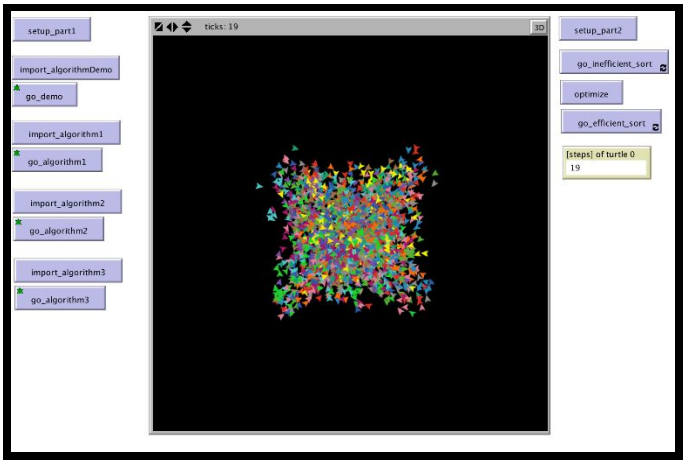
CS108L Computer Science for All
Module 6 (Part 2) Guide
Algorithms Part 2



Setup:

Do the following first:

1. Turn on world-wrapping if you haven't already.
2. You should already have a turtles-own variable from part 1. Now, add one more variable in the turtles-own section:
    a. Create a variable that will store a set of other turtles with the same color, like **turtlesWithMyColor**.
3. Setup: Make a setup_part2 button and a corresponding procedure that clears the world, resets ticks, creates 2,000 turtles, and sets those turtles to random locations.

Inefficient Sort Procedure:

The algorithm for the inefficient method is:

1. Each turtle has a color. You can create a local variable to keep track of that color for you (e.g. **myColor**).
2. Each turtle looks at the other 1,999 turtles one at a time to see what color it is. If the other turtle has the same color then it is put into a NetLogo **agentset** (a list or set of

agents). An **agentset** lets you keep track of a group of agents so you can use that group later.

3. The turtle then randomly chooses one of the other turtles of the same color (in the created agentset) and names it (e.g. **myTarget**).
4. The turtle turns to face the turtle chosen in step 3 (Hint: set heading towards **myTarget**)
5. The turtle takes one patch size step forward.

How to do Inefficient Method Steps 2 and 3:

The simplest way to do steps 2 and 3 of the inefficient method is to chain a few of NetLogo's **reporters** together with the following commands:

one-of *agentset*	Given an agentset as input, **one-of** selects and reports a random agent from that set. If the agentset is empty, the **one-of** reports **nobody**.
other *agentset*	Netlogo's **other** reporter is used to report a new agentset that is the same as the agentset it is given except with "this" turtle (the turtle in the current iteration of **ask turtles**) removed. The **other** reporter is used in this model because we want each turtle to move toward a turtle of the same color, *but not pick itself* as the turtle to move toward.
turtles	Reports the agentset consisting of all turtles.
agentset **with [***reporter***]**	The **with** reporter takes two inputs: on the left, an agentset (usually "turtles" or "patches"). On the right, the reporter must be a Boolean reporter. Given these inputs, **with** reports a new agentset containing only those agents that reported true -- in other words, the agents satisfying the given condition.

Putting this all together, we can build the powerful NetLogo statement:

let myTarget one-of other turtles with [color = MyColor]

Optimize Button and Efficient Sort Procedure:

Let's make an "optimize" button, and **have it build the agentset in advance**. Let's also have it do it only once rather than every tick. This button will build an agentset for every turtle that contains all the other turtles of the same color. Then we can pick a random turtle from the agentset and:

1. Store turtle's color in a local variable.
2. Pick a target from your created agentset.

3. The turtle turns to face the chosen target.
4. The turtle takes one patch size step forward.

Algorithm Complexity Analysis:

When you get this to work, you will see that it saves lots of time. For one turtle to create its agentset of like colored turtles, that turtle must examine the color of **all other turtles**. Thus, the more turtles there are, the longer it takes for one turtle to build its agentset.

Let's use an abstract concept of time, where each operation in our code takes 1 unit of time. These operations can be mathematical, like addition or multiplication, or comparisons, like checking to see if an item is in a set.

Let n be the number of turtles (in this lab n = 2,000). Each turtle must build its own agentset of turtles with its same color, which involves comparing itself to every other turtle. Because there are n turtles, this takes n operations, or n units of time. We say this is "on the order of n", or O(n).

That's just for one turtle. Now, we have to do this for every turtle. So n turtles build agentsets that take O(n) time. That will require O($n * n$) or O(n^2) operations (in this case about 4 million operations)! In the inefficient method, this is done **every tick**. When you use the efficient method, your optimize procedure makes it happen **only once**.

A Note on Sets versus Lists:

In Netlogo, there are things called **sets** and different things called **lists**. An **agentset** is a set that contains only agents. In casual English, the words set and list are often used interchangeably. In computer science, however, these words have very different meanings.

A **set** is an **unordered collection** where any **repeated elements make no difference** to the set. For example, the sets: {1, 2, 3, 4} and {4, 2, 1, 3} are the same. Also, if 2 is added to the set {1, 2, 3, 4}, then the resulting set is still {1, 2, 3, 4} since 2 was already an element of the set. With a set, it makes no sense to ask "what is the first element" since none of the elements have any particular order.

A **list** is an **ordered collection** where **repeated elements DO make a difference**. Thus, the lists, [1, 2, 3, 4] and [4, 2, 1, 3], are different. If 2 is added to the list, [1, 2, 3, 4], it is important to ask **where** the 2 is added because the lists [2, 1, 2, 3, 4], [1, 2, 2, 3, 4], [1, 2, 3, 4, 2], etc. are each different from one another.