# Lab 6: Creating Parallelism using a Recursive Task Cutoff Strategy

Instructor: Mack Joyner

**Course Wiki:** http://comp322.rice.edu

**Staff Email:** comp322-staff@mailman.rice.edu

## Goals for this lab

- Creating Parallelism using a Recursive Task Cutoff Strategy

## Lab Projects

Today, we will solve the simple problem of computing a sum of reciprocal array values of doubles, by using futures with a cutoff strategy to create parallelism. We will evaluate the performance in this lab using real performance numbers on your local machine. We will not use NOTS for this lab.

The GitHub classroom signup for this lab is located here:

https://classroom.github.com/a/AyJUy9i4

For instructions on checking out this repository through IntelliJ or through the command-line, please see the Lab 1 handout. The below instructions will assume that you have already checked out the lab6 folder, and that you have imported it as a Maven Project if you are using IntelliJ.

## 1 Parallelization of Reciprocal Array Sum using a Recursive Task Cutoff Strategy

We will work with the simple parallel array sum program introduced in the Demonstration Video for Topic 1.1. Edit the `ReciprocalArraySum.java` program provided in your GitHub repository. There is one `TODO` in the `ReciprocalArraySum.java` file guiding you on where to place your edits.

The goal of this exercise is to create arrays of random doubles totaling `N` elements, and compute the sum of their reciprocals. Performance in this lab will be measured using *real* performance.

You'll need to implement the `parArraySumCutoff` using futures and a cutoff threshold. Once the array size is equal to or less than the threshold, you should use `seqArraySum`. Each call `seqArraySum` in `parArraySumCutoff` will create it's own array.

There is one `TODO` in the `Lab6CorrectnessTest.java` file. You'll need to find the optimal cutoff strategy for your local machine. You can test various cutoff thresholds by passing the thresholds to `parArraySumCutoff` and timing how long it takes to compute the result. If you want to determine how many threads you have available, you can use `Module0.numWorkerThreads()`.

## 2 Turning in your lab work

For lab6, you will need to turn in your work before Friday, March 14, 2024 at 4pm, as follows:

1. Show your test passes by demonstrating that the parallel version of the algorithm has better performance the sequential version. You should output the sequential time, parallel time, and speedup.

2. Commit and push your work to your `lab6` GitHub Classroom repository. Check that all the work for today's lab is in your `lab6` directory by opening https://classroom.github.com/a/AyJUy9i4 in your web browser and checking that your changes have appeared.