current state, Need Matrix is and Available is :

Need

	A	B	C	D	
P0	7	5	3	4	✓
P1	2	1	2	2	✓
P2	3	4	4	2	✓
P3	2	3	3	1	✓
P4	4	1	2	1	✓
P5	3	4	3	3	✓

Available

A	B	C	D
6	3	5	4

P1 runs and Available becomes = [6, 4, 6, 5]
P3 runs and Available becomes = [7, 4, 6, 6]
P4 runs and Available becomes = [8, 5, 6, 6]
P5 runs and Available becomes = [9, 5, 7, 7]
P0 runs and Available becomes = [11, 5, 9, 8]
P2 runs and Available becomes = [15, 6, 9, 10]

- The sequence <P1, P3, P4, P5, P0, P2> can be executed so the current state is safe.

P5 requests [3, 2, 3, 3]

- The request does not exceed the need matrix and there are available resources. So we pretend to go into new state and do safety check.

- In the new state need and Available matrices are :

Need

	A	B	C	D
P0	7	5	3	4
P1	2	1	2	2
P2	3	4	4	2
P3	2	3	3	1
P4	4	1	2	1
P5	0	2	0	0

Available

A	B	C	D
3	1	2	1

No row in Need Matrix is <= Available vector.
So the old state should be restored and the requested resources should NOT be granted.

Q2)

a) Without TLB, for each memory operation, we need to access memory 2 times (1 to access page table and 1 to access the physical memory location corresponding to Logical Address). So:

$EAT = 150 \times 2 = 300$ ns

b) With TLB, if there is a miss, we need: $10+150+150 = 310$ ns

If there is a hit, we need: $10+150 = 160$ ns

$$EAT = \frac{85}{100} \cdot 160 + \frac{15}{100} \cdot 310 = 173,2 \text{ ns}$$

c) If we have 2-level paging used for question a):

$EAT = 150 \times 3 = 450$ ns (2 for mapping, 1 for data)

for question b): $EAT = \frac{85}{100} \cdot (10+150) + \frac{15}{100} \cdot (10+450) = 205$ ns

Q3

a) Physical addresses are 48 bits. 12 bits of it is offset. so the frame numbers consist of $48 - 12 = 36$ bits.

b) 1 4-th level page table maps $2^9 \times 2^{12} = 2^{21} = 2\,MB$ virtual space.

we need to map 6 MB of virtual memory, so we need 3 4-th level page table, 1 3-th level, 1 second-level, and 1 first-level tables.

In total we need $1 + 1 + 1 + 3 = 6$ tables

each occupies $2^9 \times 2^3 = 2^{12}$ bytes $= 4KB$

so $6 \times 4 = 24\,KB$ space is consumed.

c) For code segment we need 1 4-th level table since 126 KB is less than 2MB which is the area that can be covered by a single 4-th level table.

For data segment we need $2 \cdot 2^{10}\,MB / 2MB = 2^{10}$ 4-th level page table.

For stack segment we need $4\,MB / 2MB = 2$ 4-th level page table.

In total we need $1 + 1024 + 2 = 1027$ 4th level page tables.

1 3'rd level page table can map at most 2^9 4-th level page tables. So for 1027 4th level table we need 3 3th level page table. We also need 1 2nd and 1 first level table.

In total $1027 + 3 + 1 + 1 = 1032 \times 4\,KB = 4128\,KB$ memory is consumed

Q4

a)

Victim pointer is circled

2: (2(1)) F

4: (2(1)), 4(1) F

1: (2(1)), 4(1), 1(1) F

3: 3(1), (4(0)), 1(0) F

4: 3(1), (4(0)), 1(0)

6: 3(0), 6(1), (1(0)) F

3: 3(0), 6(1), (1(0))

6: 3(0), 6(1), (1(0))

1: 3(0), 6(1), (1(0))

2: (3(0)), 6(1), 2(1) F

1: 1(1), (6(0)) 2(0) F

5: 1(1), 5(1), (2(0)) F

2: 1(1), 5(1), (2(0))

5: 1(1), 5(1), (2(0))

4: (1(1)), 5(1), 4(1) F

1: (1(0)), 5(0), 4(0)

