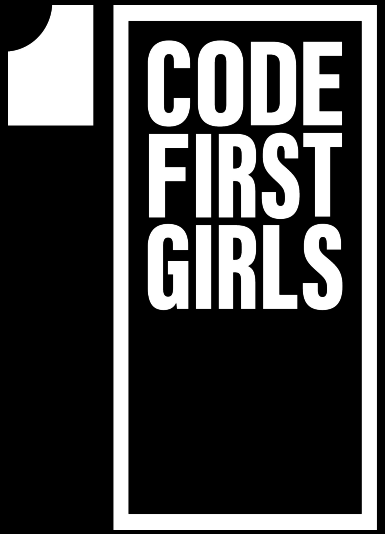
SEARCH & SORT ALGORITHMS

LESSON 15



NANODEGREE → ENGINEERING MODULE

AGENDA

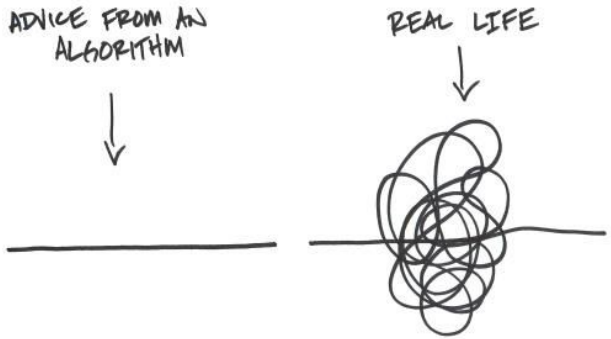
01 Search and Sort introduction

02 Search and Sort algorithms review

03 Search and Sort Practice

SEARCH & SORT INTRODUCTION

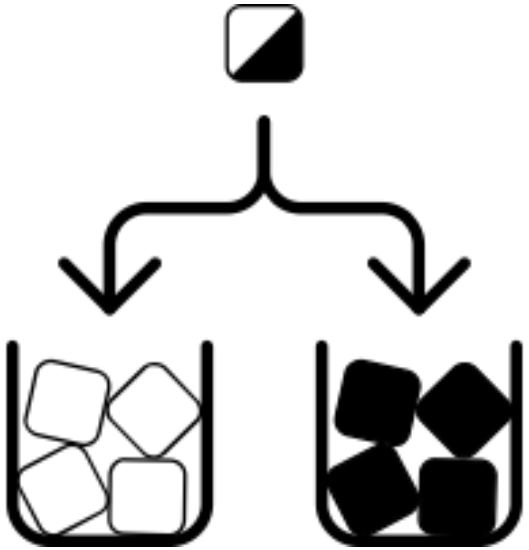
- The **searching and sorting algorithms are key algorithms** that you will study in computer science.
- **Most computer programs involve some searching and sorting** features so these key algorithms are often used when coding various computer programs.
- Searching and sorting algorithms are also useful to develop your **algorithmic thinking skills** and your ability **to compare and evaluate the effectiveness of an algorithm** to perform a specific task.



SORTING

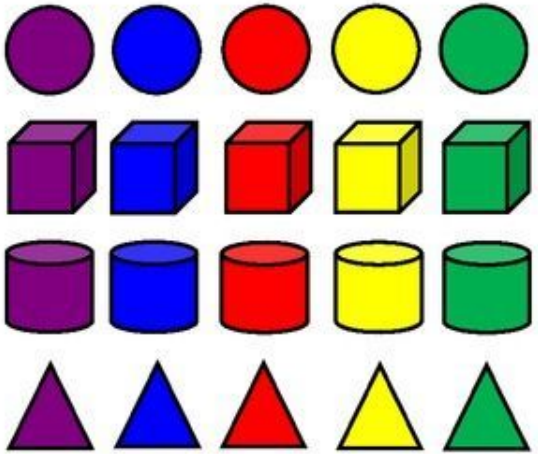
- **Sorting** is one of the key algorithms in computer science. There are many different sorting implementations and applications that you can use to make your code more **efficient and effective**.

- **Sorting** helps to make **Searching** more **efficient, quicker** and also helps to minimize errors.



