	SVKM's NMIMS School of Technology Management & Engineering Navi Mumbai Campus
	Department of Computer Engineering

Name: Rahul Purohit	**SAP ID:** 70562100106
Semester: III	**Year**: II
Subject: Operating systems	**Roll No**.: A022
Batch: 2	**Date**:

First Come First Serve algorithm:

FCFS Scheduling algorithm automatically executes the queued processes and requests in the order of their arrival. It allocates the job that first arrived in the queue to the CPU, then allocates the second one, and so on. FCFS is the simplest and easiest CPU scheduling algorithm, managed with a FIFO queue. FIFO stands for First In First Out. The FCFS scheduling algorithm places the arriving processes/jobs at the very end of the queue. So, the processes that request the CPU first get the allocation from the CPU first. As any process enters the FIFO queue, its Process Control Block (PCB) gets linked with the queue's tail. As the CPU becomes free, the process at the very beginning gets assigned to it. Even if the CPU starts working on a longer job, many shorter ones have to wait after it. The FCFS scheduling algorithm works in most of the batches of operating systems.

Code:

```cpp
#include<iostream>
#include <algorithm>
#include<iomanip>

using namespace std;

struct process_struct {
    int pid;
    int at;
    int bt;
    int ct, wt, tat, start_time;
} ps[100];
bool comparatorAT(struct process_struct a, struct process_struct b) {
    int x = a.at;
    int y = b.at;
    return x < y;
}

bool comparatorPID(struct process_struct a, struct process_struct b) {
    int x = a.pid;
    int y = b.pid;
    return x < y;
}

int main() {
    int n;
    cout << "Enter total number of processes: ";
    cin >> n;
    float sum_tat = 0, sum_wt = 0;
    int length_cycle, total_idle_time = 0;

    cout << fixed << setprecision(2);
    for (int i = 0; i < n; i++) {
        cout << "\nEnter Process " << i << " Arrival Time: ";
        cin >> ps[i].at;
        ps[i].pid = i;
    }

    for (int i = 0; i < n; i++) {
        cout << "\nEnter Process " << i << " Burst Time: ";
        cin >> ps[i].bt;
    }

    sort(ps, ps + n, comparatorAT);

    for (int i = 0; i < n; i++) {
        ps[i].start_time = (i == 0) ? ps[i].at : max(ps[i].at, ps[i - 1].ct);
        ps[i].ct = ps[i].start_time + ps[i].bt;
        ps[i].tat = ps[i].ct - ps[i].at;
        ps[i].wt = ps[i].tat - ps[i].bt;

        sum_tat += ps[i].tat;
        sum_wt += ps[i].wt;
        total_idle_time += (i == 0) ? 0 : (ps[i].start_time - ps[i - 1].ct);
    }
    length_cycle = ps[n - 1].ct - ps[0].start_time;

    sort(ps, ps + n, comparatorPID);

    cout << "\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\n";
    for (int i = 0; i < n; i++)
        cout << i << "\t\t" << ps[i].at << "\t" << ps[i].bt << "\t\t" << ps[i].ct << "\t" << ps[i].tat << "\t" << ps[i].wt << endl;
    cout << endl;

    cout << "\nAverage Turn Around time= " << sum_tat / n;
    cout << "\nAverage Waiting Time= " << sum_wt / n;

    return 0;
}
```

Output:

```
Enter total number of processes: 5

Enter Process 0 Arrival Time: 0

Enter Process 1 Arrival Time: 1

Enter Process 2 Arrival Time: 2

Enter Process 3 Arrival Time: 3

Enter Process 4 Arrival Time: 4

Enter Process 0 Burst Time: 2

Enter Process 1 Burst Time: 6

Enter Process 2 Burst Time: 4

Enter Process 3 Burst Time: 9

Enter Process 4 Burst Time: 12

Process No.     AT      CPU Burst Time  CT      TAT     WT
0               0       2               2       2       0
1               1       6               8       7       1
2               2       4               12      10      6
3               3       9               21      18      9
4               4       12              33      29      17


Average Turn Around time= 13.20
Average Waiting Time= 6.60
```