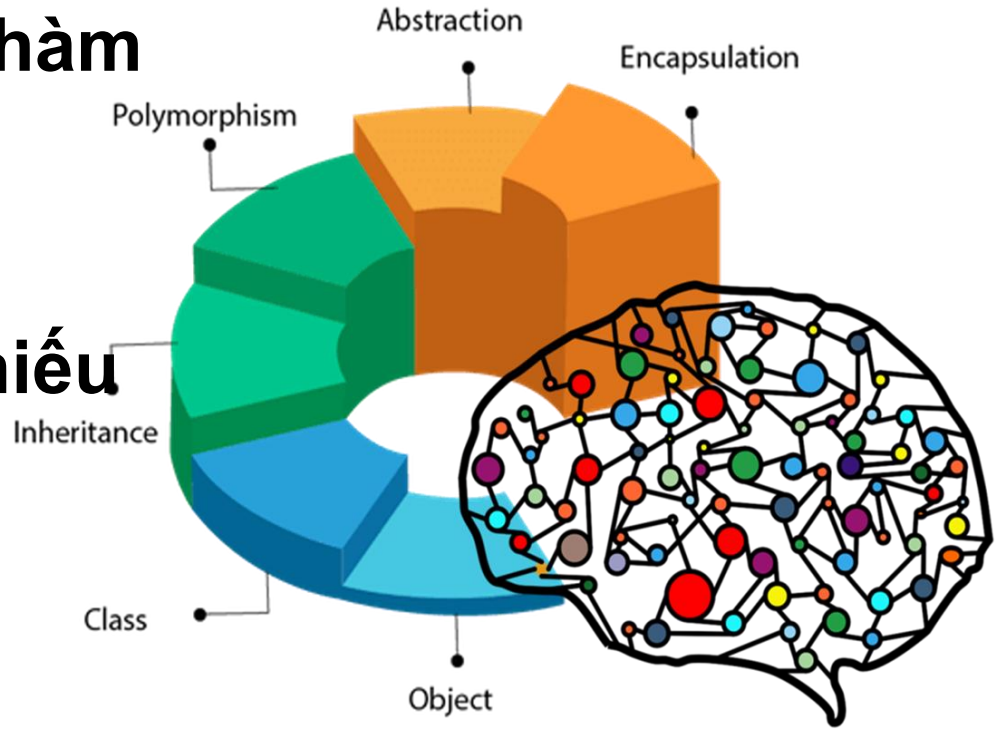
KỸ THUẬT LẬP TRÌNH

ĐẶNG VĂN NGHĨA
0975079414
nghiadv@donga.edu.vn

NỘI DUNG

1. Tổ chức chương trình thành các hàm
2. Xây dựng hàm và sử dụng hàm
3. Truyền tham trị và truyền tham chiếu
4. Đệ quy
5. Thuật toán sắp xếp và tìm kiếm



1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM

❖ **Hàm (phương thức)** là một đơn vị độc lập của chương trình. Các hàm có vai trò ngang nhau, vì vậy không cho phép xây dựng một hàm bên trong hàm khác.

❖ **Hàm được viết theo thứ tự sau**:

- **Dòng tiêu đề** chứa thông tin: từ khóa xác định phạm vi truy cập của hàm, kiểu hàm, tên hàm, kiểu và tên mỗi đối. **Ví dụ**: *static float maxThreeNumber(float num1, float num2, float num3)*
- **Thân hàm**:
  - ✓nằm ngay sau dòng tiêu đề;
  - ✓nội dung chính của hàm bắt đầu bằng dấu ngoặc nhọn mở { và kết thúc bởi dấu ngoặc nhọn đóng };

1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM

❖ **Hàm được viết theo thứ tự sau**:

- **Thân hàm**:
  - ✓chứa các câu lệnh;
  - ✓Có thể không sử dụng hoặc sử dụng 1 hoặc nhiều câu lệnh return đặt ở những vị trí khác nhau;
  - ✓Giá trị của biểu thức trong câu lệnh return sẽ được gán cho hàm.

