Name: Hanzhong Liu

UIN: 734004071

Question 1

(a)

For row major order:

- In row major order. Each row of the array has 512 elements, and the page size is also 512. Therefore, each row fits exactly in one page. With 16 frames of physical memory available, we can hold 16 rows/pages in memory at a time.
- As we access the array sequentially in row major order, we will have a page fault when we access the first element of each new row until all 16 frames are filled. After that, for each new row accessed, one page fault will occur as the least recently added page will be replaced (FIFO).
- There are a total of 512 rows in the array, so the number of page faults will be 512 (one for each row).

For column major order:

- In column major order, the array is accessed column by column. Each column of the array has 512 elements, since the array is stored in row major order, accessing a column means accessing one element from each row, which results in accessing 512 different pages.
- With only 16 frames of physical memory, we can only hold 16 pages in memory at a time. As we access the array sequentially in column major order, we will have a page fault for each element of the column, since each element is on a different page and we don't have enough frames to hold all pages needed for a single column. There are 512 elements in each column, and there are 512 columns. Therefore, the number of page faults will be 512 * 512 = 262,144.

In summary:

- For row major order: 512 page faults
- For column major order: 262,144 page faults

(b)

There will has no difference in the LRU policy.

For row major order, we still need to access these rows one by one, so get 512 page fault. For column major order, since each row need to access all 512 pages and we only have 16 frames, all 512 pages can't be loaded into memory, so LRU will no reduce the number of page fault.

Question 2

a_* b g a_* d e a_* b a_* d e g d e

(a)

The optimal policy replaces the page that will not be used for the longest period of time in the future.

- Initial state: [_, _, _, _]

- a* (fault, read, write): [a, _, _, _]
- b (fault, read): [a, b, _, _]
- g (fault, read): [a, b, g, _]
- a* (hit): [a, b, g, _]
- d (fault, read): [a, b, g, d]

```
a* b g a* d e a* b a* d e g d e
            ↑             ↑
```

- e (fault, read): [a, b, e, d] (replace g)
- a* (hit): [a, b, e, d]
- b (hit): [a, b, e, d]
- a* (hit): [a, b, e, d]
- d (hit): [a, b, e, d]
- e (hit): [a, b, e, d]
- g (fault): [g, b, e, d] (replace a)
- d (hit): [a, e, g, d]
- e (hit): [a, e, g, d]

Access Page	Replaced Page	Frames
a* (fault)		a
b (fault)		a b
g (fault)		a b g
a*		a b g
d (fault)		a b g d
e (fault)	d	a b g e
a*		a b g e
b		a b g e
a*		a b g e
d (fault)	a	d b g e
e		d b g e
g		d b g e
d		d b g e
e		d b g e

Page faults number: 6

(b) FIFO

Access Page	Replaced Page	Frames
a* (fault)		a
b (fault)		a b
g (fault)		a b g
a*		a b g
d (fault)		a b g d
e (fault)	a	b g d e
a* (fault)	b	g d e a
b (fault)	g	d e a b
a*		d e a b
d		d e a b
e		d e a b
g (fault)	d	e a b g
d (fault)	e	a b g d
e (fault)	a	b g d e

Page faults: 10

(c) Second-chance algorithm

Access Page	Replaced Page	Frames
a* (fault)		a(R:1)
b (fault)		a(R:1) b(R:1)
g (fault)		a(R:1) b(R:1) g(R:1)
a*		a(R:1) b(R:1) g(R:1)
d (fault)		a(R:1) b(R:1) g(R:1) d(R:1)
e (fault)	a	e(R:1) b(R:0) g(R:0) d(R:0)
a* (fault)	b	e(R:1) a(R:1) g(R:0) d(R:0)
b (fault)	g	e(R:1) a(R:1) b(R:1) d(R:0)
a*		e(R:1) a(R:1) b(R:1) d(R:0)
d		e(R:1) a(R:1) b(R:1) d(R:1)
e		e(R:1) a(R:1) b(R:1) d(R:1)
g (fault)	d	e(R:0) a(R:0) b(R:0) g(R:1)
d (fault)	e	d(R:1) a(R:0) b(R:0) g(R:1)
e (fault)	a	d(R:1) e(R:1) b(R:0) g(R:1)