SORTING ADVANTAGES

- **Searching:** Searching for an item on a list works much faster if the list is sorted.
- **Selection:** Selecting items from a list based on their relationship to the rest of the items is easier with sorted data.
- **Duplicates**: Finding duplicate values on a list can be done very quickly when the list is sorted.
- **Distribution**: Analyzing the frequency distribution of items on a list is very fast if the list is sorted.



- Let’s review examples of various Sort & Search Techniques
- We will review LOADS of examples together
- We will practice the **KEY** algorithms **only** to stay focused

LINEAR SEARCH

Linear search is a method of finding elements within a list. It is also called a **sequential search.** It is the simplest searching algorithm because it searches the desired element in a sequential manner.

Algorithm

- Start from the leftmost element of given array and one by one compare element x with each element of array
- If x matches with any of the element, return the index value.
- If x doesn't match with any of elements in array, return -1 or element not found.

BUBBLE SORT

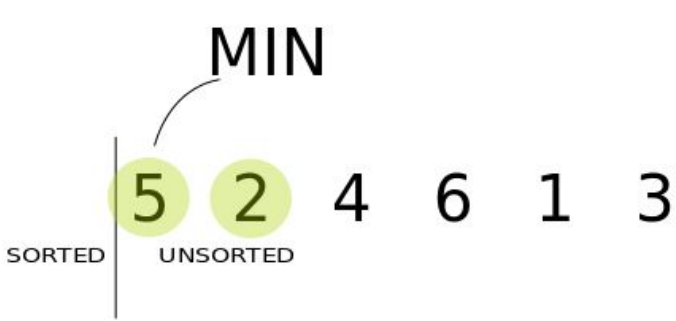
Bubble Sort is also a simple sorting algorithm that works by **repeatedly swapping the adjacent elements** if they are in wrong order. Because Bubble sort repeatedly passes through the unsorted part of the list, it has a worst case complexity of $O(n^2)$.

Algorithm

- Start by stepping through the list and compare adjacent pairs of elements.
- Swapped the elements if they are in the wrong order.
- Repeat passing through the unsorted portion of the list until the list is sorted.

SELECTION SORT

Selection Sort **sorts** an array **by finding the minimum value of the unsorted part and then swapping it** with the first unsorted element. It is an in-place algorithm, meaning you won't need to allocate additional lists. While slow, it is still used as the main sorting algorithm in systems where memory is limited.



Algorithm

- Divide input array into two parts: items already sorted and items remaining to be sorted (they make up the rest of the list).
- First find the smallest element in the unsorted sublist and place it at the end of the sorted sublist.
- Continuously grab the smallest unsorted element and place it in sorted order in the sorted sublist.
- Continue this process iteratively until the list is fully sorted.

INSERTION SORT

Insertion sort is a simple sorting algorithm that works **similar to the way you sort playing cards in your hands**. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

6 5 3 1 8 7 2 4

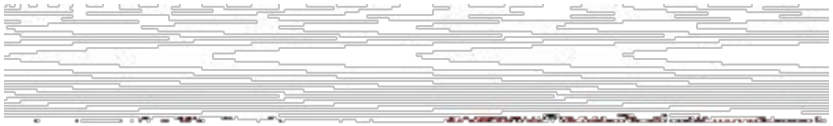
Algorithm

(sort array in asc order)

- Iterate from arr[1] to arr[n] over the array.
- Compare the current element (key) to its predecessor.
- If the key element is smaller than its predecessor, compare it to the elements before.
- Move the greater elements one position up to make space for the swapped element.

MERGE SORT

Merge sort is one of the most prominent **divide-and-conquer** sorting algorithms in the modern era. It can be used to sort the values in any traversable data structure such as a list.



Algorithm

(divide)

- Continuously divide the unsorted list until you have N sublists, where each sublist has 1 element that is “unsorted” and N is the number of elements in the original array.

(conquer)

- Repeatedly merge i.e conquer the sublists together 2 at a time to produce new sorted sublists until all elements have been fully merged into a single sorted array.

QUICK SORT

Quicksort is a popular sorting algorithm and is often used, right alongside Merge Sort. It's a good example of an efficient sorting algorithm, with an average complexity of O(nlogn). Part of its popularity also derives from the ease of implementation.