❖ **Quy tắc hoạt động của hàm**:

- Lời gọi hàm có dạng như sau: *ten_ham (danh sách tham số thực);*

1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM

❖ **Quy tắc hoạt động của hàm**:

- Khi gặp lời gọi hàm thì bắt đầu thực hiện:
  - ✓Cấp phát bộ nhớ cho các đối và biến cục bộ;
  - ✓Gán giá trị của các tham số thực cho các đối tương ứng;
  - ✓Thực hiện các câu lệnh trong thân hàm;
  - ✓Gặp câu lệnh return hoặc dấu ngoặc nhọn đóng } cuối cùng của thân hàm thì máy sẽ xóa các đối, biến cục bộ và thoát khỏi hàm.

1. TỔ CHỨC CHƯƠNG TRÌNH THÀNH CÁC HÀM

❖ **Cấu trúc tổng quát của chương trình**:

```
package <package_name>;
import <other_package>;
public class ClassName {
    <Variables (also known as fields)>;
    <constructor method(s)>;
    <other methods>;
    public static void main(String[] args) {
      //statements
    }
}
```

2. XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM

❖ **Các khái niệm liên quan đến hàm:**

- Tên hàm
- Kiểu giá trị của hàm
- Đối hay tham số hình thức
- Thân hàm
- Khai báo hàm
- Lời gọi hàm
- Tham số thực

❖ **Mỗi hàm bao giờ cũng có 2 giai đoạn:**

- Xây dựng
- Sử dụng

2. XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM

❖ **Xây dựng hàm gồm các việc sau:**

- Từ khóa xác định phạm vi truy cập của hàm
- Kiểu dữ liệu
- Tên
- Danh sách đối số
- Các câu lệnh

❖ **Sử dụng hàm**:

- Thông qua lời gọi hàm: *ten_ham (danh sách các tham số thực)*
- Số tham số thực phải bằng số lượng đối số
- Kiểu của tham số thực phải phù hợp với kiểu của đối tương ứng.

2. XÂY DỰNG HÀM VÀ SỬ DỤNG HÀM

❖ **Nguyên tắc hoạt động của hàm:**

- Tên biến và tên đối cục bộ phải khác nhau (vì hoạt động trong cùng một hàm);
- Đối và biến cục bộ đều là các biến động. Chúng được cấp phát bộ nhớ khi hàm được gọi và bị xóa trước khi ra khỏi hàm.
- Không thể sử dụng đối để thay đổi giá trị của bất kỳ đại lượng ở bên ngoài hàm.
- Tên biến và đối cục bộ có thể trùng với tên của bất kỳ đại lượng ở bên ngoài hàm.
- Khi một hàm được gọi, việc đầu tiên là giá trị của các tham số thực được gán cho các đối.
- Các đối chính là bản sao của các tham số thực.
- Hàm chỉ làm việc trên các đối.
- Các đối có thể biến đổi trong thân hàm, nhưng các tham số thực (bản chính) không hề bị thay đổi.

3. TRUYỀN THAM TRỊ VÀ TRUYỀN THAM CHIẾU

- **Truyền tham trị** là gọi một phương thức và truyền giá trị cho phương thức đó.
- Việc thay đổi giá trị chỉ có hiệu lực bên trong phương thức được gọi, không có hiệu lực bên ngoài phương thức.
- **Truyền tham chiếu** là gọi một phương thức và truyền một tham chiếu cho phương thức đó.
- Việc thay đổi giá trị của biến tham chiếu bên trong phương thức làm thay đổi giá trị gốc của nó.

3. TRUYỀN THAM TRỊ VÀ TRUYỀN THAM CHIẾU

❖ **Ví dụ 1**: truyền tham trị

```
public class Example {
    int data = 50;

    void change(int data) {
        data = data + 100;
    }

    public static void main(String args[]) {
        Example exam = new Example();

        System.out.println("Trước khi thay đổi: " + exam.data);
        exam.change(500);
        System.out.println("Sau khi thay đổi: " + exam.data);
    }
}
```

Console

<terminated> Example [Java Application]

```
Trước khi thay đổi: 50
Sau khi thay đổi: 50
```

3. TRUYỀN THAM TRỊ VÀ TRUYỀN THAM CHIẾU

❖ **Ví dụ 2**: truyền tham chiếu

```
public class Example {
    int data = 50;

    void change(Example exam) {
        exam.data = exam.data + 100;
    }

    public static void main(String args[]) {
        Example exam = new Example();

        System.out.println("Trước khi thay đổi: " + exam.data);
        exam.change(exam);
        System.out.println("Sau khi thay đổi: " + exam.data);
    }
}
```

Console

```
<terminated> Example [Java Application]
Trước khi thay đổi: 50
Sau khi thay đổi: 150
```

