9.4.

a. Number of pages: $64 = 2^6$. → Need 6 bits to represent 64 pages.
   Page size: 1,024 words = 2^{10}. → Need 10 bits to represent an offset within a page
   ➔Total bits for logical address space = Number of bits for page number + Number of bits for word offset = 6 + 10 = 16 bits.

b. Number of frames: $32 = 2^5$ → Need 5 bits to represent 32 frames.
   Since the logical address space is mapped onto the physical memory → size of frame = size of page = 1,024 words → Need 10 bits to represent an offset within a frame
   ➔Total bits for physical address space = Number of bits for frame number + Number of bits for word offset = 5 + 10 = 15 bits.

9.6. Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order)

First-fit algorithm:

Process size (KB)	Placement Partition (KB)	Remaining Space (KB)
115	300	185
500	600	100
358	750	392
200	350	150
375	750	17

Best-fit algorithm:

Process size (KB)	Placement Partition (KB)	Remaining Space (KB)
115	125	10
500	600	100
358	750	392
200	200	0
375	750	17

Worst-fit algorithm:

Process size (KB)	Placement Partition (KB)	Remaining Space (KB)
115	750	635
500	750	135
358	600	242
200	350	150
375	Not possible due to external fragmentation	

9.7. 1KB = 1024 Bytes

a. 3085 ÷ 1024 = 3
   3085 % 1024 = 13
   ⇨ page number: 3 and offset: 13

b. 42095 ÷ 1024 = 41
   42095 % 1024 = 111
   ⇨ page number: 41 and offset: 111

c. 215201 ÷ 1024 = 210
   215201 % 1024 = 161
   ⇨ page number: 210 and offset: 161

d. 650000 ÷ 1024 = 634
   650000 % 1024 = 784
   ⇨ page number: 634 and offset: 784

e. 2000001 ÷ 1024 = 1953
   2000001 % 1024 = 129
   ⇨ page number: 1953 and offset: 129

9.25.

a. Each memory access takes 50 ns, and the page table is stored in memory.
➔ Total time for a paged memory reference = Page table access + Actual memory access = 100 ns

b.
On a TLB hit (75% of the time):

- TLB lookup: 2 ns

- Direct memory access: 50 ns

⇨ Total = 2 + 50 = 52 ns

On a TLB miss (25% of the time):

- TLB lookup: 2 ns
- Page table access (in memory): 50 ns
- Actual memory access: 50 ns

⇨ Total = 2 + 50 + 50 = 102 ns

➔ Effective memory reference time = 75% x 52 + 25% x 102 = 64.5 ns

10.5.

Page size is 256 bytes = 2^8 = 0x100 ➔ Represent the offset within the page: 8 lower bits.

Page table: 12 bits ➔ Represent the page number: 12 – 8 = 4 upper bits

- 9 | EF

  ➔ Page number: 9, Offset: EF

  From the table: Page 9 maps to page frame 0

  ➔ Physical address = 0 + EF = 0xEF

- 1 | 11

➔ Page number: 1, Offset: 0x11

From the table: Page 1 maps to page frame 2

➔ Physical address = 0x200 + 0x11 = 0x211

- 7 | 00

➔ Page number: 7, Offset: 0

From the table: Page 7 is not in memory → chooses frame D to load page 7 into → Page 7 maps to page frame D

➔ Physical address = 0xD00 + 0 = 0xD00

- 0 | FF

➔ Page number: 0, Offset: 0xFF

From the table: Page 0 is not in memory → chooses frame E to load page 0 into → Page 0 maps to page frame E

➔ Physical address = 0xE00 + 0xFF = 0xEFF

10.7.

A paged memory system with pages of size 200 bytes, each row contains 100 bytes → Each page holds 2 rows in contiguous order.

Only have 1 page frame → Every time accessing a new page → get a page fault

a. This loop goes column-by-column, then row-by-row inside each column. That means for 1 column, we're accessing all 100 different rows which are in 50 different pages → For each column, get 50 page faults → Total page faults: 100 × 50 = 5000

b. This loop goes row-by-row. That means for 1 row, we're accessing all 100 columns which are in the same page → no page fault when traversing columns.
   Each page fits 2 rows → Every 2 rows, we get 1 page fault → Total page faults: 100 / 2 = 50

10.8. Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

The below table shows number of page faults would occur for the following replacement algorithms with assuming number of frames:

Number of frames	1	2	3	4	5	6	7
LRU replacement	20	18	15	10	8	7	7
FIFO replacement	20	18	16	14	10	10	7
Optimal replacement	20	15	11	8	7	7	7

10.22.

