CMC UNIVERSITY

CHƯƠNG 6. PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

©CMC Univ. 2022

CMC UNIVERSITY

Nội dung

- ✓ Chia để trị
- ✓ Quy hoạch động

©CMC Univ. 2022

CMC UNIVERSITY

CHIA ĐỂ TRỊ

©CMC Univ. 2022

Chia để trị

- Phương pháp chia để trị (divide-and-conquer) chia một bài toán thành các bài toán con nhỏ hơn, sau đó giải quyết chúng; cuối cùng, kết hợp kết quả để thu được giải pháp toàn cục, tối ưu.
- Một bài toán có thể giải bằng phương pháp chia để trị nếu :

- Có thể định nghĩa bài toán như là sự kết hợp của các bài toán cùng loại nhưng nhỏ hơn. "Nhỏ hơn" nghĩa là gì? (Định nghĩa quy tắc phân rã bài toán)

- Trường hợp đặc biệt nào của bài toán có thể được coi là đủ nhỏ để giải quyết trực tiếp? (Xác định các bài toán cơ sở)

- Một số ví dụ về kỹ thuật thiết kế chia để trị như: tìm kiếm nhị phân, sắp xếp trộn, sắp xếp nhanh, thuật toán nhân nhanh, phép nhân ma trận Strassen, cặp điểm gần nhất, …

©CMC Univ. 2022

CMC UNIVERSITY

Tìm kiếm nhị phân

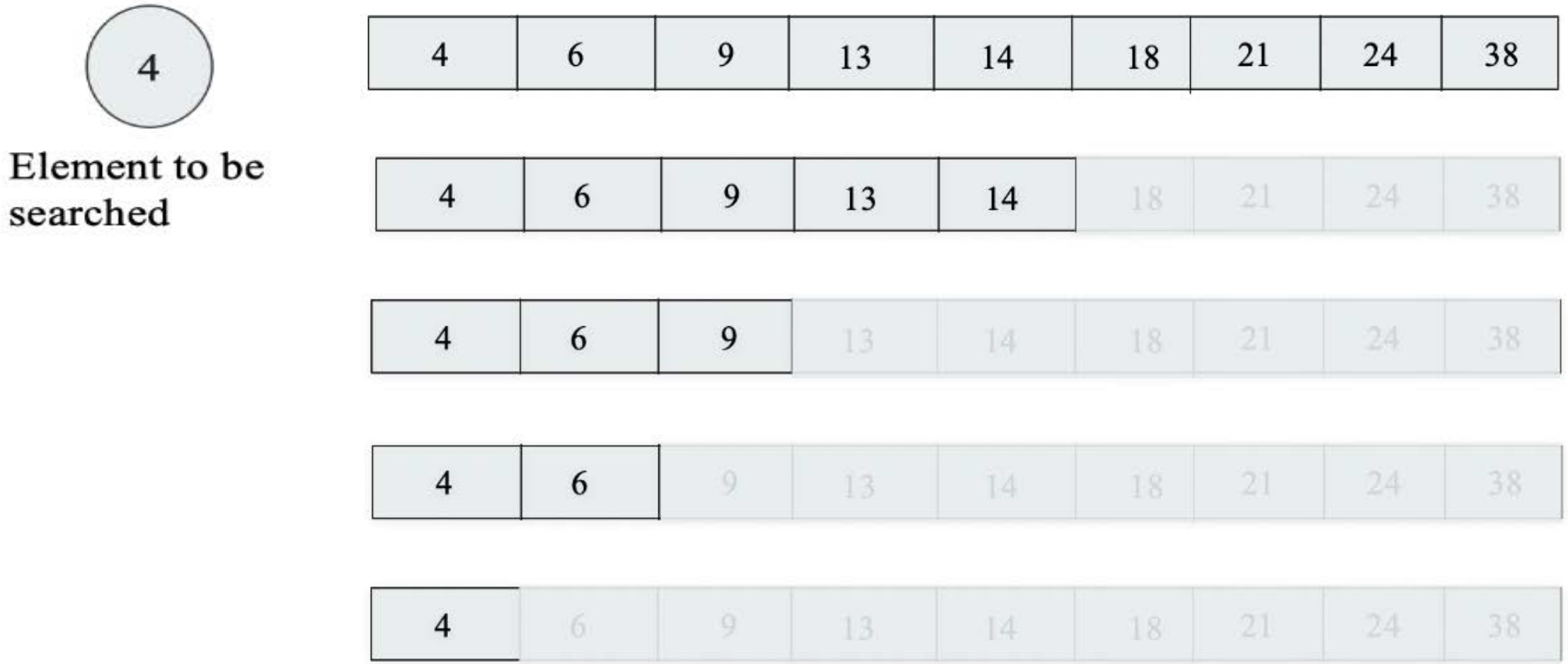
Thuật toán thực hiện tìm một phần tử cho trước từ danh sách đã được sắp xếp