Second-Chance Algorithm page faults: 10

(d) Enhanced second-chance algorithm

Access Page	Replaced Page	Frames
a* (fault)		a(u:1,d:1)
b (fault)		a(u:1,d:1) b(u:1,d:0)
g (fault)		a(u:1,d:1) b(u:1,d:0) g(u:1,d:0)
a*		a(u:1,d:1) b(u:1,d:0) g(u:1,d:0)
d (fault)		a(u:1,d:1) b(u:1,d:0) g(u:1,d:0) d(u:1,d:0)
e (fault)	b	a(u:0,d:0) e(u:1,d:0) g(u:0,d:0) d(u:0,d:0)
a*		a(u:1,d:1) e(u:1,d:0) g(u:0,d:0) d(u:0,d:0)
b (fault)	g	a(u:1,d:1) e(u:1,d:0) b(u:1,d:0) d(u:0,d:0)
a*		a(u:1,d:1) e(u:1,d:0) b(u:1,d:0) d(u:0,d:0)
d		a(u:1,d:1) e(u:1,d:0) b(u:1,d:0) d(u:1,d:0)
e		a(u:1,d:1) e(u:1,d:0) b(u:1,d:0) d(u:1,d:0)
g (fault)	d	a(u:0,d:1) e(u:0,d:0) b(u:0,d:0) g(u:1,d:0)
d (fault)	e	a(u:0,d:0) d(u:1,d:0) b(u:0,d:0) g(u:1,d:0)
e (fault)	b	a(u:0,d:0) d(u:1,d:0) e(u:1,d:0) g(u:1,d:0)

Enhanced Second-Chance Algorithm page faults: 9

Question 3

(a) LRU

Access Page	Replaced Page	Frames
A (fault)		A
C (fault)		A C
B* (fault)		A C B
D (fault)		A C B D
B*		A C D B
A		C D B A
E (fault)	C	D B A E
F (fault)	D	B A E F
B*		A E F B
F		A E B F
A		E B F A
G (fault)	E	B F A G
E (fault)	B	F A G E
F		A G E F
A		G E F A

LRU page faults: 8

(b) Optimal

A C B* D B* A E F B* F A G E F A

- Initial state: [_, _, _, _]
- A* (fault, read, write): [A, _, _, _]
- C (fault, read): [A, C, _, _]
- B* (fault, read): [A, C, B, _]
- D (fault, read): [A, C, B, D]
- B* (hit): [A, C, B, D]
- A (hit): [A, C, B, D]

  A C B* D B* A E F B* F A G E F A

  ↑
- E (fault, replace C): [A, E, B, D]

A C B* D B* A E F B* F A G E F A
↑

- F (fault, replace D): [A, E, B, F]
- B* (hit): [A, E, B, F]
- F (hit): [A, E, B, F]
- A (hit): [A, E, B, F]

  A C B* D B* A E F B* F A G E F A
  ↑

- G (fault, replace B): [A, E, G, F]
- E (hit): [A, E, G, F]
- F (hit): [A, E, G, F]
- A (hit): [A, E, G, F]

Access Page	**Replaced Page**	**Frames**
A (fault)		A
C (fault)		A C
B* (fault)		A C B
D (fault)		A C B D
B*		A C B D
A		A C B D
E (fault)	C	A E B D
F (fault)	D	A E B F
B*		A E B F
F		A E B F
A		A E B F
G (fault)	B	A E G F
E		A E G F
F		A E G F
A		A E G F