Page size: 4,096 bytes = 2^{12} bytes = 0x1000 bytes → Represent the offset within the page: 12 lower bits

Page table: 16 bits

→ Represent the page number: 16 – 12 = 4 upper bits

a.

- 6 | 21C

➔ Page number: 6, Offset: 21C

From the table: Page 6 maps to page frame 8 ➔ Set the reference bit for page 6 to 1

➔ Physical address = 0x8000 + 21C = 0x821C

- F | 0A3

➔ Page number: F = 15, Offset: 0A3
From the table: Page 15 maps to page frame 2 ➔ Set the reference bit for page 15 to 1
➔ Physical address = 0x2000 + 0A3 = 0x20A3

- B | C1A

➔ Page number: B = 11, Offset: C1A
From the table: Page 11 maps to page frame 4 ➔ Set the reference bit for page 11 to 1
➔ Physical address = 0x4000 + C1A = 0x4C1A

- 5 | BAA

➔ Page number: 5, Offset: BAA
From the table: Page 5 maps to page frame 13 = D ➔ Set the reference bit for page 5 to 1
➔ Physical address = 0xD000 + BAA = 0xDBAA

- 0 | BA1

➔ Page number: 0, Offset: BA1
From the table: Page 0 maps to page frame 9 ➔ Set the reference bit for page 0 to 1
➔ Physical address = 0x9000 + BA1 = 0x9BA1

b. From the page table, pages 1, 9, and 14 are not in memory → will cause a page fault when accessed. A logical address that results in a page fault: 0xE056 (tự chế)

c. LRU page-replacement algorithm choose the page frames that are currently in memory → Set of page frames the LRU page-replacement algorithm will choose in resolving a page fault = {0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15}.

1. 64 byte cuối của Master Boot Sector xác định số phân vùng và loại của từng phân vùng. Mỗi 16-byte block là 1 phân vùng có các byte bắt đầu lần lượt là 1BE, 1CE, 1DE, 1EE

	Byte offset +4	Loại phân vùng
Phân vùng 1	0xDE	Phân vùng OEM đặc biệt
Phân vùng 2	0x07	Windows
Phân vùng 3	0x83	Linux
Phân vùng 4	0x0F	Phân vùng mở rộng

Vì 4 phân vùng đều có byte loại phân vùng khác 0 nên ổ cứng được chia thành 4 phân vùng. Chỉ có byte đầu tiên của phân vùng 2 (1CE) = 0x80 nên phân vùng 2 là phân vùng khởi động

2.

a. Thời gian để đọc 1 tập tin vào bộ nhớ chính = Seek time + Rotational time + Transfer time = 5ms + 4ms + $\frac{8\times10^3}{8\times2^{10}}$ ms = 9.98ms

Thời gian ghi tập tin vào 1 vị trí khác trên đĩa = Seek time + Rotational time + Transfer time = 5ms + 4ms + $\frac{8\times10^3}{8\times2^{10}}$ ms = 9.98ms

➔ Tổng thời gian để đọc 1 tập tin vào bộ nhớ chính, sau đó ghi tập tin vào 1 vị trí khác trên đĩa = 9.98 x 2 = 19.96ms

b. Thực hiện dồn ½ đĩa 16GB = 8GB → Có $\frac{8\times2^{20}}{8} = 2^{20}$ tập tin → Tổng thời gian dồn ½ đĩa 8GB = $2^{20} \times 9.98\, ms \approx 10465s$

3.

a. Số lượng địa chỉ trong một khối = Kích thước khối đĩa / Kích thước mỗi chỉ mục = 1024 byte / 4 byte = 256 địa chỉ = 2^8 địa chỉ.