2: 2(1), (5(0)), 4(0) F

4: 2(1), (5(0)), 4(0)

3: 2(1), 3(1), (4(0)) F

Q4

b)

2:	2			F
4:	2	4		F
1:	2	4	1	F
3:	3	4	1	F
4:	3	4	1	
6:	3	6	1	F
3:	3	6	1	
6:	3	6	1	
1:	3	6	1	
2:	3	2	1	F
1:	3	2	1	
5:	5	2	1	F
2:	5	2	1	
5:	5	2	1	
4:	4	2	1	F
1:	4	2	1	
2:	4	2	1	
4:	4	2	1	
3:	4	2	3	F

Note: The last reference (3) is decided according to FIFO since there are no more references.

Q5

FCFS

5200 − 4500 = 700

5200 − 2000 = 3200

9500 − 2000 = 7500

9500 − 4300 = 5200

4300 − 1500 = 2800

5100 − 1500 = 3600

9600 − 5100 = 4500

9600 − 4000 = 5600

4600 − 4000 = + 600

In total 33700 cylinders

SCAN

In sorted order :

1500, 2000, 4000, 4300, [4500 ←], 4600, 5100, 5200

9500, 9600

(9999 − 4500) + (9999 − 1500)

→ 5499 + 8499 = 13998

SSTF

4500 → 4600 → 4300 → 4000 → 5100 → 5200 → 2000 → 1500

↓

9500

↓

9600

Distance = 100 + 300 + 300 + 1100 + 100 + 3200 + 500 + 8000 + 100

= 13700

Q6

Disk size = 512 GB

RPM = 3600

Avg seek time = 4 ms

Max-Transfer-Rate = 30 MB/s

Block size = 4KB

a) $T_{transfer} = 4KB / 30MB/s = \frac{4}{30 \times 1024} \times 1000 = 0,13 \text{ ms}$

$T_{rotation} = \frac{1}{3600} \times 60 \times 1000 = 8,33 \text{ ms}$

$T_{seek} = 4 \text{ ms}$

$T = \text{I/O time} = 4 + 8,33 + 0,13 = 12,46 \text{ ms}$

Number of I/Os per second $= \frac{1}{12,46} \cdot 1000 = 80$

$\text{Throughput} = \frac{4}{1024} \times 80 = 0,31 \text{ MB/s}$

b) $T_{Transfer} = \frac{512 \times 4 \text{ KB}}{30 \text{ MB/s}} = \frac{512 \times 4}{30 \times 1024} \times 1000 = 66,66 \text{ ms}$

I/O Time $= 66,66 + 4 + 8,33 = 78,99 \text{ ms}$

Number of I/Os per second $= \frac{1}{78,99} \times 1000 = 12$

$\text{Throughput} = \frac{512 \times 4}{1024} \times 12 = 24 \text{ MB/s}$

Q7.

a) In an inode we can store 4 KB / 8 = 512 pointers.
Total file size is: $10 + 512 + 512^2 + 512^3 \sim 2^{27}$ blocks
That makes $2^{27} \times 2^{12} = 2^{39}$ Bytes
Therefore max size supported is ~ 512 GB

b) A leaf index block can map $2^9 \times 2^{12} = 2^{21} = 2$ MB of file data.
For 8 GB, we need $2^{33} / 2^{21} = 2^{12} = 4096$ leaf index blocks. Hence we need to use the single level index structure (one leaf index block), and two level index structures (one first level index block and many second level - leaf - index blocks). We need to use 4096 - 1 = 4095 second level index blocks.

c) i) $2^{15} / 2^{12} = 2^3 = 8$. It is in the file block 8. Hence the respective block is pointed directly by the inode. So we need to do 1 disk access.

ii) offset = 2^{23} that is 8MB. A single index block can map 2 MB of file data. So we need to access the index block of the one-level index structure and then the data block. So we need 2 disk access.

iii) offset = 2^{32} = 4GB, 4GB / 2MB = 2048
So we need to use the two level index structure. We need 3 accesses. 1 for the first level index block, 1 for the second level index block, and 1 for the data block.