Cấu trúc dữ liệu

Thực hành 3 - Các thuật toán sắp xếp hiệu quả

Shell sort và Quick sort

Bài 1: Tìm hiểu thêm về cách chọn các chuỗi giá trị cho thuật toán shell sort như: Knuth's increments, Sedgewick's increments, Hibbard's increments, Papernov & Stasevich increment, Pratt.

Knuth's increments

```
void knuth_increments(int n, int increments[]) {
    int h = 1, i = 0;

    // Generate Knuth's increments sequence
    while (h < n) {
        increments[i++] = h;
        h = 3*h + 1;
    }

    // Reverse the sequence
    for (int j = i - 1; j >= 0; j--) {
        increments[i++] = increments[j];
    }
}
```

Sedgewick's increments

```
void sedgewick_increments(int n, int increments[]) {
    int h = 1, i = 0;
    int power2 = 1, power4 = 1;
    int k = 0;

    // Generate Sedgewick's increments sequence
    while (h < n) {
        if (k % 2 == 0) {
            increments[i++] = 9 * power2 - 9 * power4 + 1;
            power4 *= 4;
        } else {
            increments[i++] = 8 * power2 - 6 * power4 + 1;
            power2 *= 2;
            power4 *= 4;
        }
        k++;
        h = increments[i - 1];
    }

    // Reverse the sequence
    for (int j = i - 1; j >= 0; j--) {
```

```
        increments[i++] = increments[j];
    }
}
```

Hibbard's increments

```
void hibbard_increments(int n, int increments[]) {
    int h = 1, i = 0;
    int k = 1;

    // Generate Hibbard's increments sequence
    while (h < n) {
        increments[i++] = h;
        h = (1 << k) - 1;
        k++;
    }

    // Reverse the sequence
    for (int j = i - 2; j >= 0; j--) {
        increments[i++] = increments[j];
    }
}
```

Papernov & Stasevich increment

```
#include <cmath>
void papernov_stasevich_increments(int n, int increments[]) {
    int i = 0;

    // Generate Papernov & Stasevich's increments sequence
    for (int k = log2(n + 1) - 1; k >= 0; k--) {
        increments[i++] = (pow(2, k+1) - 2) * pow(2, (k/2)) + 1;
    }

    // Reverse the sequence
    for (int j = i - 2; j >= 0; j--) {
        increments[i++] = increments[j];
    }
}
```

Pratt.

```
#include <cmath>

void pratt_increments(int n, int increments[]) {
    int i = 0;
    increments[i++] = 1; // Start with 1

    // Generate Pratt's increments sequence
    for (int p2 = 0; pow(2, p2) <= n; p2++) {
        for (int p3 = 0; pow(2, p2) * pow(3, p3) <= n; p3++) {
            increments[i++] = pow(2, p2) * pow(3, p3);
```

```
        }
    }

    // Reverse the sequence
    for (int j = i - 2, k = 0; j >= 0; j--, k++) {
        increments[i + k] = increments[j];
    }
}
```

Bài 2: Đề xuất cách chọn giá trị pivot để hạn chế trường hợp xấu nhất xảy ra khi triển khai thuật toán quick sort

1. Chọn ở giữa mảng
2. Chọn ở đầu/cuối mảng
3. Chọn ngẫu nhiên
4. Chọn phần tử trung vị

Bài 4: Viết chương trình thực hiện merge sort, và count sort sử dụng đệ quy và không sử dụng đệ quy.

Merge sort sử dụng đệ quy

```
#include <iostream>
using namespace std;

// Merge two sorted subarrays into a single sorted array
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    // Copy the left subarray into a temporary array
    for (int i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }

    // Copy the right subarray into a temporary array
    for (int j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }

    int i = 0, j = 0, k = l;

    // Compare and merge the elements from the left and right subarrays
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
```

```
        }
        k++;
    }

    // Copy any remaining elements from the left subarray
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy any remaining elements from the right subarray
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Sort an array using merge sort
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        // Recursively sort the left and right subarrays
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted subarrays
        merge(arr, l, m, r);
    }
}

int main() {
    int arr[] = {38, 27, 43, 3, 9, 82, 10};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    // Sort the array using merge sort
    mergeSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Merge sort không sử dụng đệ quy

```
#include <iostream>
using namespace std;

void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    // Copy the left subarray into a temporary array
    for (int i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }

    // Copy the right subarray into a temporary array
    for (int j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }

    int i = 0, j = 0, k = l;

    // Compare and merge the elements from the left and right subarrays
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy any remaining elements from the left subarray
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy any remaining elements from the right subarray
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int n) {
    // Outer loop iterates over different subarray sizes
    for (int curr_size = 1; curr_size <= n-1; curr_size = 2*curr_size) {
        // Inner loop merges adjacent subarrays of the current size
        for (int left_start = 0; left_start < n-1; left_start += 2*curr_size)
{
            // Compute indices of the middle and right ends of the current
subarray
```

```
            int mid = min(left_start + curr_size - 1, n-1);
            int right_end = min(left_start + 2*curr_size - 1, n-1);

            // Merge the current subarray
            merge(arr, left_start, mid, right_end);
        }
    }
}


int main() {
    int arr[] = {38, 27, 43, 3, 9, 82, 10};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    // Sort the array using merge sort
    mergeSort(arr, n);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```