Sorting and Searching Algorithms in JavaScript

Sorting and searching are two of the most fundamental operations in computer science. Sorting helps organize data so it can be easily searched, while searching finds specific data within a collection. This document explains common algorithms for both, using JavaScript examples.

1. Sorting Algorithms

Bubble Sort

Bubble Sort is a simple algorithm that repeatedly compares adjacent elements and swaps them if they are in the wrong order. This process continues until no swaps are needed, meaning the array is sorted.

```
function bubbleSort(arr) {
  let n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    for (let j = 0; j < n - 1 - i; j++) {
      if (arr[j] > arr[j + 1]) {
        let temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
  return arr;
}

// Example
console.log(bubbleSort([5, 2, 9, 1, 5, 6]));
```

Selection Sort

Selection Sort divides the array into two parts — a sorted and an unsorted part. It repeatedly finds the smallest (or largest) element from the unsorted section and places it at the beginning.

```
function selectionSort(arr) {
  let n = arr.length;
  for (let i = 0; i < n; i++) {
    let min = i;
    for (let j = i + 1; j < n; j++) {
      if (arr[j] < arr[min]) {
        min = j;
      }
    }
    let temp = arr[i];
    arr[i] = arr[min];
    arr[min] = temp;
  }
  return arr;
}

// Example
console.log(selectionSort([64, 25, 12, 22, 11]));
```

Quick Sort

Quick Sort is a divide-and-conquer algorithm. It selects a 'pivot' element and partitions the array into two sub-arrays: one with elements smaller than the pivot and the other with elements larger. It then recursively sorts the sub-arrays.

```
function quickSort(arr) {
  if (arr.length <= 1) {
    return arr;
  }
```

```
  let pivot = arr[arr.length - 1];
  let left = [];
  let right = [];
  for (let i = 0; i < arr.length - 1; i++) {
    if (arr[i] < pivot) {
      left.push(arr[i]);
    } else {
      right.push(arr[i]);
    }
  }
  return [...quickSort(left), pivot, ...quickSort(right)];
}

// Example
console.log(quickSort([10, 7, 8, 9, 1, 5]));
```

2. Searching Algorithms

Linear Search

Linear Search checks each element in a list one by one until it finds the target or reaches the end. It's simple but inefficient for large lists.

```
function linearSearch(arr, target) {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === target) {
      return i; // Found
    }
  }
  return -1; // Not found
}

// Example
console.log(linearSearch([3, 5, 7, 9, 1], 7));
```

Binary Search

Binary Search is an efficient method for finding a target value in a sorted array. It repeatedly divides the search space in half. If the target is smaller than the middle element, search the left half; otherwise, search the right half.

```
function binarySearch(arr, target) {
  let left = 0;
  let right = arr.length - 1;
  while (left <= right) {
    let mid = Math.floor((left + right) / 2);
    if (arr[mid] === target) return mid;
    if (arr[mid] < target) left = mid + 1;
    else right = mid - 1;
  }
  return -1;
}

// Example
console.log(binarySearch([1, 2, 3, 4, 5, 6, 7, 8, 9], 6));
```

Summary

- **Bubble Sort:** Simple but slow for large datasets. - **Selection Sort:** Easy to understand, but not efficient. - **Quick Sort:** Very efficient for large datasets. - **Linear Search:** Works for any array, but slow. - **Binary Search:** Very fast, but requires a sorted array.