Các chỉ số trực tiếp: Có 12 chỉ số trực tiếp, mỗi chỉ số trỏ đến 1 khối dữ liệu. → Số khối dữ liệu = 12 → Dung lượng = 12 x 2^{10} byte = 12KB

Chỉ mục 1 cấp: Chỉ mục này trỏ đến một khối chứa các địa chỉ của các khối dữ liệu khác → Số khối dữ liệu = 256 → Dung lượng = 256 x 2^{10} byte = 256KB

Chỉ mục 2 cấp: Chỉ mục này trỏ đến một khối chứa các địa chỉ của các khối chỉ mục 1 cấp. Mỗi khối chỉ mục gián tiếp đơn này lại trỏ đến các khối dữ liệu.→ Số khối dữ liệu = 2^8 x 2^8 = 2^{16} → Dung lượng = 2^{16} x 2^{10} byte = 64MB

Chỉ mục 3 cấp: Chỉ mục này trỏ đến một khối chứa các địa chỉ của các khối chỉ mục 2 cấp. Mỗi khối chỉ mục 2 cấp lại trỏ đến các khối chỉ mục 1 cấp, và cuối cùng trỏ đến các khối dữ liệu.
→ Số khối dữ liệu = 2^8 x 2^8 x 2^8 = 2^{24} → Dung lượng = 2^{24} x 2^{10} byte = 16GB

➔ Kích thước tập tin lớn nhất có thể quản lý được là: 16GB + 64MB + 256KB + 12KB = 16GB + 64MB + 268KB

b. Tổng số chỉ mục trong cấu trúc Inode = 12 chỉ mục trực tiếp + 1 chỉ mục 1 cấp + 1 chỉ mục 2 cấp + 1 chỉ mục 3 cấp = 15 chỉ mục
→ Inode có thể lưu trực tiếp tập tin có kích thước tối đa = 15 x 4 = 60 byte

4.

a.

Nội dung	Giá trị
Số byte cho 1 sector	0x0200 = 512
Số sector cho 1 cluster	0x04 = 4
Số sector vùng Boot sector (Số sector dành riêng)	0x0001 = 1
Số sector cho 1 bảng FAT	0x0028 = 40
Số bảng FAT	0x02 = 2
Số sector cho bảng RDET	0x0200 x 32 / 512 = 32
Kích thước volume (megabytes)	0x00009FE0 = 40928 / 2048 = 19.98
Sector đầu tiên của bảng FAT1	0x0001 = 1

Sector đầu tiên của bảng RDET	1 + 40 x 2 = 81
Sector đầu tiên của vùng data	81 + 32 = 113

b.

	Thuộc tính trạng thái (Offset B – 1 byte)	Kích thước (Offset 1C – 4 bytes)	Chỉ số các sector chứa dữ liệu (Offset 14 – 2 bytes cao Offset 1A – 2 bytes thấp)
EXTERNAL.DLL	Tập tin	1E00 = 7680B	Chỉ số các cluster được cấp: 15, 16, 17, 18 Sector đầu tiên của cluster 15: (15 - 2) * 4 + 113 = 165 **→ Chỉ số các sector được cấp: 165 đến 180** Số sector cần thiết: 7680 / 512 = 15 sector **→ Chỉ số các sector chứa dữ liệu: 165 đến 179**
File Type Signature.txt	Tập tin	0xDAD = 3501B	Chỉ số các cluster được cấp: 5, 6 **→ Chỉ số các sector được cấp: 125 đến 132** Số sector cần thiết: 7 sector **→ Chỉ số các sector chứa dữ liệu: 125 đến 131**
ABC	Thư mục	0B	Chỉ số các cluster được cấp: 7 **→ Chỉ số các sector được cấp: 133 đến 136** **→ Chỉ số các sector chứa dữ liệu: 133**
Ext Superblock.tpl	Tập tin	0x0911 = 2321B	Chỉ số các cluster được cấp: 8, 9 **→ Chỉ số các sector được cấp: 137 đến 144** Số sector cần thiết: 5 sector

			→ Chỉ số các sector chứa dữ liệu: 137 đến 141
TEST	Thư mục	0B	Chỉ số các cluster được cấp: 20 **→ Chỉ số các sector được cấp: 185 đến 188** **→ Chỉ số các sector chứa dữ liệu: 185**
USER.TXT	Tập tin	0xA2 = 162B	Chỉ số các cluster được cấp: 10 **→ Chỉ số các sector được cấp: 145 đến 148** Số sector cần thiết: 1 sector **→ Chỉ số các sector chứa dữ liệu: 145**
SESSION.PRJ	Tập tin	0x025F = 607B	Chỉ số các cluster được cấp: 11 **→ Chỉ số các sector được cấp: 149 đến 152** Số sector cần thiết: 2 sector **→ Chỉ số các sector chứa dữ liệu: 149, 150**