- Đầu tiên, so sánh phần tử cần tìm với phần tử ở giữa danh sách; nếu phần tử cần tìm nhỏ hơn phần tử ở giữa, thì nửa danh sách có các phần tử lớn hơn phần tử ở giữa sẽ bị loại bỏ;
- Quá trình này lặp lại đệ quy cho đến khi tìm thấy phần tử cần tìm hoặc đến khi hết danh sách.
- Điều quan trọng cần lưu ý, trong mỗi lần lặp, một nửa không gian tìm kiếm bị loại bỏ, điều này cải thiện hiệu suất của toàn bộ thuật toán vì số lượng phần tử cần tìm kiếm ít hơn.

©CMC Univ. 2022

Tìm kiếm nhị phân

Ví dụ: Tìm số 4 trong danh sách các phần tử đã được sắp xếp



©CMC Univ. 2022

CMC UNIVERSITY

Tìm kiếm nhị phân

```java
public class BinarySearch {
    public static int binarySearch(int[] array, int key) {
        int left = 0;
        int right = array.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (array[mid] == key) {
                return mid;
            }
            if (array[mid] < key) {
                left = mid + 1;
            }
            else {
                right = mid - 1;
            }
        }
        return -1;   // Key was not found
    }
```

```java
    public static void main(String[] args) {
            int[] array = {4, 6, 9, 13, 14, 18, 21, 24, 38};
            int key = 13;
            int result = binarySearch(array, key);
            if (result == -1) {
                System.out.println("Element not present in the array");
            } else {
         System.out.println("Element found at index " + result);
            }
        }
}
```

Độ phức tạp thời gian trường hợp tồi nhất: O(log n)

©CMC Univ. 2022

CMC UNIVERSITY

Sắp xếp trộn

Sắp xếp trộn (Merge sort) là một thuật toán sắp xếp một danh sách gồm n số tự nhiên theo thứ tự tăng dần.

