PREEMPTIVE SCHEDULING:
Preemptive Scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process.

Examples of preemptive scheduling are:Round Robin Scheduling, Priority Scheduling and Shortest Job First Scheduling.(SJF)

NONPREEMPTIVE SCHEDULING:
(i)Nonpreemptive scheduling ensures that a process relinquishes the control of the CPU only when it finishes with its current CPU burst.
(ii)If nonpreemtive scheduling is used in a computer centre, a process is capable of keeping other processes waiting for long time.

Examples of nonpreemptive scheduling are:Shortest Job First(SJF basically nonpreeemptive), Priority(nonpreemptive version),etc

DIFFERENCES BETWEEN PREEMPTIVE AND NONPREEMPTIVE SCHEDULING:

1) In preemptive scheduling the CPU is allocated to the processes for the limited time whereas in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to waiting state.

2) The executing process in preemptive scheduling is interrupted in the middle of execution when higher priority one comes whereas, the executing process in non-preemptive scheduling is not interrupted in the middle of execution and wait till its execution.

3) In Preemptive Scheduling, there is the overhead of switching the process from ready state to running state, vise-verse, and maintaining the ready queue. Whereas in case of non-preemptive scheduling has no overhead of switching the process from running state to ready state.

4) In preemptive scheduling, if a high priority process frequently arrives in the ready queue then the process with low priority has to wait for a long, and it may have to starve. On the other hands, in the non-preemptive scheduling, if CPU is allocated to the process having larger burst time then the processes with small burst time may have to starve.

In our question they said to use only nonpreemptive scheduling..

To find waiting time: Time taken by all processes before the current process to be started (i.e. burst time of all previous processes) – arrival time of current process
**wait_time[i] = (bt[0] + bt[1] +…… bt[i-1] ) – arrival_time[i]**
**Implementation:**

```
1- Input the processes along with their burst time(bt)
```

```
   and arrival time(at)
2- Find waiting time for all other processes i.e. for
   a given process  i:
       wt[i] = (bt[0] + bt[1] +..... bt[i-1]) - at[i]
3- Now find turn around time
         = waiting_time + burst_time for all processes
4- Average waiting time =
                    total_waiting_time / no_of_processes
5- Average turn around time =
                  total_turn_around_time / no_of_processes
```

a) Execute in order arrived to the CPU
   P1 → P2 → P3 = ( 8 + (12 - 0.4) + (13 - 1)) / 3 = 10.53

```
// Function to calculate turn around time
static void findTurnAroundTime(int []processes, int n, int[] bt,
                                    int []wt, int[] tat)
{
    // Calculating turnaround time by adding bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}
// Function to calculate turn-around time.

static void findavgTime(int []processes, int n, int []bt, int []at)
{
     int []tat = new int[n];

  // Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with all details
    Console.Write("Processes " + " Burst Time " + " Arrival Time "
        + " Waiting Time " + " Turn-Around Time "
        + " Completion Time \n");
    int  total_tat = 0;
    for (int i = 0 ; i < n ; i++)
    {
        total_tat = total_tat + tat[i];
        int compl_time = tat[i] + at[i];
        Console.WriteLine(i+1 + "\t\t" + bt[i] + "\t\t"
            + at[i] + "\t\t" + wt[i] + "\t\t "
            + tat[i] + "\t\t " + compl_time);
    }

    Console.Write("\nAverage turn around time = "
        + (float)total_tat / (float)n);
}
```

**Output:**

```
Processes  Burst Time  Arrival Time  Waiting Time  Turn-Around Time
 1        5        0        0         5
 2        9        3        2         11
 3        6        6        8         14
Average turn around time = 10.0
```

b) P1 executes and then we find P3 is a shorter process than P2.
P1 → P3 → P2 = ( 8 + (9 - 1) + (13 - 0.4)) / 3 = 9.5

**How to compute below times in SJF using a program?**

1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time.
   Waiting Time = Turn Around Time – Burst Time

```
// C++ program to implement Shortest Job first with Arrival Time
#include<iostream>
using namespace std;
int mat[10][6];

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
    for(int i=0; i<num; i++)
    {
        for(int j=0; j<num-i-1; j++)
        {
            if(mat[j][1] > mat[j+1][1])
            {
                for(int k=0; k<5; k++)
                {
                    swap(mat[j][k], mat[j+1][k]);
                }
            }
        }
    }
}

void completionTime(int num, int mat[][6])
{
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];
```

```
    for(int i=1; i<num; i++)
    {
        temp = mat[i-1][3];
        int low = mat[i][2];
        for(int j=i; j<num; j++)
        {
            if(temp >= mat[j][1] && low >= mat[j][2])
            {
                low = mat[j][2];
                val = j;
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for(int k=0; k<6; k++)
        {
            swap(mat[val][k], mat[i][k]);
        }
    }
}

int main()
{
    int num, temp;

    cout<<"Enter number of Process: ";
    cin>>num;

    cout<<"...Enter the process ID...\n";
    for(int i=0; i<num; i++)
    {
        cout<<"...Process "<<i+1<<"...\n";
        cout<<"Enter Process Id: ";
        cin>>mat[i][0];
        cout<<"Enter Arrival Time: ";
        cin>>mat[i][1];
        cout<<"Enter Burst Time: ";
        cin>>mat[i][2];
    }

    cout<<"Before Arrange...\n";
    cout<<"Process ID\tArrival Time\tBurst Time\n";
    for(int i=0; i<num; i++)
    {
        cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\n";
    }

    arrangeArrival(num, mat);
    completionTime(num, mat);
    cout<<"Final Result...\n";
    cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";
    for(int i=0; i<num; i++)
    {
```

```
        cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\t\t"<<mat[i][4]<<"\t\t"<<mat[i]
    }
}
```

Output:

```
Process ID      Arrival Time    Burst Time
1               2               3
2               0               4
3               4               2
4               5               4
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
2               0               4               0               4
3               4               2               0               2
1               2               3               4               7
4               5               4               4               8
```

c)Idle for 1, then use SJF. P3 shortest, P2 next shortest, P1 longest.
Idle → P3 → P2 → P1 = ((2 - 1) + ( 6 - 0.4 ) + ( 14 - 0)) / 3 = 6.87

It see all the burst times of the processes which one is least that should be processed first.

Here Job 3 having less burst time of all process,so it is executed according to the Optimal non preemptive scheduling algorithm.

So system waits until '1' unit of time because Job 3 comes at A.T of 1.

After completion of J3 all processes are arrived and execute according least burst time