**Link source code:**

# Bài 1:

Merge hai mảng A và B đã sắp xếp tăng dần thành một mảng duy nhất vẫn được sắp xếp.

VD:

Input: A = [1, 3, 5], B = [2, 4, 6]

Output: [1, 2, 3, 4, 5, 6]

* ***Nâng cao:*** Không sử dụng mảng trung gian, hợp nhất các phần tử của B vào A, giả sử mảng A có đủ không gian trống.

***Đơn giản:***

private static int[] mergeSortedArrayBasic(int[] A, int[] B) {

        int aLength = A.length;

        int bLength = B.length;

        int[] result = new int[aLength + bLength];

        // i: A index, j: B index, k: result index

        int i = 0, j = 0, k = 0;

        // Compare A and B then orderly put into result

        while (i < aLength && j < bLength) {

            if (A[i] <= B[j]) {

                result[k++] = A[i++];

            } else {

                result[k++] = B[j++];

            }

        }

        // Append the rest of A elements

        while (i < aLength) {

            result[k++] = A[i++];

        }

        // Append the rest of B elements

        while (j < bLength) {

            result[k++] = B[j++];

        }

        return result;

    }

Giải thích:

* Khởi tạo các chỉ số i, j cho A và B, và k cho result.
* So sánh từng phần tử của A và B, đặt phần tử nhỏ hơn hoặc bằng vào result, tăng index tương ứng.
* Sau khi một mảng hết, thêm các phần tử còn lại của mảng còn lại vào result.

Complexity:

**Time:** (m + n)

**Space:** (m + n)

Với m là độ dài của mảng A và n là độ dài của mảng B.

***Nâng cao:***

private static int[] mergeSortedArrayAdvanced(int[] A, int[] B) {

        // Figure out how many existing elements in A

        int i = 0;

        while (A[i + 1] >= A[i]) {

            i++;

        }

        int j = B.length - 1;

        int k = A.length - 1;

        // Loop backward to find the large element from both array

        while (i >= 0 && j >= 0) {

            if (A[i] >= B[j]) {

                A[k--] = A[i--];

            } else {

                A[k--] = B[j--];

            }

        }

        // Insert the rest of B elements (if any)

        while (j >= 0) {

            A[k--] = B[j--];

        }

        return A;

    }

Complexity:

**Time:** O(m + n)

**Space:** O(1), không cần mảng mới

Với m là độ dài của mảng A và n là độ dài của mảng B.

# Bài 2:

Cho 1 mảng prices, prices[i] là giá để mua vàng vào ngày thứ i. Mục tiêu đặt ra là tối đa hóa lợi nhuận có được bằng việc mua đáy bán đỉnh. Xác định lợi nhuận tối đa có được bằng việc chọn một ngày mua và một ngày bán trong tương lai.

VD:

Input: prices = [7,1,5,3,6,4]

Output: 5 (Mua ngày 2 (price = 1) và bán ngày 5 (price = 6), profit= 6 - 1 = 5)

* ***Nâng cao:*** *Có thể chọn nhiều lần mua và bán. Tuy nhiên tại một thời điểm chỉ được phép giữ* ***1 đơn vị vàng*** *(nghĩa là phải bán trước khi mua lần tiếp theo).*

***Đơn giản:***

private static int findMaxProfit(int[] prices) {

        // Check null

        if (prices == null || prices.length < 2) {

            return 0;

        }

        // Initialize value

        int min\_price = Integer.MAX\_VALUE;

        int max\_profit = 0;

        // Loop for identifying cheapest price and maximum profit

        for (int price : prices) {

            if (price < min\_price) {

                min\_price = price;

            } else {

                max\_profit = Math.max(max\_profit, price - min\_price);

            }

        }

        return max\_profit;

    }

Giải thích:

* Khởi tạo các chỉ số i, j cho A và B, và k cho result.
* So sánh từng phần tử của A và B, đặt phần tử nhỏ hơn hoặc bằng vào result, tăng index tương ứng.
* Sau khi một mảng hết, thêm các phần tử còn lại của mảng còn lại vào result.