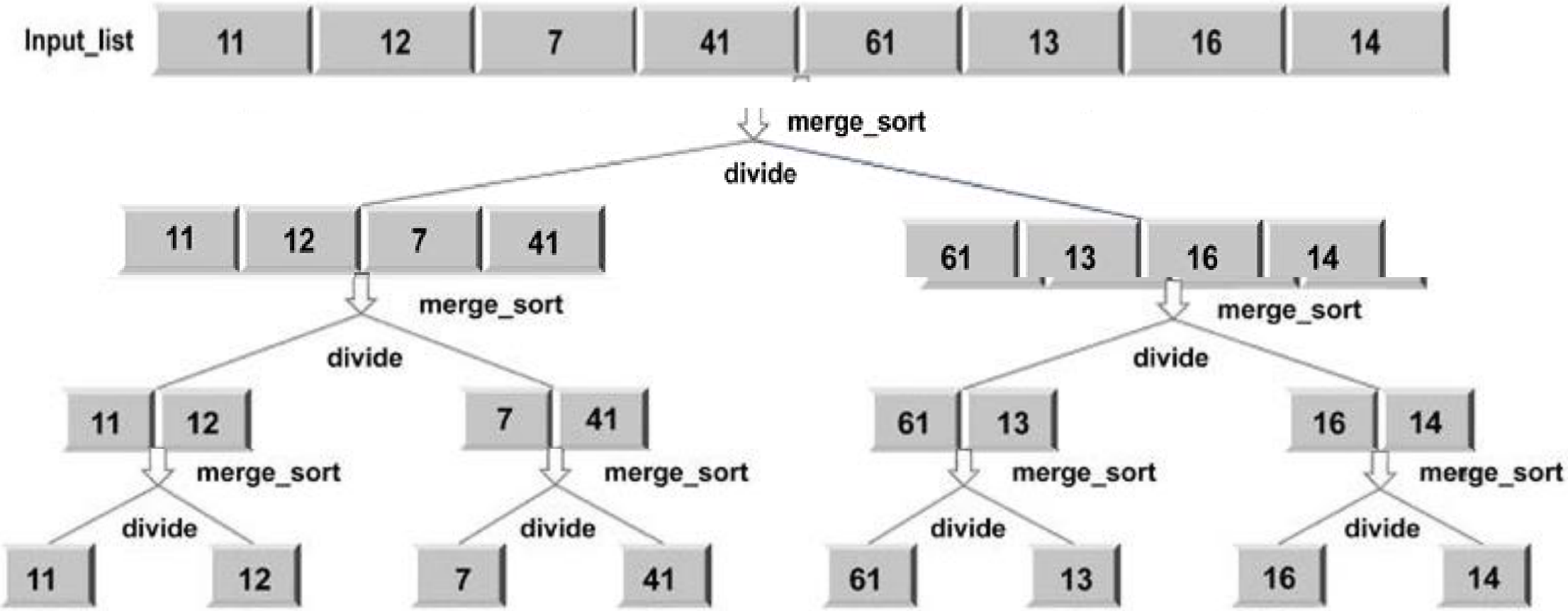
- Đầu tiên, danh sách các phần tử cho trước được chia ra thành các phần bằng nhau theo cách lặp lại cho đến khi mỗi danh sách con chứa một phần tử, sau đó các danh sách con này được kết hợp lại để tạo thành một danh sách mới theo thứ tự đã sắp xếp.
- Phương pháp này dựa trên phương pháp chia để trị và nhấn mạnh nhu cầu phân chia một bài toán thành các bài toán con nhỏ hơn cùng loại hoặc cùng hình thức như bài toán ban đầu. Các bài toán con này được giải quyết riêng biệt và sau đó kết hợp các kết quả để thu được giải pháp cho bài toán ban đầu.

Ví dụ 1: Cho một danh sách các phần tử chưa được sắp xếp, chia danh sách thành hai nửa xấp xỉ bằng nhau, tiếp tục chia danh sách thành các nửa một cách đệ quy.

