

Searching and Sorting Algorithms

You said:



Give me all searching and sorting algorithm with graphic example explaining them in simple and then write simple java code
And give their time and space complexity as well
Discuss about all the most asked searching and sorting algorithms

By graphic example means how by directly doing first then write code

ChatGPT said:



Certainly! Let's discuss some commonly asked searching and sorting algorithms. I'll explain each one with a simple example and provide Java code along with their time and space complexities.

Searching Algorithms

1. Linear Search

Explanation: Linear search scans each element of the array sequentially until the desired element is found or the array is exhausted.

Example: Let's search for the number `5` in the array `[3, 6, 5, 2, 9]`.

1. Check `3` - not a match.
2. Check `6` - not a match.
3. Check `5` - match found.

Time Complexity: $O(n)$

Space Complexity: $O(1)$

Java Code:

```
java
```

```
public class LinearSearch {
    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i; // Target found
            }
        }
        return -1; // Target not found
    }

    public static void main(String[] args) {
        int[] arr = {3, 6, 5, 2, 9};
        int target = 5;
        int result = linearSearch(arr, target);
    }
}
```

```
        System.out.println(result); // Output: 2
    }
}
```

2. Binary Search

Explanation: Binary search works on sorted arrays. It repeatedly divides the search interval in half.

Example: For the array `[2, 3, 5, 6, 9]`, to find `5`:

1. Middle element is `5`, which matches the target.

Time Complexity: $O(\log n)$

Space Complexity: $O(1)$ for iterative, $O(\log n)$ for recursive

Java Code:

```
java

public class BinarySearch {
    public static int binarySearch(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {
                return mid; // Target found
            } else if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return -1; // Target not found
    }

    public static void main(String[] args) {
        int[] arr = {2, 3, 5, 6, 9};
        int target = 5;
        int result = binarySearch(arr, target);
        System.out.println(result); // Output: 2
    }
}
```

Sorting Algorithms

1. Bubble Sort

Explanation: Bubble sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.

Example: For array `[5, 3, 8, 4, 2]`:

1. Compare `5` and `3`, swap -> `[3, 5, 8, 4, 2]`
2. Compare `5` and `8`, no swap.
3. Compare `8` and `4`, swap -> `[3, 5, 4, 8, 2]`
4. Compare `8` and `2`, swap -> `[3, 5, 4, 2, 8]`
5. Repeat until sorted.

Time Complexity: $O(n^2)$

Space Complexity: $O(1)$

Java Code:

java

```
public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {5, 3, 8, 4, 2};
        bubbleSort(arr);
        for (int i : arr) {
            System.out.print(i + " "); // Output: 2 3 4 5 8
        }
    }
}
```

2. Merge Sort

Explanation: Merge sort divides the array into halves, recursively sorts each half, and then merges the sorted halves.

Example: For array `[5, 3, 8, 4, 2]`:

1. Divide into `[5, 3]` and `[8, 4, 2]`
2. Sort `[5, 3]` to `[3, 5]`
3. Sort `[8, 4, 2]` to `[2, 4, 8]`
4. Merge `[3, 5]` and `[2, 4, 8]` to `[2, 3, 4, 5, 8]`

Time Complexity: $O(n \log n)$

Space Complexity: $O(n)$

Java Code:

java

```
public class MergeSort {
    public static void mergeSort(int[] arr) {
        if (arr.length < 2) return;

        int mid = arr.length / 2;
        int[] left = new int[mid];
        int[] right = new int[arr.length - mid];

        System.arraycopy(arr, 0, left, 0, mid);
        System.arraycopy(arr, mid, right, 0, arr.length - mid);

        mergeSort(left);
        mergeSort(right);
        merge(arr, left, right);
    }
}
```

```
private static void merge(int[] arr, int[] left, int[] right) {
    int i = 0, j = 0, k = 0;
    while (i < left.length && j < right.length) {
        if (left[i] <= right[j]) {
            arr[k++] = left[i++];
        } else {
            arr[k++] = right[j++];
        }
    }
    while (i < left.length) {
        arr[k++] = left[i++];
    }
    while (j < right.length) {
        arr[k++] = right[j++];
    }
}

public static void main(String[] args) {
    int[] arr = {5, 3, 8, 4, 2};
    mergeSort(arr);
    for (int i : arr) {
        System.out.print(i + " "); // Output: 2 3 4 5 8
    }
}
```

3. Quick Sort

Explanation: Quick sort picks an element as a pivot and partitions the array around the pivot, recursively sorting the partitions.

Example: For array `[5, 3, 8, 4, 2]`, pivot is `4`:

1. Partition into `[3, 2]` and `[5, 8]`
2. Sort `[3, 2]` and `[5, 8]` around pivots
3. Merge into `[2, 3, 4, 5, 8]`

Time Complexity: $O(n \log n)$ average, $O(n^2)$ worst-case

Space Complexity: $O(\log n)$

Java Code:

java

```
public class QuickSort {
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    private static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
    }
}
```

```

        arr[high] = temp;
        return i + 1;
    }

    public static void main(String[] args) {
        int[] arr = {5, 3, 8, 4, 2};
        quickSort(arr, 0, arr.length - 1);
        for (int i : arr) {
            System.out.print(i + " "); // Output: 2 3 4 5 8
        }
    }
}

```

These are some of the most commonly asked searching and sorting algorithms. Each has its own strengths and use cases depending on the size and nature of the data.

Is this conversation helpful so far?

ChatGPT can make mistakes. Check important info.