Page faults number: 7

(c) Second-Chance Algorithm

Access Page	Replaced Page	Frames
A (fault)		A(R:1)
C (fault)		A(R:1) C(R:1)
B* (fault)		A(R:1) C(R:1) B(R:1)
D (fault)		A(R:1) C(R:1) B(R:1) D(R:1)
B*		A(R:1) C(R:1) B(R:1) D(R:1)
A		A(R:1) C(R:1) B(R:1) D(R:1)
E (fault)	A	E(R:1) C(R:0) B(R:0) D(R:0)
F (fault)	C	E(R:1) F(R:1) B(R:0) D(R:0)
B*		E(R:1) F(R:1) B(R:1) D(R:0)
F		E(R:1) F(R:1) B(R:1) D(R:0)
A (fault)	D	E(R:1) F(R:1) B(R:0) A(R:1)
G (fault)	B	E(R:0) F(R:0) G(R:1) A(R:1)
E		E(R:1) F(R:0) G(R:1) A(R:1)
F		E(R:1) F(R:1) G(R:1) A(R:1)
A		E(R:1) F(R:1) G(R:1) A(R:1)

Second-Chance Algorithm page faults: 8

Question 4

(a)

Since there are n distinct page numbers and m page frames, the minimum number of page faults that must occur is when each of the m page frames is filled with a distinct page number without any faults. However, since n > m, at least n - m additional page faults must occur to load the remaining distinct pages into the memory. Therefore, the lower bound on the number of page faults is:

$$LowerBound = m + (n - m) = n \tag{1}$$

(b)

The upper bound on the number of page faults occurs in the worst-case scenario, where every page reference results in a fault. This could happen if the page replacement scheme always replaces a page that will be needed immediately after. In this case, every page reference in the string of length p would result in a page fault. Therefore, the upper bound on the number of page faults is:

$$UpperBound = p \tag{2}$$

Question 5

(a)

Access Page	**Replaced Page**	**Frames**
0 (fault)		0
1 (fault)		0 1
2 (fault)		0 1 2
3 (fault)		0 1 2 3
0		0 1 2 3
1		0 1 2 3
4 (fault)	0	1 2 3 4
0 (fault)	1	2 3 4 0
1 (fault)	2	3 4 0 1
2 (fault)	3	4 0 1 2
3 (fault)	4	0 1 2 3
4 (fault)	0	1 2 3 4

FIFO page faults (frame number is 4): 10

Access Page	Replaced Page	Frames
0 (fault)		0
1 (fault)		0 1
2 (fault)		0 1 2
3 (fault)	0	1 2 3
0 (fault)	1	2 3 0
1 (fault)	2	3 0 1
4 (fault)	3	0 1 4
0		0 1 4
1		0 1 4
2 (fault)	0	1 4 2
3 (fault)	1	4 2 3
4		4 2 3

FIFO page faults (frame number is 3): 9

So, we got Belady's anomaly.

(b)

LRU page faults (frame number is 4): 8
LRU page faults (frame number is 3): 10

Enhanced-Second-Chance' page faults (frame number is 4): 7
Enhanced-Second-Chance' page faults (frame number is 3): 9

Optimal page faults (frame number is 4): 6
Optimal page faults (frame number is 3): 7

So, for these agorithm (LRU, the second-chance algorithm, and the optimal algorithm), as the page frames increase, the page fault decreases. There is no Belady's anomaly for them.

Question 5

I think it's a good design:

1. **Reduced I/O Overhead**: By not swapping out pages of executable code, the operating system reduces the amount of I/O operations needed to write to and read from the swap device. This can lead to improved performance, especially if the swap device is slow.

2. **Efficient Use of Swap Space**: Swap space is a limited resource, and by not using it for pages that can be fetched directly from the executable file, the operating system can use the swap space more efficiently for other data that cannot be easily retrieved from disk.

However, there are also a problem:

Disk Access Time: While this design reduces I/O overhead for swapping, it may increase the time to access the executable code if it is not already in memory, as reading from disk is generally slower than reading from swap space.