☼Hi, welcome to Algorithm Design: Searching.

In this lesson, we will be using searching as an example to illustrate the concept of algorithm design. Searching is the technique of selecting specific data from a collection of data based on some conditions. We are familiar with this concept from our experience in performing file search to locate files and folders by title, key words, modified date, or size, web searches to locate pages containing words or phrases or when looking up a phone number in the telephone contacts. To start from basic knowledge, in this lesson, we will restrict the term searching to refer to the process of finding a particular item in a collection.

☼**At the end of this lesson, you should be able to:**

- Describe the process of searching
- Explain the importance of different types of searching algorithms
- Search for a given value in an array using linear search and binary search.
- Apply Search algorithms in problem solving
- Recognize that "no single" best search algorithm applies to all scenarios

☼ We will go through the following topics in this lesson.

☼ Thinking about searching the list of Food & Beverage in North Spine Plaza to find out if pizza hut is operating there.

Naturally , • We will start at the first item

- is it the one I am looking for?
- if not, goes to next item
- repeats until found or all the items are checked

☼ Actually, this simple thought is an algorithm called Linear search, which is the most basic kind among search methods. This technique iterates over the sequence, one item at a time, until the specific item is found or all items have been examined .

The element that needs to be found is called a search key.

linear search, also called sequential search

- is an intuitive approach
- starts at the first item
- is it the one I am looking for?
- if not, goes to next item
- repeats until found or all the items are checked

this approach is necessary if items are not sorted.

☼How does it work for a sorted list? Still the same, This time, burger king is our search key, we scan the list from the beginning, comparing each entry with the target name, if we find the

target name, the search terminates as a success. However, if we reach the end of the list without finding the target, our search is a failure. We searched the whole list and found burger king is not in North Spine Plaza. We can feel that the algorithm is inefficient.

☼In fact, if we reach a name greater than the target name without finding the target, our search terminates as a failure. Remember, the list is arranged in alphabetical order, so reaching a name greater than the target name indicates that the target does not appear in the list.

The number of comparisons we need to perform depends on the total length of the list, and also whether the element we are looking for is near the beginning or near the end of the list. In the worst-case scenario, if our element is the last element of the list, we will have to search through the entire list to find it. The linear search algorithm for a sorted sequence produced a slight improvement over the linear search with an unsorted sequence, but both have a linear time-complexity in the worst case.

☼From the above example and analysis , we can get that Given a list of data, searching is finding the location of a particular value or reporting that value is not present.

Similar to sorting, searching is one of the Fundamental problems in computer science and programming.

Sorting is done to make searching easier.

Again just like sorting, there are multiple different algorithms that are used to solve the same problem.

How do we know which algorithm is "better"? There is no single “best” algorithm for all scenarios. Let’s explore other algorithms.

☼To improve the search time for a sorted sequence, we can modify the search technique itself.

Thinking about searching a dictionary . We do not perform a sequential entry-by-entry or even a page-by-page procedure . Rather, we begin by opening the dictionary directly to a page in the area where we believe that the target entry is located. If we are lucky, we will find the target value there; otherwise, we must continue searching. But at this point we will have narrowed our search considerably.

Of course, in the case of searching a dictionary, we have prior knowledge of where words are likely to be found. If we are looking for the word “copyright”, we would start by opening to the beginning portion of the dictionary. In the case of generic lists, however, we do not have this advantage,

☼So let us agree to always start our search with the middle entry in the list.

Here, we write the word middle in quotation mark because the list might have an even number of entries and thus, no middle entry in the exact sense.

Is the middle entry in the list the target value? If yes, we can declare the search a success. Otherwise, we can at least restrict the search process to the half of the list. If the middle entry is less than the target, move to second half of list. If the middle entry is more than the target, move to first half of list

repeat until found or sub list size becomes 0

So, if items are sorted, then we can *divide and conquer*

That is, dividing your work in half with each step. It is the concept of binary search.

- it is generally a good thing

☼This approach of the searching process is illustrated in this slide.

First, locate the middle item of the list and compare it with the search key. If they match, the search key is found, the search is successful. we immediately terminate the search and return True. Otherwise, we determine if the target is less than the item at the midpoint or greater. If it is less, we adjust the high marker to be one less than the mid-point, and if it is greater, we adjust the low marker to be one greater than the midpoint. In the next iteration of the loop, the only portion of the sequence considered are those elements between the low and high markers, as adjusted. Assume lower,we move to first half of list. And locates the new middle item. This process is repeated until the item is found, which indicates a successful search, or there are no items left to be processed, indicating the target is not in the sorted sequence.

☼Let's compare linear search and Binary search using "Hi-Low" Number Guessing Game.

You are supposed to guess what is the number that someone is thinking of, which is between 1 to 19, you can adjust your guess by asking smaller-or-larger questions.

Of course, you could do it in 19 guesses using linear search, but how many do you really need?

To make the most of your questions, you can also try to cut the number of possibilities in half. For example, you need only a few questions. The first one is something like "Is the number greater than 10?" If it is, then you ask, "Is it greater than 15?" You keep halving the interval until you find the number. This is the concept of binary search.

From the animation, we can see that binary search is a more efficient search algorithm which relies on the elements in the list being sorted.

The difference with the binary search is that not every item in the sequence has to be examined before determining that the target is not in the sequence, even in the worst case. Since the sequence is sorted, the input size is repeatedly reduced by half during each iteration of a loop, there will be log n iterations in the worst case. Thus, the binary search algorithm has a worst case time-complexity of O(log n), which is more efficient than the linear search. You will learn more about algorithm complexity in the next topic.

☼There are many other Searching Algorithms available. You can explore by yourself if you are interested in them.

☼Quick summary: We have discussed the concept of two algorithms that perform searching:

- linear search,
- which simply checks the items in sequence until the desired item is found
- Because of its simplicity, Linear search is often used for short lists. However, it is inefficient for large and sorted lists.
-
- binary search,
- which requires a sorted sequence of list,
- and checks for the value in the middle of the list,
- repeatedly discarding the half of the list which contains values which are definitely either all larger or all smaller than the desired value.

I hope you enjoy this lecture.

And, I will see you next time.