Name: - Harsh Zanwar

Class: - AIDS-C / B1

Roll Number: - 77
PRN: - 12110333

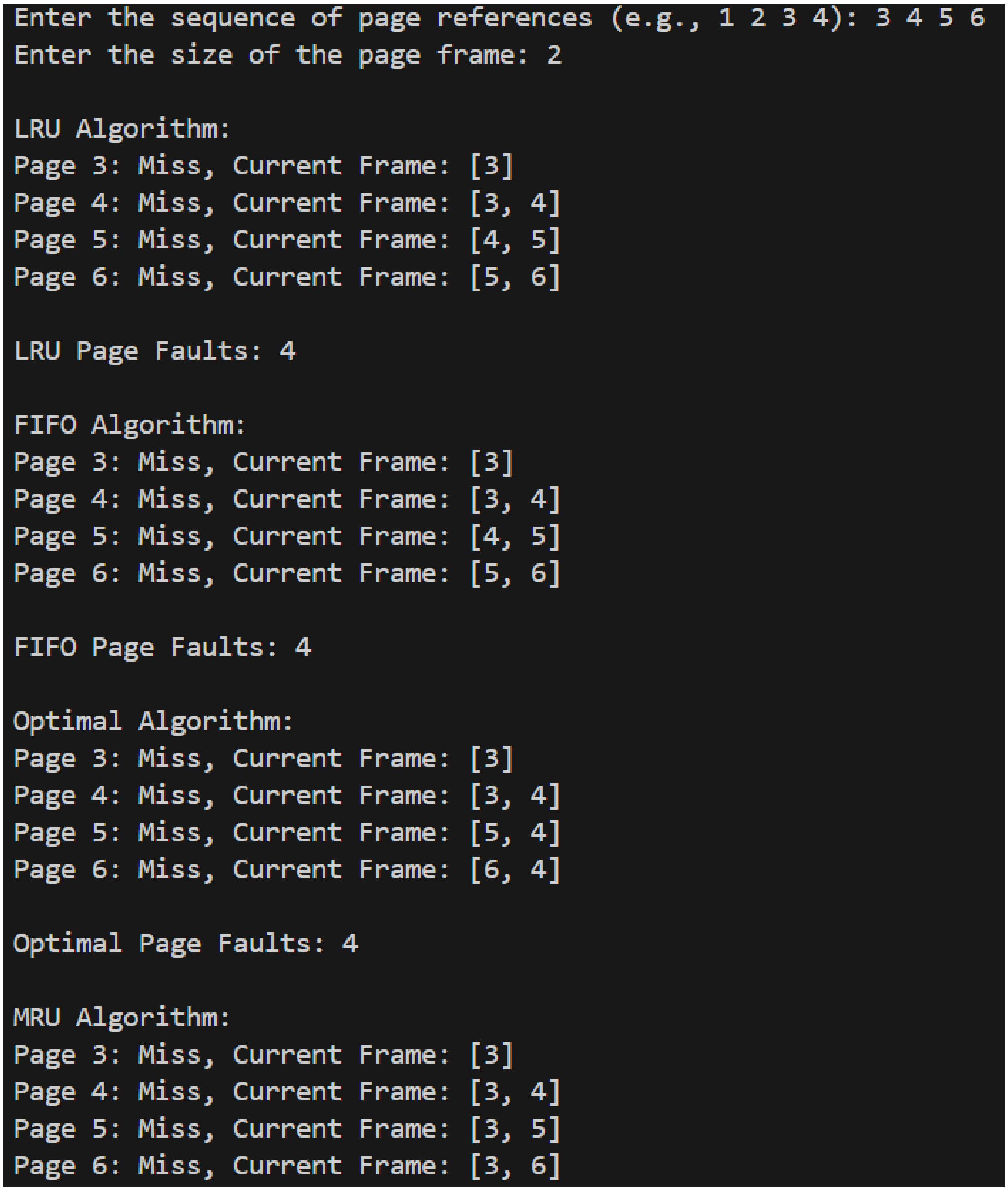
Code :-

```python
class PageReplacementAlgorithm:
    def_init_(self, sequence, frame_size):
        self.sequence = sequence
        self.frame_size = frame_size
        self.page_faults = 0
        self.page_table = {}

    def run_algorithm(self):
        pass


class LRU(PageReplacementAlgorithm):
    def run_algorithm(self):
        frame = []
        page_faults = 0

        for index, page in enumerate(self.sequence):
            hit_or_miss = "Hit" if page in frame else "Miss"
            if page not in frame:
                page_faults += 1
                if len(frame) < self.frame_size:
                    frame.append(page)
                else:
                    least_recently_used = min(frame, key=frame.index)
                    frame.remove(least_recently_used)
                    frame.append(page)
            else:
                frame.remove(page)
                frame.append(page)

            print(f"Page {page}: {hit_or_miss}, Current Frame: {frame}")

        self.page_faults = page_faults
```

```python
class FIFO(PageReplacementAlgorithm):
    def run_algorithm(self):
        frame = []
        page_faults = 0

        for index, page in enumerate(self.sequence):
            hit_or_miss = "Hit" if page in frame else "Miss"
            if page not in frame:
                page_faults += 1
                if len(frame) < self.frame_size:
                    frame.append(page)
                else:
                    frame.pop(0)
                    frame.append(page)

            print(f"Page {page}: {hit_or_miss}, Current Frame: {frame}")

        self.page_faults = page_faults


class Optimal(PageReplacementAlgorithm):
    def run_algorithm(self):
        frame = []
        page_faults = 0

        for index, page in enumerate(self.sequence):
            hit_or_miss = "Hit" if page in frame else "Miss"
            if page not in frame:
                page_faults += 1
                if len(frame) < self.frame_size:
                    frame.append(page)
                else:
                    future_occurrences = {p: self.next_occurrence(index, p) for p
in frame}
                    page_to_remove = max(future_occurrences,
key=future_occurrences.get)
                    frame[frame.index(page_to_remove)] = page

            print(f"Page {page}: {hit_or_miss}, Current Frame: {frame}")

        self.page_faults = page_faults