Complexity:

**Time:** O(n)

**Space:** O(1)

Với n là số phần tử trong mảng đầu vào

***Nâng cao: Tổng lợi nhuận từ nhiều giao dịch bằng cách cộng dồn tất cả các chênh lệch giá tăng liên tiếp.***

private static int findMaxProfitMultiple(int[] prices) {

        int totalProfit = 0;

        for (int i = 0; i < prices.length; i++) {

            if (i != 0) {

                if (prices[i] > prices[i - 1]) {

                    totalProfit += prices[i] - prices[i - 1];

                }

            }

        }

        return totalProfit;

}

Complexity:

**Time:** O(n)

**Space:** O(1), không cần mảng mới

Với n là số phần tử có trong mảng truyền vào.

# Bài 3:

Tìm 2 phần tử trong một mảng tăng dần có tổng bằng k.

VD:

Input: arr = [1, 2, 3, 4, 6], k = 6

Output: [2, 4]

* ***Nâng cao: Tìm 3 phần tử trong mảng có tổng bằng k.***

***Đơn giản:***

public static int[] twoSum(int[] arr, int k) {

        int left = 0;

        int right = arr.length - 1;

        while (left < right) {

            int sum = arr[left] + arr[right];

            if (sum == k) {

                return new int[] { arr[left], arr[right] };

            } else if (sum < k) {

                left++;

            } else {

                right--;

            }

        }

        return new int[0]; // Not found

}

Giải thích:

* Khởi tạo hai con trỏ: left bắt đầu từ đầu mảng và right bắt đầu từ cuối mảng.
* Lặp lại trong khi left nhỏ hơn right:
* Tính tổng của các phần tử tại vị trí left và right.
* Nếu tổng bằng k, trả về cặp số đó.
* Nếu tổng nhỏ hơn k, di chuyển con trỏ left sang phải để tăng tổng.
* Nếu tổng lớn hơn k, di chuyển con trỏ right sang trái để giảm tổng.
* Nếu không tìm thấy cặp số phù hợp, trả về mảng rỗng.

Complexity:

**Time:** O(n)

**Space:** O(1)

Với n là số phần tử trong mảng đầu vào

***Nâng cao:***

private static int[] threeSum(int[] arr, int k) {

        int n = arr.length;

        if (n < 3)

            return new int[0];

        for (int i = 0; i < n - 2; i++) {

            int left = i + 1;

            int right = n - 1;

            while (left < right) {

                int currentSum = arr[i] + arr[left] + arr[right];

                if (currentSum == k) {

                    return new int[] { arr[i], arr[left], arr[right] };

                } else if (currentSum < k) {

                    left++;

                } else {

                    right--;

                }

            }

        }

        return new int[0]; // Not found

}

Giải thích:

* Cố định phần tử thứ nhất arr[i].
* Sử dụng hai con trỏ left và right để tìm hai phần tử còn lại sao cho tổng bằng k - arr[i].
* Duyệt tất cả các phần tử arr[i] từ đầu đến phần tử thứ ba từ cuối để đảm bảo lúc nào cũng sẽ còn bộ ba để so sánh

Complexity:

**Time:** O(n)

**Space:** O(1), không cần mảng mới

Với n là số phần tử có trong mảng truyền vào.

# Bài 4:

Cho một chuỗi s. Kiểm tra xem chuỗi đó có phải là chuỗi đối xứng (palindrome) hay không?

VD1:

Input: s = "radar"

Output: True

VD2:

Input: s = "hello"

Output: False

* ***Nâng cao: Đếm số chuỗi con đối xứng có trong chuỗi.***

***Đơn giản:***

private static boolean isPalindrome(String s) {

        if (s == null || s.isEmpty()) {

            return true;

        }

        int left = 0;

        int right = s.length() - 1;

        while (left < right) {

            if (s.charAt(left) != s.charAt(right)) {

                return false;

            }

            left++;

            right--;

        }

        return true;

}

Giải thích:

* Kiểm tra chuỗi đối xứng: Sử dụng hai con trỏ left và right để so sánh các ký tự từ hai đầu chuỗi tiến vào trong. Nếu tất cả các cặp ký tự đều giống nhau, chuỗi là đối xứng.

