THUẬT TOÁN SẮP XẾP

Thuật toán sắp xếp:

Sắp xếp là một thuật toán quan trọng, được sử dụng cực kì nhiều trong các ứng dụng thực tế. Các bạn có thể không biết các thuật toán như quy hoạch động, chia và trị nhưng bắt buộc phải nắm được sắp xếp và tìm kiếm.

Mục lục:

1. Thuật toán sắp xếp chọn (Selection sort):

Code:

```
public static void selectionSort(int a[], int n){
    for(int i = 0; i < n; i++){
        int min_pos = i;
        for (int j = i + 1; j < n; j++)
            if (a[j] < a[min_pos])
                min_pos = j;
        int tmp = a[min_pos];
        a[min_pos] = a[i];
        a[i] = tmp;
    }
}
```

Độ phức tạp: **O(N^2)**

2. Thuật toán sắp xếp nổi bọt (Bubble sort):

Code:

```
public static void bubbleSort(int a[], int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n - i - 1; j++){
            if (a[j] > a[j + 1]){
                int tmp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = tmp;
            }
        }
    }
}
```

Độ phức tạp: **O(N^2)**

3. Thuật toán sắp xếp chèn (Insertsion sort):

Code:

```
public static void insertionSort(int a[], int n){
    for(int i = 1; i < n; i++){
        int pos = i - 1, x = a[i];
        while(pos >= 0 && a[pos] > x){
            a[pos + 1] = a[pos];
            --pos;
        }
        a[pos + 1] = x;
    }
}
```

Độ phức tạp: **O(N^2)**

4. Thuật toán sắp xếp đếm phân phối (Counting sort):

Code:

Điều kiện áp dụng: Có thể khai báo được mảng đếm có số lượng phần tử lớn hơn giá trị lớn nhất của phần tử trong mảng

```
public static int[] dem = new int[1000001]; // 0 <= a[i] <= 10^6
void countingSort(int a[], int n){
    int K = -1e9;
    for(int i = 0; i < n; i++){
        dem[a[i]]++;
        K = Math.max(K, a[i]);
    }
    for(int i = 0; i <= K; i++){
        if(dem[i]){
            for(int j = 0; j < dem[i]; j++)
                System.out.print(i + " ");
        }
    }
}
```

Độ phức tạp: **O(N+K)**

5. Thuật toán sắp xếp trộn (Merge sort):

Thao tác trộn:

```
public static void merge(int a[], int l, int m, int r){
    int n1 = m - l + 1, n2 = r - m;
    int x[n1], y[n2];
    for(int j = l; j <= m; j++)
        x[j - l] = a[j];
    for(int j = m + 1; j <= r; j++)
        y[j - m - 1] = a[j];
    int i = 0, j = 0, cnt = l;
    while(i < n1 && j < n2){
        if(x[i] <= y[j])
            a[cnt++] = x[i++];
        else
            a[cnt++] = y[j++];
    }
    while(i < n1) a[cnt++] = x[i++];
    while(j < n2) a[cnt++] = y[j++];
}
```

5. Thuật toán sắp xếp trộn (Merge sort):

Hàm merge sort và main:

```
public static void mergeSort(int a[], int l, int r){
    if(l < r){
        int m = (l + r) / 2;
        mergeSort(a, l, m);
        mergeSort(a, m + 1, r);
        merge(a, l, m , r);
    }
}
public static void main(String[] args) {
        int a[] = {3, 1, 0, 4, 2, 6, 5};
        mergeSort(a, 0, 6);
        for(int x : a){
            System.out.print(x + " ");
        }
}
```

Độ phức tạp: **O(NlogN)**

6. Thuật toán quick sort:

Thao tác phân hoạch bằng Lomuto partition

```
public static int partition(int a[], int l, int r){
    int pivot = a[r];
    int i = l - 1;
    for(int j = l; j < r; j++){
        if(a[j] <= pivot){
            ++i;
            int tmp = a[i];
            a[i] = a[j];
            a[j] = tmp;
        }
    }
    ++i;
    int tmp = a[i];
    a[i] = a[r];
    a[r] = tmp;
    return i;
}
```

6. Thuật toán quick sort:

Hàm quickSort và main

```
public static void quickSort(int a[], int l, int r){
    if(l < r){
        int m = (l + r) / 2;
        int p = partition(a, l, r);
        quickSort(a, l, p - 1);
        quickSort(a, p + 1, r);
    }
}
public static void main(String[] args) {
        int a[] = {3, 1, 0, 4, 2, 6, 5};
        quickSort(a, 0, 6);
        for(int x : a)
            System.out.print(x + " ");
}
```

Độ phức tạp: **O(NlogN)**

7. Comparator cho hàm sort:

Sử dụng comparator:

Để sắp xếp một cách linh hoạt, ta cần **sử dụng comparator** cho hàm sort. Comparator chỉ áp dụng khi các phần tử trong mảng là object hoặc áp dụng với array list.

7. Comparator cho hàm sort:

Sắp xế các phần tử tăng dần:

```
public static void main(String[] args) {
        Integer a[] = {3, 1, 0, 4, 2, 6, 5};
        Arrays.sort(a, new Comparator<Integer>(){
            @Override
            public int compare(Integer o1, Integer o2){
                if(o1 < o2)
                    return -1;
                else
                    return 1;
            }
        });
        for(int x : a) System.out.print(x + " ");
 }
```

OUTPUT

0 1 2 3 4 5 6

7. Comparator cho hàm sort:

Sắp xế các phần tử giảm dần:

```
public static void main(String[] args) {
        Integer a[] = {3, 1, 0, 4, 2, 6, 5};
        Arrays.sort(a, new Comparator<Integer>(){
            @Override
            public int compare(Integer o1, Integer o2){
                if(o1 > o2)
                    return -1;
                else
                    return 1;
            }
        });
        for(int x : a) System.out.print(x + " ");
 }
```

OUTPUT

6 5 4 3 2 1 0

7. Comparator cho hàm sort:

Như vậy nếu bạn muốn sắp khi sắp xếp, số đứng trước là o1 nhỏ hơn số đứng sau là o2 để mảng của mình đúng thứ tự tăng dần thì bạn trả về -1, ngược lại bạn trả về 1.

Sắp xế các phần tử tăng dần theo trị tuyệt đối:

```
public static void main(String[] args) {
        Integer a[] = {3, -1, 0, -4, 2, -6, 5};
        Arrays.sort(a, new Comparator<Integer>(){
            @Override
            public int compare(Integer o1, Integer o2){
                return Math.abs(o1) - Math.abs(o2);
            }
        });
        for(int x : a) System.out.print(x + " ");
    }
```

OUTPUT

```
0 -1 2 3 -4 5 -6
```

7. Comparator cho hàm sort:

Sắp xếp các phần tử theo tổng chữ số tăng dần, nếu 2 số có cùng tổng chữ số thì số nhỏ hơn sẽ xếp trước

Hàm tổng chữ số:

```
public static int tongChuSo(int n){
        int sum = 0;
        while(n != 0){
            sum += n % 10;
            n /= 10;
        }
        return sum;
    }
```

OUTPUT

11 20 3 111 42 8 35 71

Hàm main:

```
public static void main(String[] args) {
    Integer a[] = {3, 111, 20, 35, 42, 8, 71, 11};
    Arrays.sort(a, new Comparator<Integer>(){
        @Override
        public int compare(Integer o1, Integer o2) {
            if(tongChuSo(o1) != tongChuSo(o2))
                return tongChuSo(o1) - tongChuSo(o2);
            else
                return o1 - o2;
        }
    });
    for(int x : a) System.out.print(x + " ");
}
```