# Assignment # 1

# Name: Asif Khan

**No: 468895**

**BSDS 1-A**

# CPU Process Scheduling Algorithm Using Circular Linked List

## Approach

This implementation uses a **circular linked list** to model the process scheduling of a CPU, where each node represents a process with attributes like:

* process\_id: A unique identifier for the process.
* execution\_time: Total time required for the process to complete.
* remaining\_time: Time left for the process to complete.

The processes are scheduled based on **time slices** using **Round Robin Scheduling**. During each cycle:

1. The remaining time of each process is decremented by the time slice.
2. If a process's remaining time reaches 0 or below, the process is removed from the list.
3. A new process is added during the second cycle as an additional step to simulate dynamic process arrival.
4. The process continues until only one process remains in the list.

## Assumptions

* The scheduler uses **Round Robin Scheduling** with a fixed time slice (in this case, 2 units).
* Processes arrive at the start of execution except for process 6, which arrives during the second cycle.
* Each process has a fixed execution time, and the process is removed once its remaining time reaches zero.
* In each cycle, all processes run for the given time slice, and the list updates dynamically.

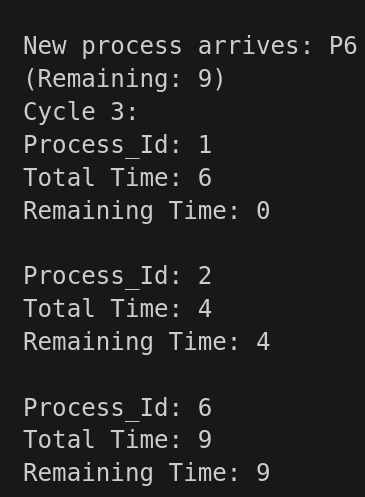
## Code Explanation

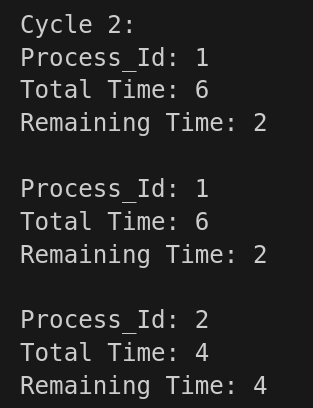
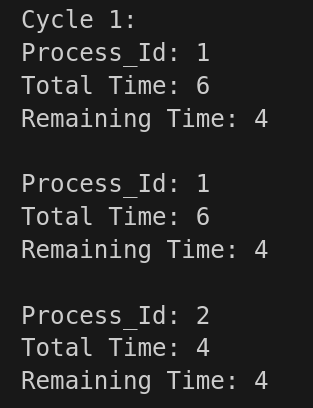
* **Insert Process:** Inserts a process into the circular linked list. If the list is empty, the head and tail are initialized.
* **Cycle:** Reduces the remaining time of all processes by the time slice and removes any processes whose remaining time is zero.
* **Scheduler:** Manages the entire scheduling process with multiple cycles and simulates the dynamic arrival of a new process during the second cycle.
* **Check Time:** Helper function that checks if a process has completed execution and removes it from the list.

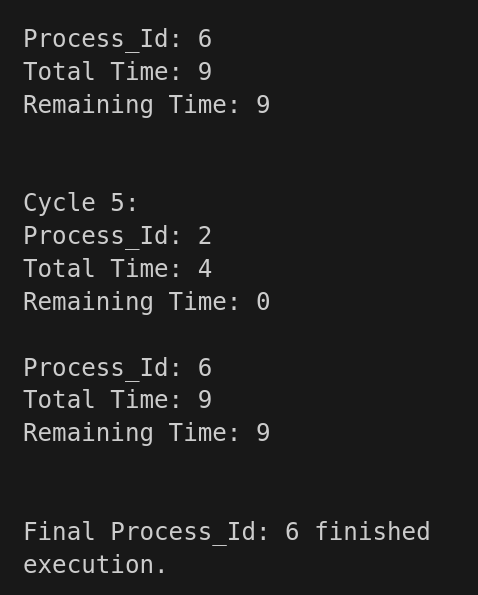
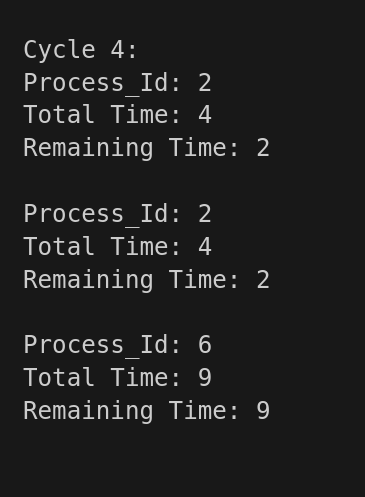
## Challenges Faced

* **Circular Linked List Complexity:** Handling edge cases such as the removal of the head or tail node while maintaining the circular structure.
* **Dynamic Process Addition:** Ensuring that new processes can be added during the execution without disturbing the current cycle.
* **Round Robin Algorithm:** Managing the logic to ensure each process gets an equal share of CPU time and efficiently removing completed processes.

## Screenshots







# Miller-Rabin Primality Test Using Linked List Storage for Large Numbers

## Approach

This project implements a system for handling large numbers using a **singly linked list**, where each node stores a 64-bit chunk of the number. The number is then processed using the **Miller-Rabin Primality Test** to check whether a given small prime number is prime.

### Key Features:

1. **Linked List Representation of Large Numbers**:  
   The BigNumber class uses a linked list where each node holds a 64-bit chunk of a large number.
2. **Reconstruction of the Number**:  
   The reconstruct method reconstructs the entire number from the linked list for easy access and further operations.
3. **Miller-Rabin Primality Test**:  
   The Miller-Rabin test is implemented for probabilistic primality testing of 64-bit numbers. The algorithm is repeated multiple times (NUM\_TESTS), and if a number passes all tests, it is considered probably prime.
4. **Modular Exponentiation**:  
   The modExp method efficiently computes (base^exp) % mod using **modular exponentiation**, which is crucial for the Miller-Rabin test.

### Primality Testing:

The Miller-Rabin test is applied to small primes for simplicity. In the current example, the number **11** is tested for primality using this method.

## Assumptions

* The linked list is designed for handling numbers as large as the sum of its 64-bit parts.
* The Miller-Rabin test is probabilistic, meaning it provides a high probability of primality but is not deterministic for large non-primes.
* A random base a is selected for the Miller-Rabin test from the range [2, n-2].

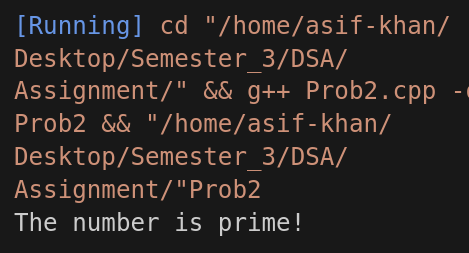
## Publicly Accessible GitHub Link

## Challenges Faced

* **Efficient Large Number Handling**: Storing large numbers as linked lists and manipulating them efficiently was challenging.
* **Modular Arithmetic**: Implementing modular exponentiation efficiently to handle large numbers was crucial to ensuring the correctness of the Miller-Rabin test.

## Screenshots

This is when we use a prime number:



This is when we do not use a prime number:

