USECASE

Aim: Efficiently schedule multiple cloud tasks onto servers to minimize total execution time (make span) while keeping servers within memory limits.

Algorithm:

1. Sort all tasks by required CPU cycles (largest first).
2. For each task:
    - Find the server that can fit it (enough memory) and will finish earliest if assigned there.
3. Assign the task to that server and update its load time and used memory.
4. After assigning all tasks, compute the make span = max load among servers.

Program:

```
import java.util.*;


public class CloudSchedulerMini {


  static class Task {
    String id; double cpu, mem;
    Task(String id, double cpu, double mem) { this.id = id; this.cpu = cpu; this.mem = mem; }
  }


  static class Server {
    String id; double speed, totalMem, usedMem = 0, load = 0;
    List<Task> tasks = new ArrayList<>();
    Server(String id, double speed, double totalMem) {
      this.id = id; this.speed = speed; this.totalMem = totalMem;
    }
  }
```

```
public static void main(String[] args) {
    // Sample Servers (id, speed, memory)
    List<Server> servers = Arrays.asList(
            new Server("S1", 5_000_000, 16000),
            new Server("S2", 3_000_000, 8000),
            new Server("S3", 8_000_000, 32000)
    );

    // Sample Tasks (id, cpuCycles, memory)
    List<Task> tasks = Arrays.asList(
            new Task("T1", 40_000_000, 1000),
            new Task("T2", 10_000_000, 500),
            new Task("T3", 120_000_000, 2000),
            new Task("T4", 70_000_000, 4000),
            new Task("T5", 25_000_000, 1000)
    );

    // Sort tasks by CPU demand (descending)
    tasks.sort((a, b) -> Double.compare(b.cpu, a.cpu));

    for (Task t : tasks) {
        Server best = null;
        double bestFinish = Double.MAX_VALUE;

        for (Server s : servers) {
            if (s.totalMem - s.usedMem >= t.mem) {
                double finish = s.load + (t.cpu / s.speed);
                if (finish < bestFinish) {
                    best = s;
                    bestFinish = finish;
                }
```

```
            }
        }

        if (best != null) {
            best.tasks.add(t);
            best.usedMem += t.mem;
            best.load += t.cpu / best.speed;
        }
    }

    // Print results
    double makespan = 0;
    for (Server s : servers) {
        System.out.println(s.id + " -> " + s.tasks.stream().map(t -> t.id).toList());
        makespan = Math.max(makespan, s.load);
    }
    System.out.printf("Total Execution Time (Makespan): %.4f seconds%n", makespan);
  }
}
```

Output:

S1 -> [T3]

S2 -> [T2, T5]

S3 -> [T1, T4]

Total Execution Time (Makespan): 15.5000 seconds

Result: Thus, efficiently scheduling multiple cloud tasks onto servers to minimize total execution time (makespan) while keeping servers within memory limits is successfully executed.