Complexity:

**Time:** O(n)

**Space:** O(1)

Với n là số phần tử trong mảng đầu vào

***Nâng cao:***

private static int countPalindromes(String s) {

        if (s == null || s.isEmpty()) {

            return 0;

        }

        int count = 0;

        for (int i = 0; i < s.length(); i++) {

            // Count substring palindrome (odd)

            count += expandAroundCenter(s, i, i);

            // Count substring palindrome (even)

            count += expandAroundCenter(s, i, i + 1);

        }

        return count;

    }

    private static int expandAroundCenter(String s, int left, int right) {

        int count = 0;

        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {

            count++;

            left--;

            right++;

        }

        return count;

}

Giải thích:

* expandAroundCenter là một hàm hỗ trợ để mở rộng từ một điểm trung tâm ra ngoài, kiểm tra cả chuỗi đối xứng độ dài lẻ (một ký tự làm trung tâm) và chẵn (hai ký tự làm trung tâm).
* Mỗi lần mở rộng thành công từ một vị trí, chúng ta tăng biến đếm count.

Complexity:

**Time:** O(n^2)

**Space:** O(1), không cần mảng mới

Với n là số phần tử có trong mảng truyền vào.

# Bài 5:

Trong một lớp mẫu giáo, có **n bé ngoan** đang tham gia nhận thưởng vì thành tích trong tuần. Mỗi bé được giáo viên đánh giá bằng một **phiếu bé ngoan**, ratings[i] là số phiếu bé ngoan mà bé thứ i nhận được. Để khen thưởng, giáo viên yêu cầu các bé xếp thành hàng và sẽ phát kẹo cho các bé theo **hai quy tắc quan trọng**:

- Mỗi đứa trẻ nhận ít nhất một viên kẹo.

- Những bé có nhiều phiếu bé ngoan hơn bạn bên cạnh phải nhận được nhiều kẹo hơn bạn đó.

Hãy tính xem cần ít nhất bao nhiêu viên kẹo để giáo viên hoàn thành nhiệm vụ này.

VD:

Input: ratings = [1, 0, 2]

Output: 5

**Giải thích:**

* Bé thứ 1 (phiếu = 1): nhận 2 viên kẹo (vì nhiều phiếu hơn bé bên cạnh).
* Bé thứ 2 (phiếu = 0): nhận 1 viên kẹo (ít phiếu nhất).
* Bé thứ 3 (phiếu = 2): nhận 2 viên kẹo (vì nhiều phiếu hơn bé bên cạnh).  
  Tổng số kẹo là 1+2+2=5

public static int minCandies(int[] ratings) {

        int n = ratings.length;

        int[] candies = new int[n];

        // Mỗi bé nhận ít nhất một viên kẹo

        for (int i = 0; i < n; i++) {

            candies[i] = 1;

        }

        // Duyệt từ trái sang phải

        // Duyệt từ phải sang trái (làm tương tự)

        for (int i = 0; i < n; i++) {

            if (i == 0) {

                if (ratings[i] > ratings[i + 1]) {

                    candies[i]++;

                }

            } else {

                if (ratings[i] > ratings[i - 1]) {

                    candies[i]++;

                }

            }

        }

        // Tính tổng số kẹo cần phát

        int totalCandies = 0;

        for (int candy : candies) {

            totalCandies += candy;

        }

        return totalCandies;

}

Giải thích:

* Khởi tạo số kẹo: Đầu tiên, chúng ta khởi tạo mảng candies với tất cả các phần tử đều là 1, nghĩa là mỗi bé nhận ít nhất một viên kẹo.
* Duyệt từ trái sang phải: Nếu bé hiện tại có nhiều phiếu bé ngoan hơn bé đứng trước, bé hiện tại sẽ nhận số kẹo nhiều hơn bé đứng trước một viên.
* Tính tổng số kẹo: Cuối cùng, tính tổng số kẹo cần phát bằng cách cộng tất cả các phần tử trong mảng candies.

Complexity:

**Time:** O(n)

**Space:** O(n)

Với n là số phần tử trong mảng đầu vào

Thông tin liên hệ:

* **Email:** [**hung.NHK@kdc.vn**](mailto:hung.NHK@kdc.vn)
* **Tell: 0933.20.54.20 (Tell, Zalo)**