**Selection Sort**

**Concept | Examples | Java Implementation**

Getting Started: In this lesson you will be introduced to the world of algorithms, by learning a crucial sorting algorithm: selection sort. We will cover the concept of the algorithm, then go through an example, and finally implement the algorithm in Java with another example.

What is Selection Sort?

Selection sort, as the name suggests, is a sorting algorithm. This means that it can be used to sort an array of a certain data type. For simplicity’s sake, this lesson will be sorting integers to explain the algorithm, but other data types can also easily be sorted once you understand the algorithm.

Selection sort is one of many sorting algorithms. The reason for so many different algorithms existing is because of Big O notation, an interesting term that will be consistently brought up in many computer science classes. Aside from Big O notation, covered in the next section, there are also other factors that impact the algorithms you learn. For example, some algorithms have simpler concepts in comparison to other algorithms. Selection sort is nowhere near the best algorithm to use in regards to Big O notation, but is a great algorithm to get your foot in the door with sorting algorithms.

Big O Notation

Big O notation refers to the limiting space and time complexity of a certain function. In simpler terms, it can be used to compare algorithms based on how much space and time they use in the worst-case.

Selection sort has a Big O notation of O(1) for space, and O(n^2) for space. N refers to the size of the array. For example, array [5, 2, 1, -2, 8] has 5 elements, meaning equals 5.

Concept

Selection sort uses a nested for loop. It essentially works through the array from the beginning, with all the elements behind the current index being sorted, and the elements in front of the current element to be unsorted. Eventually, it will reach the end and the entire array will be solved.

This can be done using a nested for loop. The outer for loop will start at the first index in the array. Let’s call this value i. The loop will contain a variable known as min\_index, which will start by holding the value of i.

Next, we will enter the nested for loop, which starts at the index after i. We will call this value j. We will compare the value at index j with that of index i and if it is smaller than the value at index i, then the min\_index variable will be set to j.

After iterating through the entire array in the nested for loop, we return to our outer for loop, and swap the values at index i and the min\_index. This will ensure that the first element of the array now has the smallest value in the array. The next iteration will move onto the second element, and so on. Once the outer array has iterated through every element of the array, it will be sorted.

Additionally, the outer for loop does not have to iterate to the final element of the array. This is because, all the elements before it will be ordered from least to greatest, and since the last element is checked every time in the inner loop, the final element will always end up being the largest value.

Java Implementation

This implementation of the algorithm is an ascending order sort on a sample integer of arrays.

public class SelectionSortImplementation {

public static void selectionSort(int[] arr) {

for(int i=0; i<arr.length-1; i++) {

int minIndex = i;

for(int j=i+1; j<arr.length; j++) {

if(arr[j] < arr[minIndex])

minIndex = j;

}

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

printArray(arr);

}

}

public static void main(String[] args) {

int[] array1 = {31, 15, -12, 4, 10};

selectionSort(array1);

}

}