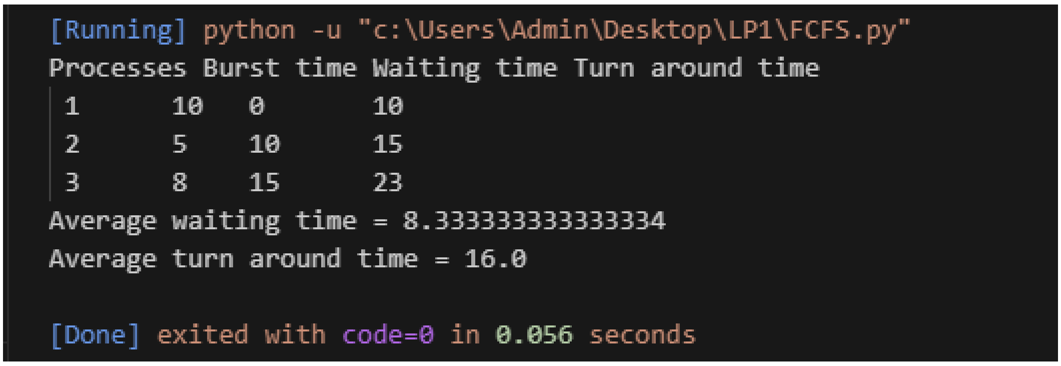
FIFO:-

```
def findWaitingTime(processes, n, bt, wt):
   wt[0] = 0
   for i in range(1, n):
       wt[i] = bt[i - 1] + wt[i - 1]
def findTurnAroundTime(processes, n, bt, wt, tat):
   for i in range(n):
       tat[i] = bt[i] + wt[i]
def findavgTime(processes, n, bt):
   wt = [0] * n
   tat = [0] * n
   total_wt = 0
   total_tat = 0
   findWaitingTime(processes, n, bt, wt)
   findTurnAroundTime(processes, n, bt, wt, tat)
   print("Processes Burst time Waiting time Turn around time")
   for i in range(n):
       total_wt += wt[i]
       total_tat += tat[i]
       print(f" {i + 1}\t\t{bt[i]}\t {wt[i]}\t\t {tat[i]}")
   print("Average waiting time = " + str(total_wt / n))
   print("Average turn around time = " + str(total_tat / n))
if __name__ == "__main__":
   # Process IDs
   processes = [1, 2, 3]
   n = len(processes)
   # Burst time of all processes
   burst_time = [10, 5, 8]

   findavgTime(processes, n, burst_time)
```

OUTPUT :-

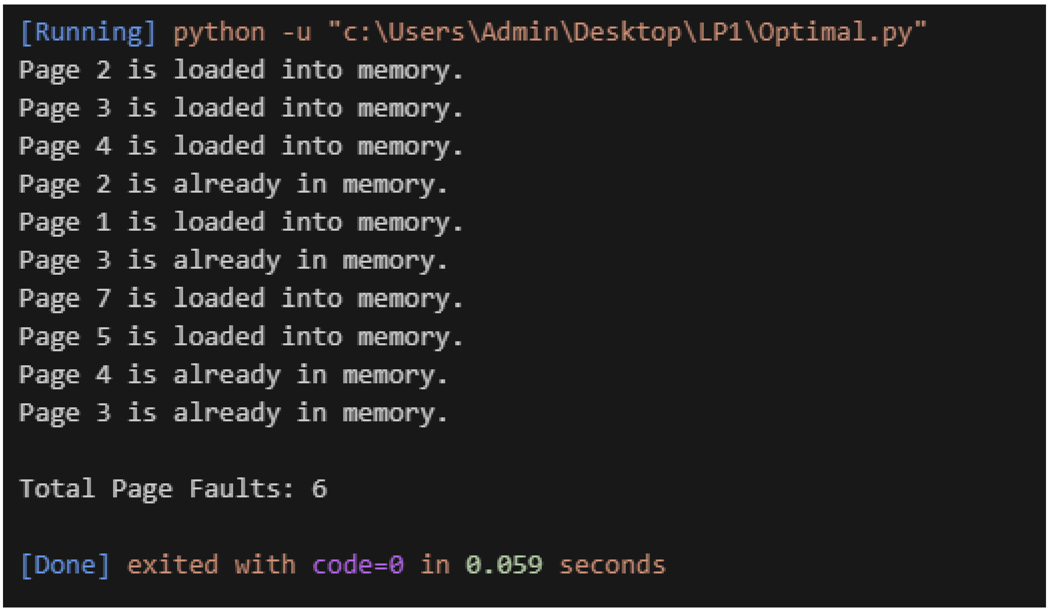


```
[Running] python -u "c:\Users\Admin\Desktop\LP1\FCFS.py"
Processes Burst time Waiting time Turn around time
 1       10    0        10
 2       5     10       15
 3       8     15       23
Average waiting time = 8.333333333333334
Average turn around time = 16.0

[Done] exited with code=0 in 0.056 seconds
```