    def next_occurrence(self, index, page):
        try:
            return self.sequence[index:].index(page)
        except ValueError:
            return float('inf')


class MRU(PageReplacementAlgorithm):
    def run_algorithm(self):
        frame = []
        page_faults = 0

        for index, page in enumerate(self.sequence):
            hit_or_miss = "Hit" if page in frame else "Miss"
            if page not in frame:
                page_faults += 1
                if len(frame) < self.frame_size:
                    frame.append(page)
                else:
                    most_recently_used = max(frame, key=frame.index)
                    frame.remove(most_recently_used)
                    frame.append(page)
            else:
                frame.remove(page)
                frame.append(page)

            print(f"Page {page}: {hit_or_miss}, Current Frame: {frame}")

        self.page_faults = page_faults


# Take user input
sequence = input("Enter the sequence of page references (e.g., 1 2 3 4):
").split()
sequence = [int(page) for page in sequence]
frame_size = int(input("Enter the size of the page frame: "))

# Run algorithms
lru = LRU(sequence, frame_size)
print("\nLRU Algorithm:")
lru.run_algorithm()
print("\nLRU Page Faults:", lru.page_faults)

fifo = FIFO(sequence, frame_size)
print("\nFIFO Algorithm:")
fifo.run_algorithm()
print("\nFIFO Page Faults:", fifo.page_faults)
```

```python
optimal = Optimal(sequence, frame_size)
print("\nOptimal Algorithm:")
optimal.run_algorithm()
print("\nOptimal Page Faults:", optimal.page_faults)

mru = MRU(sequence, frame_size)
print("\nMRU Algorithm:")
mru.run_algorithm()
print("\nMRU Page Faults:", mru.page_faults)
```

Output :-



```
Enter the sequence of page references (e.g., 1 2 3 4): 3 4 5 6
Enter the size of the page frame: 2

LRU Algorithm:
Page 3: Miss, Current Frame: [3]
Page 4: Miss, Current Frame: [3, 4]
Page 5: Miss, Current Frame: [4, 5]
Page 6: Miss, Current Frame: [5, 6]

LRU Page Faults: 4

FIFO Algorithm:
Page 3: Miss, Current Frame: [3]
Page 4: Miss, Current Frame: [3, 4]
Page 5: Miss, Current Frame: [4, 5]
Page 6: Miss, Current Frame: [5, 6]

FIFO Page Faults: 4

Optimal Algorithm:
Page 3: Miss, Current Frame: [3]
Page 4: Miss, Current Frame: [3, 4]
Page 5: Miss, Current Frame: [5, 4]
Page 6: Miss, Current Frame: [6, 4]

Optimal Page Faults: 4

MRU Algorithm:
Page 3: Miss, Current Frame: [3]
Page 4: Miss, Current Frame: [3, 4]
Page 5: Miss, Current Frame: [3, 5]
Page 6: Miss, Current Frame: [3, 6]
```