6 5 3 1 8 7 2 4

Algorithm

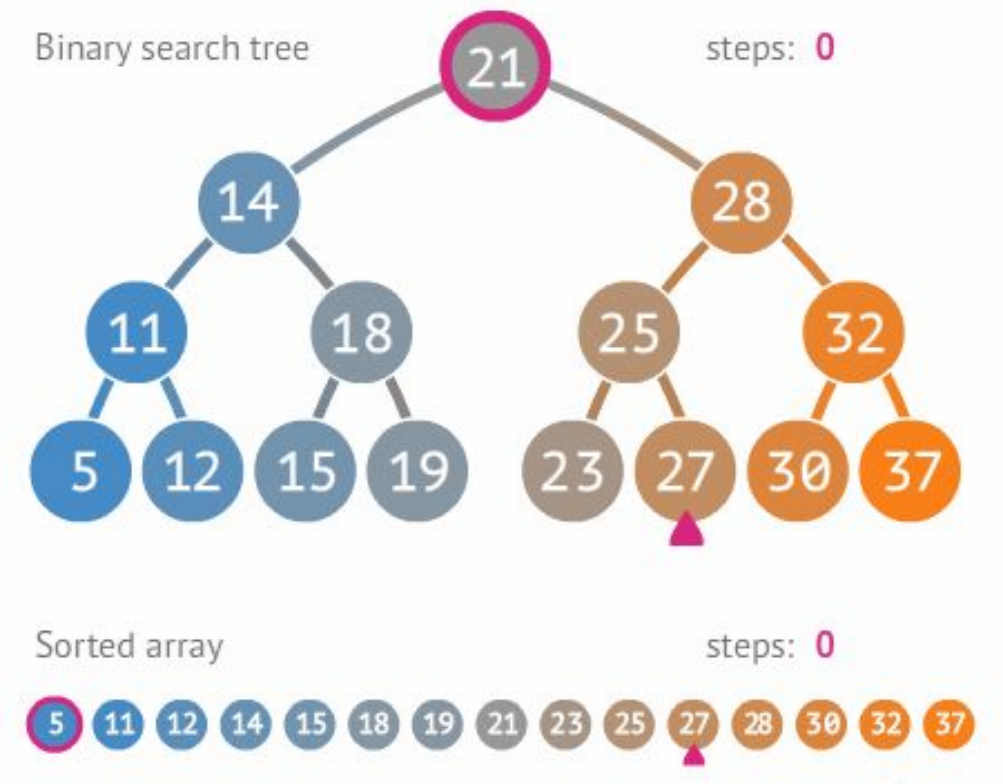
- Divide the collection in two (roughly) equal parts by taking a pseudo-random element and using it as a pivot.
- Elements smaller than the pivot get moved to the left of the pivot, and elements larger than the pivot to the right of it.
- This process is repeated for the collection to the left of the pivot, as well as for the array of elements to the right of the pivot until the whole array is sorted.

BINARY SEARCH

- **Binary search** is an algorithm that finds the position of an element in an ordered array.
- Binary searches repeatedly divide a list into two halves. Then, a search compares if a value is higher or lower than the middle value in the list.

BINARY STRUCTURE

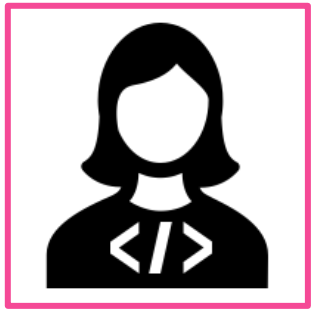
BINARY SEARCH TREE VS ARRAY



MORE SORT TYPES

- **Shell Sort**
- **Heap Sort**
- **Counting Sort**
- **Bucket Sort**
- **Radix Sort**
- **Cubesort**
- and many more...

Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	Ω(N)	Θ(N^2)	O(N^2)	O(1)
Selection Sort	Ω(N^2)	Θ(N^2)	O(N^2)	O(1)
Insertion Sort	Ω(N)	Θ(N^2)	O(N^2)	O(1)
Quick Sort	Ω(N log N)	Θ(N log N)	O(N^2)	O(N)
Merge Sort	Ω(N log N)	Θ(N log N)	O(N log N)	O(N)



DEMO &
EXERCISES

ALGORITHMS
EXERCISES & PRACTICE

THANK YOU!