```julia
### s194624.jl

using Random

include("./IO.jl")
include("./SolutionBuilder.jl")

struct ArgumentException <: Exception
    message::String
end

function main()
    # Check arguments
    instanceLocation = ARGS[1]
    solutionLocation = ARGS[2]
    maxTime = parse(Int, ARGS[3])

    name, dim, LB ,rev, rev_pair, k, H, p = read_instance(instanceLocation)

    if (ARGS[2] == " ")
        solutionLocation = string("sols/", name, ".sol")
    end

    # Initialize solution
    sol = [Int[] for i in 1:k]
    revenue = 0
    remainingProducts = [i for i in 1:dim]
    manufacturingTimes = Int[0 for i in 1:k]

    destroyFunctions = [destroyPercentage, destroyPercentageAllProductionLines, destroyElementWithLeastRevenue]
    repairFunctions = [repairRandom, repairBestInsert]

    sol, revenue, manufacturingTimes, remainingProducts = repairRandom(sol, revenue, manufacturingTimes, remainingProducts, dim, rev, rev_pair, k, H, p)

    # ALSN
    println()
    println("Running ALSN for ", maxTime, " seconds...")
    println()

    iterations = 0
    elapsedTime = 0
    start = time_ns()

    while (elapsedTime < maxTime)

        # Print status message every 3 seconds
        if (elapsedTime % 3 == 0)
            println("\r", "Elapsed time: ", elapsedTime, "s, Iterations: ", iterations, ", Revenue: ", revenue, ", Lower bound: ", LB)
        end

        destroyFunction = destroyFunctions[rand(1:length(destroyFunctions))]
        repairFunction = repairFunctions[rand(1:length(repairFunctions))]

        newSol, newRevenue, newManufacturingTimes, newRemainingProducts =
            destroyFunction(sol, revenue, manufacturingTimes, remainingProducts, dim, rev, rev_pair, k, H, p)
        newSol, newRevenue, newManufacturingTimes, newRemainingProducts =
            repairFunction(newSol, newRevenue, newManufacturingTimes, newRemainingProducts, dim, rev, rev_pair, k, H, p)

        if (newRevenue > revenue)
            sol = newSol
            revenue = newRevenue
            manufacturingTimes = newManufacturingTimes
            remainingProducts = newRemainingProducts
        end
        elapsedTime = round((time_ns()-start)/1e9,digits=3)
        iterations += 1
    end

    println()
    println("Iterations: ", iterations)
    println("Final solution with revenue: ", revenue)
    println("Lower bound: ", LB)

    writeSolution(sol, solutionLocation)

    output = run(Cmd(["./POChecker.exe", instanceLocation, solutionLocation]))
    println(output)
end
main()
```