OPTIMAL :-

```
def optimal_page_replacement(pages, capacity):
    page_faults = 0
    page_frames = [-1] * capacity
    for i in range(len(pages)):
        if pages[i] not in page_frames:
            if -1 in page_frames:
                # If there is an empty frame, place the page in it
                index = page_frames.index(-1)
                page_frames[index] = pages[i]
            else:
                # Find the page that will not be used for the longest period in the future
                future_occurrences = {page: float('inf') for page in page_frames}
                for j in range(i + 1, len(pages)):
                    if pages[j] in future_occurrences:
                        future_occurrences[pages[j]] = j
                # Replace the page that is not needed for the longest time
                page_to_replace = max(future_occurrences, key=future_occurrences.get)
                index = page_frames.index(page_to_replace)
                page_frames[index] = pages[i]
            print(f"Page {pages[i]} is loaded into memory.")
            page_faults += 1
        else:
            print(f"Page {pages[i]} is already in memory.")
    print(f"\nTotal Page Faults: {page_faults}")
if __name__ == "__main__":
    # Example usage
    page_references = [2, 3, 4, 2, 1, 3, 7, 5, 4, 3]
    memory_capacity = 3

    optimal_page_replacement(page_references, memory_capacity)
```

OUTPUT:-



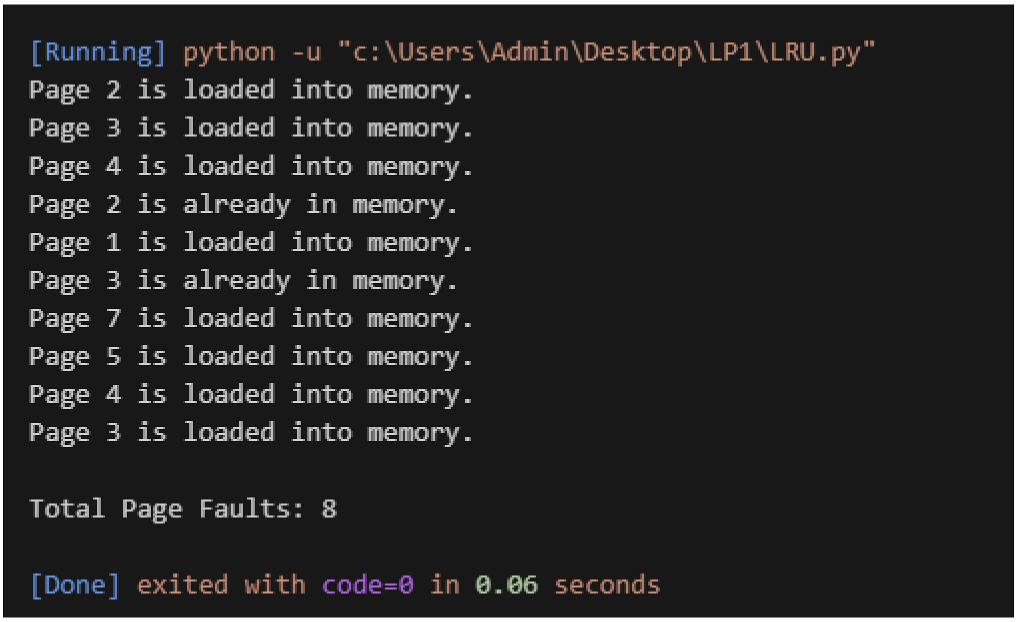
```
[Running] python -u "c:\Users\Admin\Desktop\LP1\Optimal.py"
Page 2 is loaded into memory.
Page 3 is loaded into memory.
Page 4 is loaded into memory.
Page 2 is already in memory.
Page 1 is loaded into memory.
Page 3 is already in memory.
Page 7 is loaded into memory.
Page 5 is loaded into memory.
Page 4 is already in memory.
Page 3 is already in memory.

Total Page Faults: 6

[Done] exited with code=0 in 0.059 seconds
```

LRU :-

```
from collections import OrderedDict
class LRUCache:
    def __init__(self, capacity):
        self.cache = OrderedDict()
        self.capacity = capacity
    def refer(self, page):
        if page in self.cache:
            # Move the page to the end to mark it as most recently used
            self.cache.move_to_end(page)
        else:
            # Check if the cache is full
            if len(self.cache) >= self.capacity:
                # Remove the least recently used page (the first item in the ordered dictionary)
                self.cache.popitem(last=False)
            # Add the new page to the cache
            self.cache[page] = None

def lru_page_replacement(pages, capacity):
    lru_cache = LRUCache(capacity)
    page_faults = 0
    for page in pages:
        if page not in lru_cache.cache:
            print(f"Page {page} is loaded into memory.")
            lru_cache.refer(page)
            page_faults += 1
        else:
            print(f"Page {page} is already in memory.")

    print(f"\nTotal Page Faults: {page_faults}")
```

```
if __name__ == "__main__":
    # Example usage
    page_references = [2, 3, 4, 2, 1, 3, 7, 5, 4, 3]
    memory_capacity = 3

    lru_page_replacement(page_references, memory_capacity)
```

OUTPUT :-



```
[Running] python -u "c:\Users\Admin\Desktop\LP1\LRU.py"
Page 2 is loaded into memory.
Page 3 is loaded into memory.
Page 4 is loaded into memory.
Page 2 is already in memory.
Page 1 is loaded into memory.
Page 3 is already in memory.
Page 7 is loaded into memory.
Page 5 is loaded into memory.
Page 4 is loaded into memory.
Page 3 is loaded into memory.

Total Page Faults: 8

[Done] exited with code=0 in 0.06 seconds
```