4. ĐỆ QUY

- **Hàm đệ quy** cho phép từ một điểm trong thân của một hàm gọi đến chính hàm đó.
- Mỗi lần gọi hàm đệ quy, máy sẽ tạo ra một tập các biến cục bộ mới hoàn toàn độc lập với các tập biến (cục bộ) đã được tạo ra trong các lần gọi trước.
- Có bao nhiêu lần gọi đến hàm thì có bấy nhiêu lần thoát ra khỏi hàm và mỗi lần ra khỏi hàm thì một tập các biến cục bộ bị xóa.
- Sự tương ứng giữa các lần gọi đến hàm và các lần ra khỏi hàm được thực hiện theo thứ tự ngược, nghĩa là: lần ra đầu tiên ứng với lần vào cuối cùng và lần ra khỏi hàm cuối cùng ứng với lần đầu tiên gọi đến hàm.

4. ĐỆ QUY

❖ **Ưu điểm:**

- Đơn giản;
- Mất ít thời gian để viết, sửa lỗi và duy trì code.

❖ **Nhược điểm:**

- Vị trí lưu trữ mới cho biến được phân bổ trên stack khi thực hiện gọi đệ quy;
- Mỗi khi đệ quy trả về, các biến và tham số cũ được loại ra khỏi stack;
- Sử dụng nhiều bộ nhớ;
- Chạy chậm.

4. ĐỆ QUY

❖ **Cú pháp:**

```
returntype methodname() {
        //mã nguồn
        methodname();
    }
```

❖ **Ví dụ 1: Chương trình tính giai thừa sử dụng đệ quy**

```
public class Example {
    static int giaiThua(int n) {
        if (n == 1)
            return 1;
        else
            return (n * giaiThua(n - 1));
    }
    public static void main(String[] args) {
        System.out.println("Giai thừa của 5 là: " + giaiThua(5));
    }
}
```

Console

<terminated> Example [Java Application]

```
Giai thừa của 5 là: 120
```

4. ĐỆ QUY

❖ **Ví dụ 1**: Giải thích hoạt động

```
giaiThua(5) = 5*giaiThua(4)
   giaiThua(4) = 4*giaiThua(3)
      giaiThua(3) = 3*giaiThua(2)
         giaiThua(2) = 2*giaiThua(1)
            giaiThua(1)
               return 1
            return 2*1 = 2
         return 3*2 = 6
      return 4*6 = 24
   return 5*24 = 120
```

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Thuật toán sắp xếp:**

- Sắp xếp nổi bọt (Bubble Sort)
- Sắp xếp chọn (Selection Sort)
- Sắp xếp chèn (Insertion Sort)
- Sắp xếp trộn (Merge Sort)
- Sắp xếp nhanh (Quick Sort)

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Sắp xếp nổi bọt (Bubble Sort):** Duyệt qua danh sách, làm cho các phần tử nhỏ nhất hoặc lớn nhất dịch chuyển về cuối danh sách, tiếp tục thực hiện cho đến hết danh sách (làm cho các phần tử nhỏ nhất nổi lên). Độ phức tạp của thuật toán $O(n^2)$.

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Các bước thực hiện của giải thuật như sau**:

1. Gán i = 0
2. Gán j = 0
3. Nếu A[j] > A[j + 1] thì đổi chỗ A[j] và A[j + 1]
4. Nếu j < n – i – 1:
   - Đúng thì j = j + 1 và quay lại bước 3
   - Sai thì sang bước 5
5. Nếu i < n – 1:
   - Đúng thì i = i + 1 và quay lại bước 2
   - Sai thì dừng

```java
public class Example {
  static void bubbleSort(int a[]) {
    // lặp để duyệt qua mỗi phần tử của mảng
    for (int i = 0; i < a.length - 1; i++)
      // lặp để so sánh mỗi phần tử của mảng
      for (int j = 0; j < a.length - i - 1; j++)
        // So sánh 2 phần tử đứng kề nhau
        if (a[j] > a[j + 1]) {
          // Đổi chỗ nếu phần tử đứng không đúng thứ tự
          int temp = a[j];
          a[j] = a[j + 1];
          a[j + 1] = temp;
        }
  }
  static void display(int a[])
  {
         for(int i = 0; i < a.length; i++)
         {
                  System.out.print("\t" + a[i]);
         }
  }
  public static void main(String args[]) {
    int a[] = { -2, 45, 0, 11, -9 };
    System.out.print("Mảng ban đầu:");
    display(a);
    bubbleSort(a);
    System.out.print("\nMảng sau khi sắp xếp:");
    display(a);
  }
}
```