Giả sử các thao tác ở các câu c, d, e là độc lập với nhau

c. Khi thực hiện lệnh XÓA đối với một tập tin có phần mở rộng là ".prj", các thao tác sẽ ảnh hưởng đến một số vùng bảng trên Volume như sau:

- Sector 185: Byte tại offset 0x017260 sẽ được ghi giá trị 0xE5. Đây là dấu hiệu cho biết entry tương ứng với tập tin đã bị xóa khỏi thư mục (đánh dấu là "deleted").
- RDET: Không có bất kỳ thay đổi nào, vì tập tin không nằm trong thư mục gốc (root directory).
- FAT 1: Hai byte tại các offset 0x0216 và 0x0217 sẽ được đặt về 0x00. Điều này có nghĩa là cluster 11 – nơi lưu trữ nội dung tập tin – đã được giải phóng và hiện sẵn sàng để cấp phát cho tập tin mới.
- DATA: Nội dung thực tế của tập tin trong vùng dữ liệu không bị thay đổi. Tuy nhiên, do các cluster chứa dữ liệu tập tin này không còn được hệ thống

gán cho tập tin nào nữa, chúng có thể bị ghi đè nếu được sử dụng cho các tập tin mới trong tương lai.

c. Khi thực hiện lệnh DI CHUYỂN tập tin có phần mở rộng là "TPL" tới thư mục gốc thì thao tác này sẽ ảnh hưởng đến một số vùng bảng trên Volume như sau:

- RDET: Hệ thống tìm được 3 vị trí trống liên tiếp trong bảng RDET, bắt đầu từ địa chỉ 0xA200 đến 0xA240.
  Tiếp theo, nội dung của ba entry từ sector 133 được sao chép lần lượt như sau:
  - Entry tại offset 0x10A40 được sao chép vào 0xA200.
  - Entry tại offset 0x10A60 được sao chép vào 0xA220.
  - Entry tại offset 0x10A80 được sao chép vào 0xA240.
- Sector 133: Sau khi sao chép, ba entry gốc tại các offset 0x10A40, 0x10A60 và 0x10A80 sẽ được ghi giá trị 0xE5. Đây là dấu hiệu cho biết các entry này đã bị xóa khỏi thư mục con.
- FAT1, DATA: Không có bất kỳ thay đổi nào đối với bảng FAT1 hay nội dung dữ liệu thực tế của các tập tin. Các cluster dữ liệu vẫn giữ nguyên và chưa bị ghi đè hay cấp phát lại.

d. Khi thực hiện lệnh SAO CHÉP tập tin EXTERNAL.DLL tới thư mục ABC thì thao tác này sẽ ảnh hưởng đến một số vùng bảng trên Volume như sau:

- Sector 133: Hệ thống tìm thấy một entry trống tại địa chỉ 0x10AA0. Nội dung của entry tại 0xA260 trong RDET – bao gồm toàn bộ thông tin về tập tin nguồn – được sao chép vào 0x10AA0.
  Sau đó, hai byte tại địa chỉ 0xA27A và 0xA27B (trong bản sao entry) được đặt lần lượt là 0x02 và 0x00, biểu thị rằng cluster đầu tiên của tập tin là cluster số 2.
- FAT: Hệ thống tìm thấy các cluster trống: 2, 3, 4 và 12. Các cluster này được liên kết với nhau theo danh sách liên kết như sau:
  - 0x0204, 0x0205 được ghi giá trị 0x03, 0x00 → cluster 2 trỏ đến cluster 3
  - 0x0206, 0x0207 được ghi 0x04, 0x00 → cluster 3 trỏ đến cluster 4
  - 0x0208, 0x0209 được ghi 0x0C, 0x00 → cluster 4 trỏ đến cluster 12

    - 0x0218, 0x0219 được ghi 0xFF, 0xFF → cluster 12 là cluster cuối cùng trong chuỗi
- DATA: Nội dung dữ liệu từ các sector:
    - 165 đến 176 được sao chép lần lượt sang các sector 113 đến 124
    - 177 đến 180 được sao chép sang các sector 153 đến 156
      Việc này nhằm bảo toàn nội dung của tập tin trong vùng dữ liệu mới được gán.
- RDET: Không có bất kỳ thay đổi nào đối với thông tin của tập tin gốc trong RDET. Entry của tập tin nguồn vẫn được giữ nguyên.