

#### 4) SJF:

```
def sjf_preemptive(processes, arrival, burst):
    n = len(processes)
    complete = 0
    t = 0
    shortest = 0
    check = False

    wt = [0] * n
    tat = [0] * n
    rt = burst[:]
    gantt = []

    while complete != n:
        minm = 999999
        check = False
        for j in range(n):
            if arrival[j] <= t and rt[j] > 0 and rt[j] < minm:
                minm = rt[j]
                shortest = j
                check = True
        if not check:
            gantt.append("Idle")
            t += 1
            continue
        rt[shortest] -= 1
        gantt.append(processes[shortest])
        if rt[shortest] == 0:
            complete += 1
            finish_time = t + 1
            wt[shortest] = finish_time - burst[shortest] - arrival[shortest]
            if wt[shortest] < 0:
                wt[shortest] = 0
            t += 1

    for i in range(n):
        tat[i] = burst[i] + wt[i]

    print(f"\n{'Process':<10} {'Arrival':<10} {'Burst':<10} {'Waiting':<10} {'Turnaround':<12}")
    for i in range(n):
        print(f"{'processes[i]:<10'} {'arrival[i]:<10'} {'burst[i]:<10'} {'wt[i]:<10'} {'tat[i]:<12}")

    avg_wt = sum(wt) / n
    avg_tat = sum(tat) / n
    print(f"\nAverage Waiting Time: {avg_wt:.2f}")
    print(f"Average Turnaround Time: {avg_tat:.2f}")

    print("\nGantt Chart:")
    print("|", end="")
    for p in gantt:
        print(f" {p} |", end="")
    print()
    print("0", end="")
    for i in range(1, len(gantt) + 1):
        print(f" {i}", end="")
    print()

n = int(input("Enter number of processes: "))
processes = []
arrival = []
burst = []

for i in range(n):
    processes.append(input(f"Enter process name {i+1}: "))
    arrival.append(int(input(f"Enter arrival time for {processes[i]}: ")))
    burst.append(int(input(f"Enter burst time for {processes[i]}: ")))

print("\n--- SJF (Preemptive / SRTF) Scheduling ---")
sjf_preemptive(processes, arrival, burst)
```

#### 5) Priority:

```
def priority_scheduling(processes, burstTimes, priorities):
    n = len(processes)
    proc_data = sorted(zip(processes, burstTimes, priorities), key=lambda x: x[2])

    sorted_procs = [p[0] for p in proc_data]
    sorted_bursts = [p[1] for p in proc_data]
    sorted_priorities = [p[2] for p in proc_data]

    waitTime = [0] * n
    turnAround = [0] * n

    for i in range(1, n):
```

```

    waitTime[i] = waitTime[i-1] + sorted_bursts[i-1]

for i in range(n):
    turnAround[i] = waitTime[i] + sorted_bursts[i]

print(f"\n{'Process':<10} {'Priority':<10} {'Burst Time':<12} {'Waiting Time':<14} {'Turnaround Time':<16}")
for i in range(n):
    print(f"{'sorted_procs[i]:<10} {'sorted_priorities[i]:<10} {'sorted_bursts[i]:<12} {'waitTime[i]:<14} {'turnAround[i]:<16}")

avg_wt = sum(waitTime) / n
avg_tat = sum(turnAround) / n

print(f"\nAverage Waiting Time: {avg_wt:.2f}")
print(f"Average Turnaround Time: {avg_tat:.2f}")

print("\nGantt Chart:")
print(" | ".join(sorted_procs))
print("0", end="")
for t in turnAround:
    print(f" {t}", end="")
print()

n = int(input("Enter number of processes: "))
processes, burstTimes, priorities = [], [], []

for i in range(n):
    processes.append(input(f"Enter process name {i+1}: "))
    burstTimes.append(int(input(f"Enter burst time for {processes[i]}: ")))
    priorities.append(int(input(f"Enter priority for {processes[i]} (lower = higher priority): ")))

print("\n--- Priority (Non-Preemptive) Scheduling ---")
priority_scheduling(processes, burstTimes, priorities)