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Sắp xếp chọn (Selection Sort):** Duyệt từ đầu đến phần tử kế cuối danh sách, duyệt tìm phần tử nhỏ nhất từ vị trí kế phần tử đang duyệt đến hết, sau đó đổi vị trí của phần tử nhỏ nhất với phần tử đang duyệt và tiếp tục thực hiện. Độ phức tạp của thuật toán $O(n^2)$.

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Các bước thực hiện của giải thuật như sau**:

1. Gán i = 0
2. Gán j = i + 1 và min = i
3. Nếu j < n:
   - Nếu A[j] < A[min] thì min = j
   - j = j + 1
   - Quay lại bước 3
4. Đổi chỗ A[min] và A[i]
5. Nếu i < n – 1:
   - Đúng thì i = i + 1 và quay lại bước 2
   - Sai thì dừng

```
public class Example {
    static void selectionSort(int a[]) {
        for (int i = 0; i < a.length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < a.length; j++) {
           if (a[j] < a[min]) {
             min = j;
           }
        }
         int temp = a[i];
         a[i] = a[min];
         a[min] = temp;
       }
    }
    static void display(int a[])
    {
        for(int i = 0; i < a.length; i++)
        System.out.print("\t" + a[i]);
    }
    public static void main(String args[]) {
        int a[] = { 20, 12, 10, 15, 2 };
        System.out.print("Mảng ban đầu:");
        display(a);
        selectionSort(a);
        System.out.print("\nMảng sau khi sắp xếp:");
        display(a);
    }
}
```

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Sắp xếp chèn (Insertion Sort):** Mảng A ban đầu có phần tử A[0] xem như đã sắp xếp, ta sẽ duyệt từ phần tử 1 đến n – 1, tìm cách chèn những phần tử đó vào vị trí thích hợp trong mảng ban đầu đã được sắp xếp. Độ phức tạp của thuật toán $O(n^2)$.

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Các bước thực hiện của giải thuật như sau**:

1. Gán i = 1
2. Gán x = A[i] và j = i – 1
3. Nếu j >= 0 và A[j] > x:
   - A[j + 1] = A[j]
   - j = j – 1
   - Quay lại bước 3
4. A[j + 1] = x
5. Nếu i < n:
   - Đúng thì i = i + 1 và quay lại bước 2
   - Sai thì dừng

