Name(s)_____ Period _____ Date _____

Activity - Card Searching Algorithm

Analyzing Algorithms

Linear Search
A common way to process a list is to find out if it contains a specific item. We implemented one algorithm to do this called **linear search**. Linear search is pretty simple. You start at the beginning of the list, look at every item, one at a time, and see if it matches what you are looking for. Stop once you've found it, or when there are no more items to consider.

Cost and Efficiency
In computer science, we measure the "cost" or "efficiency" of an algorithm by how much work - roughly, how many primitive operations - the computer has to execute to arrive at the answer. All computers perform roughly the same set of primitive commands, but some are much faster at running them than others. Measuring the total number of primitive commands, rather than the time it takes to run a program, is a better way to compare one algorithm to another, since it does not depend on the speed of the computer running it.

Worst-Case Analysis
The same algorithm might take a different amount of time to run based on the input it is given. Consider linear search. This algorithm will likely require more work to run on larger lists, since we may need to look at every item. At the same time, we could always get lucky and find the item in the first place we look. To make it easier to compare efficiencies of algorithms, we usually **consider the worst-case scenario**. In other words, we consider the input that would cause the algorithm to do the most work. Then we can guarantee that our algorithm can't do worse than that.

Worst-Case Analysis of Linear Search
The worst case for linear search (and most searching algorithms) is that the item you're looking for is not in the list. This would make the algorithm look at every item once to verify that it wasn't there. We typically talk about the efficiency of an algorithm **in comparison to the size of the input, or data, that it must consider.** Therefore, we would say that if a list has *N* items, linear search requires that you look at all *N* items in the worst case. (For most searching algorithms, the only primitive command we consider is accessing an item in the list.) Based on this analysis, we can compare linear search to other searching algorithms.

Searching on Sorted Lists

Different Algorithms for Different Inputs
Linear search can find an item in any list, no matter how it's ordered. If we know the items in the list are in **sorted order**, however, then there are more efficient algorithms we could use to search for an item.

Challenge: Develop a searching algorithm for a sorted list
Develop an algorithm that searches for an item **in a sorted list.** Your algorithm should be as efficient as possible, as measured by a **worst-case analysis**. Note the list [1, 2, 3, 4, 5] is sorted, but so is [1, 1, 1, 1, 2] and [1, 2, 2, 2, 2].

How To Do It
On the next page you have space to **write out your algorithm in pseudocode**. Your algorithm should be written to work on any list, but you might want some manipulatives, such as **a deck of cards,** that you can use to test out your ideas as you go. If you are having trouble expressing your algorithm in pseudocode, it is fine to just describe it so that a friend could run it on a small row of cards.

My Algorithm for Searching a Sorted List

Write the steps of your algorithm below.

Analyze Your Algorithm

Worst-Case Analysis

Share your algorithm with a classmate. Once you understand one another's algorithms, try to come up with the worst-case input for each algorithm. What is the sorted list that would make it perform the most work?

Quantify Your Results

For the purposes of comparison, we'll consider the most important primitive command when comparing searching algorithms: **accessing an item in a list**. In the worst case, we know that linear search must look at every item in a list. Try to determine **how many items your algorithm must look at in the worst case** for the different-sized inputs shown below.

Number of Items Looked at in the Worst Case						
Algorithm	**8 items**	**16 items**	**32 items**	**64 items**	**100 items**	**1,000 items**
Linear Search	8	16	32	64	100	1,000
Your Algorithm						