```

## 6) RoundRobin:

```

def round_robin(processes, burstTimes, quantum):
    n = len(processes)
    rem_bt = list(burstTimes)
    waitTime = [0] * n
    turnAround = [0] * n
    t = 0
    gantt = []
    timeline = []

    while True:
        done = True
        for i in range(n):
            if rem_bt[i] > 0:
                done = False
                gantt.append(processes[i])
                if rem_bt[i] > quantum:
                    t += quantum
                    rem_bt[i] -= quantum
                else:
                    t += rem_bt[i]
                    waitTime[i] = t - burstTimes[i]
                    rem_bt[i] = 0
                timeline.append(t)
            if done:
                break

    for i in range(n):
        turnAround[i] = waitTime[i] + burstTimes[i]

    print(f"\n{'Process':<10} {'Burst Time':<12} {'Waiting Time':<14} {'Turnaround Time':<16}")
    for i in range(n):
        print(f"{'processes[i]:<10} {'burstTimes[i]:<12} {'waitTime[i]:<14} {'turnAround[i]:<16}")

    print("\nGantt Chart:")
    for i, p in enumerate(gantt):
        print(p, end=" | " if i < len(gantt) - 1 else "\n")
    print("0", end="")
    for t in timeline:
        print(f" {t}", end="")
    print()

n = int(input("Enter number of processes: "))
processes, burstTimes = [], []

for i in range(n):
    processes.append(input(f"Enter process name {i+1}: "))
    burstTimes.append(int(input(f"Enter burst time for {processes[i]}: ")))

quantum = int(input("Enter time quantum: "))

```

```
print("\n--- Round Robin Scheduling ---")
round_robin(processes, burstTimes, quantum)
```

## 7) Mutex:

```
#include <iostream>
using namespace std;

class Sync {
    int buffer[5];
    int mutex;
    int empty;
    int full;
    int in, out;

    void wait(int &x) {
        while (x <= 0) {}
        x--;
    }

    void signal(int &x) {
        x++;
    }

public:
    Sync() : mutex(1), empty(5), full(0), in(0), out(0) {}

    void producer() {
        if (empty == 0) {
            cout << "Buffer full! Cannot produce.\n";
            return;
        }
        wait(empty);
        wait(mutex);

        cout << "Enter data to produce: ";
        int data;
        cin >> data;
        buffer[in] = data;
        in = (in + 1) % 5;
        cout << "Produced " << data << endl;

        signal(mutex);
        signal(full);
    }

    void consumer() {
        if (full == 0) {
            cout << "Buffer empty! Cannot consume.\n";
            return;
        }
        wait(full);
        wait(mutex);

        int data = buffer[out];
        out = (out + 1) % 5;
        cout << "Data consumed: " << data << endl;

        signal(mutex);
        signal(empty);
    }
};

int main() {
    Sync sc;
    int choice;
    do {
        cout << "\n1. Produce\n2. Consume\n3. Exit\nEnter choice: ";
        cin >> choice;
        switch (choice) {
            case 1: sc.producer(); break;
            case 2: sc.consumer(); break;
            case 3: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while (choice != 3);
    return 0;
}
```

## 8) BestFit:

```
def bestFit(blocks, files):
    allocation = [-1] * len(files)
    remaining = blocks.copy()
```

```

for i in range(len(files)):
    best_idx = -1
    for j in range(len(blocks)):
        if remaining[j] >= files[i]:
            if best_idx == -1 or remaining[j] < remaining[best_idx]:
                best_idx = j
    if best_idx != -1:
        allocation[i] = best_idx
        remaining[best_idx] -= files[i]
return allocation, remaining

# ----- MAIN -----
n_blocks = int(input("Enter number of memory blocks: "))
blocks = [int(input(f"Enter size of block {i+1}: ")) for i in range(n_blocks)]

n_files = int(input("\nEnter number of files: "))
files = [int(input(f"Enter size of file {i+1}: ")) for i in range(n_files)]

allocation, remaining = bestFit(blocks, files)

print("\n--- Best Fit Allocation ---")
print("Block No | Block Size | File No | File Size | Fragment Size")
print("-----")
for i in range(n_blocks):
    file_no = "-"
    file_size = "-"
    frag = "-"
    for j in range(len(files)):
        if allocation[j] == i:
            file_no = f"F {j+1}"
            file_size = files[j]
            frag = remaining[i]
    print(f"{i+1:^9}|{blocks[i]:^12}|{file_no:^9}|{file_size:^11}|{frag:^14}")