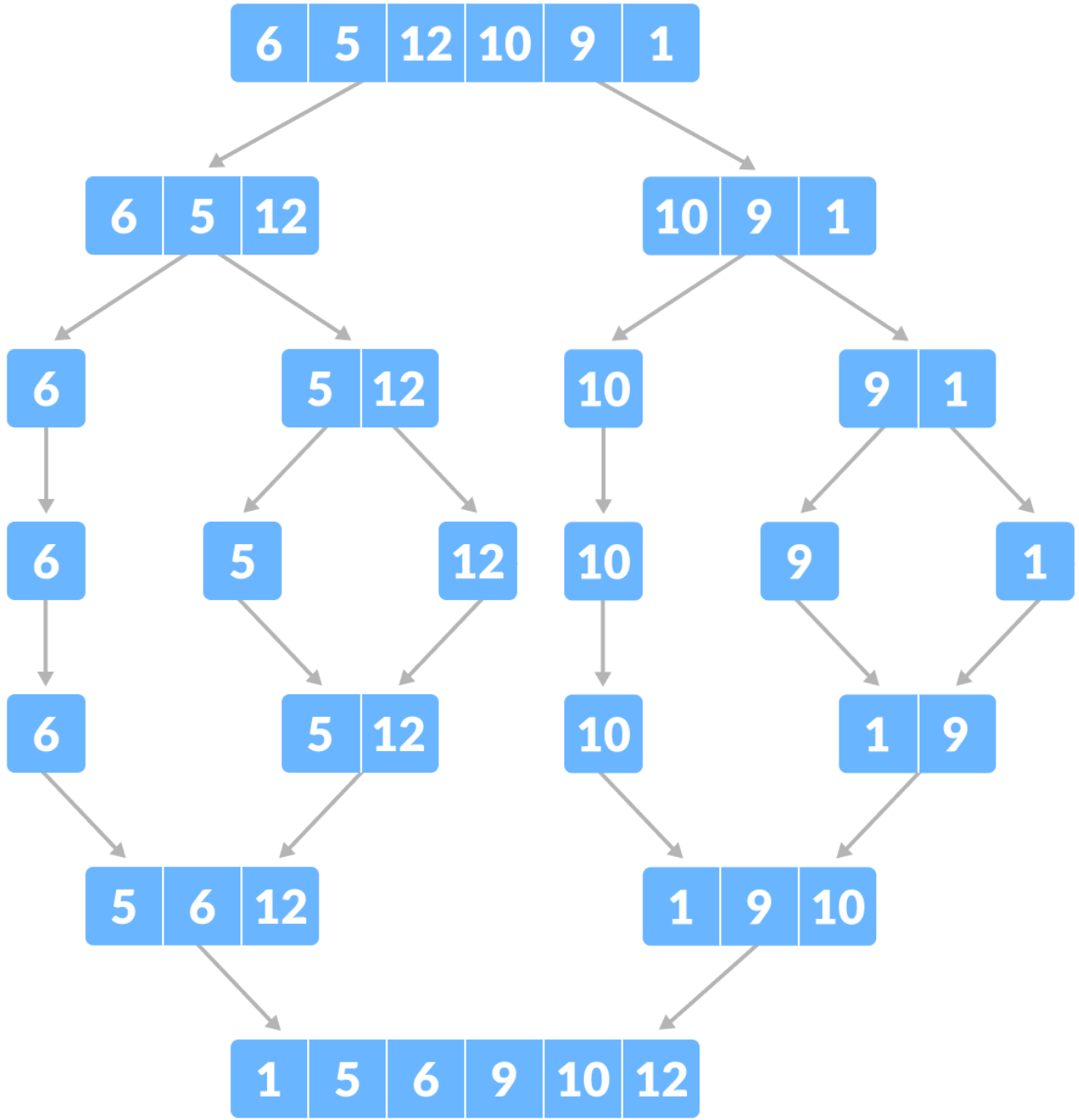
```java
public class Example {
  static void insertionSort(int a[]) {
    for (int i = 1; i < a.length; i++) {
      int key = a[i];
      int j = i - 1;
      while (j >= 0 && key < a[j]) {
        a[j + 1] = a[j];
        --j;
      }
      a[j + 1] = key;
    }
  }
  static void display(int a[])
  {
     for(int i = 0; i < a.length; i++)
         System.out.print("\t" + a[i]);
  }
  public static void main(String args[]) {
     int a[] = { 9, 5, 1, 4, 3 };
     System.out.print("Mảng ban đầu:");
     display(a);
     insertionSort(a);
     System.out.print("\nMảng sau khi sắp xếp:");
     display(a);
  }
}
```

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Sắp xếp trộn (Merge Sort):** là thuật toán dựa trên kỹ thuật chia để trị, thực hiện chia đôi mảng thành hai mảng con, sắp xếp hai mảng con và trộn lại theo đúng thứ tự, mảng con được sắp xếp bằng cách tương tự. Độ phức tạp của thuật toán O(n).

❖ **Các bước thực hiện của giải thuật như sau**:

- Nếu mảng còn có thể chia đôi được (tức left < right)
  - ✓Tìm vị trí chính giữa mảng
  - ✓Sắp xếp mảng thứ nhất (từ vị trí left đến mid)
  - ✓Sắp xếp mảng thứ 2 (từ vị trí mid + 1 đến right)
  - ✓Trộn hai mảng đã sắp xếp với nhau