```

## 9) FirstFit:

```

def firstFit(blocks, files):
    allocation = [-1] * len(files)
    remaining = blocks.copy()

    for i in range(len(files)):
        for j in range(len(blocks)):
            if remaining[j] >= files[i]:
                allocation[i] = j
                remaining[j] -= files[i]
                break
    return allocation, remaining

# ----- MAIN -----
n_blocks = int(input("Enter number of memory blocks: "))
blocks = [int(input(f"Enter size of block {i+1}: ")) for i in range(n_blocks)]

n_files = int(input("\nEnter number of files: "))
files = [int(input(f"Enter size of file {i+1}: ")) for i in range(n_files)]

allocation, remaining = firstFit(blocks, files)

print("\n--- First Fit Allocation ---")
print("Block No | Block Size | File No | File Size | Fragment Size")
print("-----")
for i in range(n_blocks):
    file_no = "-"
    file_size = "-"
    frag = "-"
    for j in range(len(files)):
        if allocation[j] == i:
            file_no = f"F {j+1}"
            file_size = files[j]
            frag = remaining[i]
    print(f"{i+1:^9}|{blocks[i]:^12}|{file_no:^9}|{file_size:^11}|{frag:^14}")

```

## 10) NextFit:

```

def nextFit(blocks, files):
    allocation = [-1] * len(files)
    remaining = blocks.copy()
    start = 0

    for i in range(len(files)):
        count = 0
        j = start
        while count < len(blocks):
            if remaining[j] >= files[i]:
                allocation[i] = j

```

```

        remaining[j] -= files[i]
        start = j
        break
    j = (j + 1) % len(blocks)
    count += 1
return allocation, remaining

# ----- MAIN -----
n_blocks = int(input("Enter number of memory blocks: "))
blocks = [int(input(f"Enter size of block {i+1}: ")) for i in range(n_blocks)]

n_files = int(input("\nEnter number of files: "))
files = [int(input(f"Enter size of file {i+1}: ")) for i in range(n_files)]

allocation, remaining = nextFit(blocks, files)

print("\n--- Next Fit Allocation ---")
print("Block No | Block Size | File No | File Size | Fragment Size")
print("-----")
for i in range(n_blocks):
    file_no = "-"
    file_size = "-"
    frag = "-"
    for j in range(len(files)):
        if allocation[j] == i:
            file_no = f"F {j+1}"
            file_size = files[j]
            frag = remaining[i]
    print(f"{i+1:^9}|{blocks[i]:^12}|{file_no:^9}|{file_size:^11}|{frag:^14}")

```

### 11) WorstFit:

```

def worstFit(blocks, files):
    allocation = [-1] * len(files)
    remaining = blocks.copy()

    for i in range(len(files)):
        worst_idx = -1
        for j in range(len(blocks)):
            if remaining[j] >= files[i]:
                if worst_idx == -1 or remaining[j] > remaining[worst_idx]:
                    worst_idx = j
        if worst_idx != -1:
            allocation[i] = worst_idx
            remaining[worst_idx] -= files[i]
    return allocation, remaining

# ----- MAIN -----
n_blocks = int(input("Enter number of memory blocks: "))
blocks = [int(input(f"Enter size of block {i+1}: ")) for i in range(n_blocks)]

n_files = int(input("\nEnter number of files: "))
files = [int(input(f"Enter size of file {i+1}: ")) for i in range(n_files)]

allocation, remaining = worstFit(blocks, files)

print("\n--- Worst Fit Allocation ---")
print("Block No | Block Size | File No | File Size | Fragment Size")
print("-----")
for i in range(n_blocks):
    file_no = "-"
    file_size = "-"
    frag = "-"
    for j in range(len(files)):
        if allocation[j] == i:
            file_no = f"F {j+1}"
            file_size = files[j]
            frag = remaining[i]
    print(f"{i+1:^9}|{blocks[i]:^12}|{file_no:^9}|{file_size:^11}|{frag:^14}")