6 5 12 10 9 1
6 5 12
10 9 1
6
5 12
10
9 1
6
5
12
10
9
1
6
5 12
10
1 9
5 6 12
1 9 10
1 5 6 9 10 12

```java
static void merge(int a[], int l, int m, int r) {
  int len1 = m - l + 1, len2 = r - m;
  int tam1[] = new int[len1];
  int tam2[] = new int[len2];
  for (int i = 0; i < len1; i++) //Tạo mảng tam1 <- a[l...m],
    tam1[i] = a[l + i];          //tam2 <- a[m+1...r]
  for (int j = 0; j < len2; j++)
    tam2[j] = a[m + 1 + j];
  int i, j, k; //chỉ số hiện tại của mảng con và mảng chính
  i = 0;
  j = 0;
  k = l;
while (i < len1 && j < len2) { //Duyệt qua các phần tử
   if (tam1[i] <= tam2[j]) { //của mảng tam1, tam2
      a[k] = tam1[i]; //so sánh và đưa vào đúng
      i++;            //vị trí trong mảng a[l...r]
    } else {
      a[k] = tam2[j];
      j++;
    }
    k++;
  }
 while (i < len1) {
    a[k] = tam1[i]; // Duyệt các phần tử còn lại
    i++;            // của tam1 hoặc tam2 và đưa vào đúng
    k++;            // vị trí trong mảng a[l...r]
  }
  while (j < len2) {
    a[k] = tam2[j];
    j++;
    k++;
  }
}
```

```java
static void mergeSort(int a[], int l, int r) {
    if (l < r) {
      int m = (l + r) / 2;//m là điểm chia mảng a
      mergeSort(a, l, m);// thành 2 mảng
      mergeSort(a, m + 1, r);
      merge(a, l, m, r);// trộn hai mảng đã được
    }                   // sắp xếp lại với nhau
  }
  // Hiển thị mảng
  static void display(int a[]) {
    for (int i = 0; i < a.length; ++i)
      System.out.print("\t" + a[i]);
  }
  public static void main(String args[]) {
    int a[] = { 6, 5, 12, 10, 9, 1 };
    System.out.print("Mảng ban đầu:");
    display(a);
    mergeSort(a, 0, arr.length - 1);
    System.out.print("\nMảng sau khi sắp xếp:");
    display(a);
  }
```

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Sắp xếp nhanh (Quick Sort):** là thuật toán sắp xếp dựa trên kỹ thuật chia để trị, thực hiện chọn một điểm làm chốt (gọi là pivot), sắp xếp mọi phần tử bên trái chốt đều nhỏ hơn chốt và mọi phần tử bên phải đều lớn hơn chốt, sau khi xong ta được 2 dãy con bên trái và bên phải, áp dụng tương tự cách sắp xếp này cho 2 dãy con vừa tìm cho đến khi dãy con chỉ còn 1 phần tử. Độ phức tạp của thuật toán O(nlog(n)).

❖ **Các bước thực hiện của giải thuật như sau**:

- Chọn một phần tử làm chốt
- Sắp xếp phần tử bên trái nhỏ hơn chốt
- Sắp xếp phần tử bên phải lớn hơn chốt
- Sắp xếp hai mảng con bên trái và bên phải pivot

```java
static void partition(int a[], int low, int high)
{
    // Kiểm tra xem nếu mảng có 1 phần tử thì không cần sắp xếp
    if (low >= high)
        return;
    int pivot = a[(low + high) / 2]; // Chọn phần tử chính giữa dãy làm chốt
    int i = low, j = high; // i là vị trí đầu và j là cuối đoạn
    while (i < j)
    {
        while (a[i] < pivot) //Nếu phần tử bên trái nhỏ hơn pivot thì bỏ qua
            i++;
        while (a[j] > pivot) //Nếu phần tử bên phải nhỏ hơn pivot thì bỏ qua
            j--;
        // Sau khi kết thúc hai vòng while ở trên thì chắc chắn
        // vị trí A[i] phải lớn hơn pivot và A[j] phải nhỏ hơn pivot
        // nếu i < j
        if (i <= j)
        {
            if (i < j) //nếu i != j (tức không trùng thì mới cần hoán đổi)
            {
            int tam = a[j]; //Thực hiện đổi chổ ta được
            a[j] = a[i];    // A[i] < pivot và A[j] > pivot
            a[i] = tam;
            }
            i++;
            j--;
        }
    }
    partition(a, low, j);// Gọi đệ quy sắp xếp dãy bên trái pivot
    partition(a, i, high);// Gọi đệ quy sắp xếp dãy bên phải pivot
}
```

```java
// Hàm sắp xếp chính
static void quickSort(int a[])
{
    partition(a, 0, a.length-1);
}
// Hàm hiển thị các phần tử của mảng
static void display(int a[])
{
for(int i = 0; i < a.length; i++ )
System.out.print("\t" + a[i]);
}
public static void main(String[] args) {
// Các câu lệnh xử lý
int a[] = {3, 2, 6, 4, 9};
System.out.print("Mảng ban đầu:");
display(a);
quickSort(a);
System.out.print("\nMảng sau khi sắp xếp:");
display(a);
}
```

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Thuật toán tìm kiếm:**

- Tìm kiếm tuyến tính
- Tìm kiếm nhị phân

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Tìm kiếm tuyến tính (linear search)/tìm kiếm tuần tự (sequential search)** là thuật toán tìm kiếm bằng cách duyệt qua tất cả phần tử của danh sách cho đến khi gặp phần tử cần tìm hoặc hết danh sách. Độ phức tạp của thuật toán O(n).