```

### 12) LRU:

```

def lru(pages, frame_size):
    frames = []
    page_faults = 0

    print("\n Step | Page | Frames after operation | Hit/Fault |")
    print("-----|-----|-----|-----")

    for i in range(len(pages)):
        page = pages[i]
        status = "Hit"

        if page not in frames:
            status = "Fault"

```

```

page_faults += 1

if len(frames) < frame_size:
    frames.append(page)
else:
    recent_use = []
    for f in frames:
        if f in pages[:i]:
            recent_use.append(len(pages[:i]) - 1 - pages[:i][::-1].index(f))
        else:
            recent_use.append(-1)
    replace_index = recent_use.index(min(recent_use))
    frames[replace_index] = page

print(f"| {i+1:<4} | {page:<4} | {str(frames):<26} | {status:<10} |")

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

# ----- MAIN -----
frame_size = int(input("Enter number of frames: "))
n = int(input("Enter number of pages: "))
pages = [int(input(f"Enter page {i+1}: ")) for i in range(n)]

print("\nLRU Page Replacement:\n")
lru(pages, frame_size)

```

### 13) OptimalPage:

```

def optimal(pages, frame_size):
    frames = []
    page_faults = 0

    print("\n| Step | Page | Frames after operation | Hit/Fault |")
    print("|-----|-----|-----|-----|")

    for i in range(len(pages)):
        page = pages[i]
        status = "Hit"

        if page not in frames:
            status = "Fault"
            page_faults += 1

        if len(frames) < frame_size:
            frames.append(page)
        else:
            future_use = []
            for f in frames:
                if f in pages[i+1:]:
                    future_use.append(pages[i+1:].index(f))
                else:
                    future_use.append(float('inf'))

            replace_index = future_use.index(max(future_use))
            frames[replace_index] = page

        print(f"| {i+1:<4} | {page:<4} | {str(frames):<26} | {status:<10} |")

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

# ----- MAIN -----
frame_size = int(input("Enter number of frames: "))
n = int(input("Enter number of pages: "))
pages = [int(input(f"Enter page {i+1}: ")) for i in range(n)]

print("\nOptimal Page Replacement:\n")
optimal(pages, frame_size)

```

## IOT

### 1) LED:

Let +ve (larger end ) to Gpio 3 Let -ve ( smaller) to resistor 1 end Resistor second end to pin 6 via breadboard

```

import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(3, GPIO.OUT)

```

```

while True:
    GPIO.output(3, True)
    time.sleep(1)
    GPIO.output(3, False)
    time.sleep(1)

```

## 2) IR SENSOR:

IR Sensor VCC → 3.3V (Pin 1), IR Sensor GND → GND (Pin 6), IR Sensor OUT → GPIO 18 (Pin 12), LED +ve → GPIO 3 (Pin 5), LED -ve → Resistor → GND (Pin 9).

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.IN)
GPIO.setup(5, GPIO.OUT)

try:
    while True:
        if GPIO.input(12) == 0:
            GPIO.output(5, True)
            print("Obstacle Detected!")
        else:
            GPIO.output(5, False)
            time.sleep(0.1)

except KeyboardInterrupt:
    GPIO.cleanup()
```

## 3) Camera:

```
Libcamera-hello
libcamera-still -o path/to/your/image -t 5000 (extension:jpg)
libcamera-vid -o path/to/your/vid -t 10000 (extension: h264)
```