❖ **Ví dụ**: Cho mảng A gồm n phần tử. Hãy tìm phần tử x trong mảng A, thực hiện như sau:

1. Gán i = 0.
2. So sánh giá trị của A[i] và x:
   - Nếu A[i] == x thì dừng và trả về giá trị của i (vị trí của x trong mảng A).
   - Nếu A[i] != x thì sang bước 3.
3. Gán i = i + 1:
   - Nếu i == n (tức hết mảng) thì dừng lại và trả kết quả là -1 (không tìm thấy x).
   - Nếu i < n thì quay lại bước 2.

```java
public class Example {
     static int linearSearch(int a[], int x) {
       for (int i = 0; i < a.length; i++) {
          if (a[i] == x)
             return i;
       }
       return -1;
     }
     static void display(int a[])
     {
          for(int i = 0; i < a.length; i++)
                    System.out.print("\t" + a[i]);
     }
     public static void main(String args[]) {
       int a[] = { 2, 4, 0, 1, 9 };
       int x = 1;
       int result = linearSearch(a, x);
       System.out.print("Mảng ban đầu:");
       display(a);
       if (result == -1)
         System.out.printf("\nKhông tìm thấy phần tử %d trong mảng",x);
       else
         System.out.printf("\nTìm thấy phần tử %d trong mảng tại vị trí %d", x, result);
     }
}
```

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Tìm kiếm nhị phân** là thuật toán tìm kiếm dựa trên việc chia đôi khoảng đang xét sau mỗi lần lặp, sau đó xét tiếp trong nửa khoảng có khả năng chứa giá trị cần tìm, cứ như vậy cho đến khi không thể chia đôi được nữa. Thuật toán tìm kiếm nhị phân **chỉ áp dụng được cho danh sách đã sắp xếp**. Độ phức tạp của thuật toán O(log(n)) tốt hơn rất nhiều so với tìm kiếm tuyến tính.

5. THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

❖ **Ví dụ**: Cho mảng A gồm n phần tử. Hãy tìm phần tử x trong mảng A, thực hiện như sau:

1. Gán left = 0, right = n – 1.
2. Gán mid = (left + right) / 2 (lấy phần nguyên, đây là phần tử chính giữa của khoảng hiện tại)
   - Nếu như A[mid] == x: Dừng và trả về giá trị của mid (chính là vị trí của x trong mảng A).
   - Nếu như A[mid] > x (có thể x nằm trong nửa khoảng trước): right = mid – 1 // Tìm kiếm nửa khoảng trước
   - Nếu như A[mid] < x (có thể x nằm trong nửa khoảng sau): left = mid + 1 // Tìm kiếm nửa khoảng sau
3. Nếu left <= right:
   - Đúng thì quay lại bước 2 (còn chia đôi được).
   - Sai thì dừng và trả về kết quả -1 (không tìm thấy x)

```java
public class Example {
  static int binarySearch(int a[], int x, int low, int high) {
    while (low <= high) {
      int mid = (low + high) / 2;
      if (a[mid] == x)
        return mid;
      if (a[mid] < x)
        low = mid + 1;
      else
        high = mid - 1;
    }
    return -1;
  }
  static void display(int a[])
  {
       for(int i = 0; i < a.length; i++)
          System.out.print("\t" + a[i]);
  }
  public static void main(String args[]) {
    int a[] = { 3, 4, 5, 6, 7, 8, 9 };
    int n = a.length;
    int x = 4;
    int result = binarySearch(a, x, 0, n - 1);
    System.out.print("Mảng ban đầu:");
    display(a);
    if (result == -1)
      System.out.printf("\nKhông tìm thấy phần tử %d trong mảng",x);
    else
      System.out.printf("\nTìm thấy phần tử %d trong mảng tại vị trí %d",x,result);
  }
}
```