©CMC Univ. 2022

Sắp xếp trộn

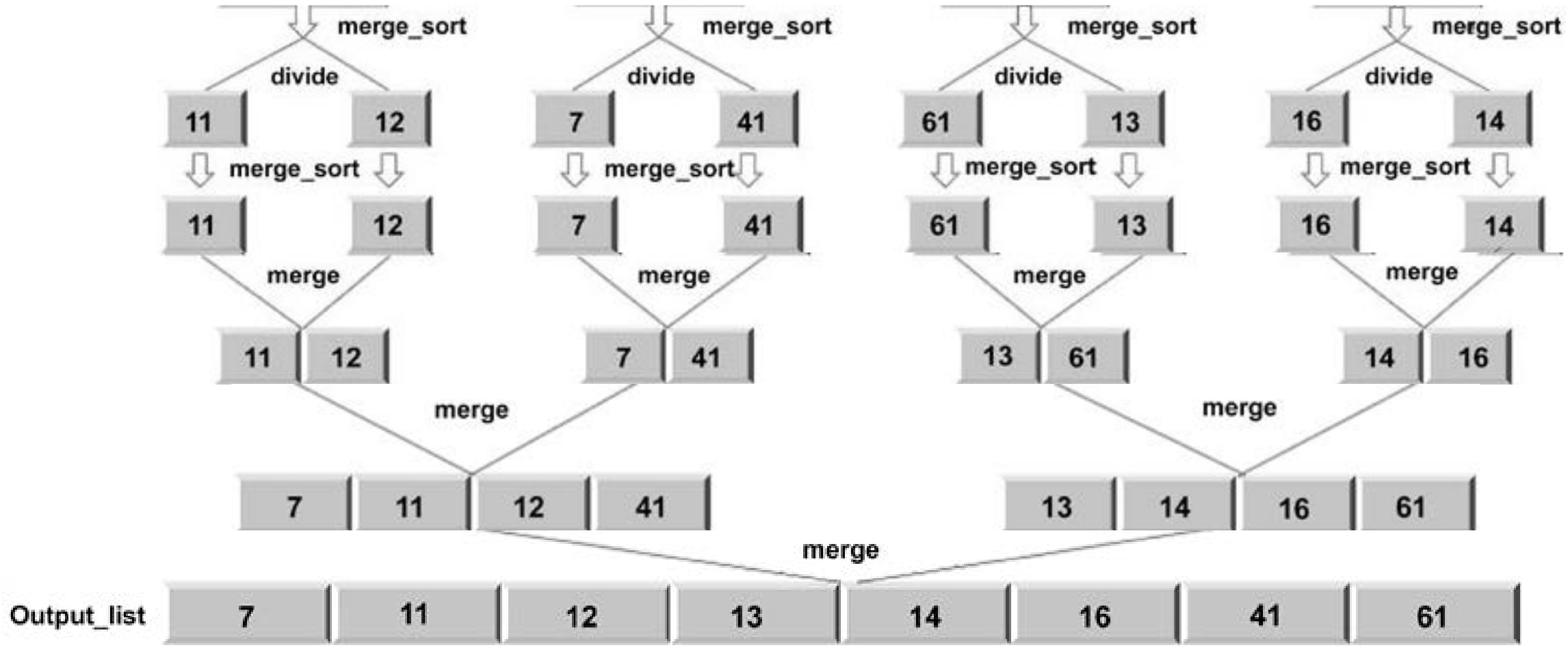
Sau một thời gian, danh sách con được tạo ra từ kết quả của lệnh gọi đệ quy sẽ chỉ chứa một phần tử. Tại thời điểm đó, ta hợp nhất các giải pháp trong bước chia hoặc trộn.



©CMC Uni

CMC UNIVERSITY

Sắp xếp trộn



©CMC Univ. 2022

CMC UNIVERSITY

Sắp xếp trộn

Việc triển khai thuật toán sắp xếp trộn được thực hiện bằng phương thức mergeSort, phương thức này đệ qui việc chia danh sách.

```
1  public class MergeSortList {
2      private static List<Integer> merge(List<Integer> left, List
           <Integer> right) {
3          List<Integer> merged = new ArrayList<>();
4          int i = 0, j = 0;
5          while (i < left.size() && j < right.size()) {
6              if (left.get(i) <= right.get(j)) {
7                  merged.add(left.get(i));
8                  i++;
9              } else {
10                 merged.add(right.get(j));
11                 j++;
12             }
13         }
14         while (i < left.size()) {
15             merged.add(left.get(i));
16             i++;
17         }
18         while (j < right.size()) {
19             merged.add(right.get(j));
20             j++;
21         }
22         return merged;
23     }
```

```
24     private static List<Integer> mergeSort(List<Integer> list)
           {
25         if (list.size() <= 1) {
26             return list;
27         }
28         int middle = list.size() / 2;
29         List<Integer> left = new ArrayList<>(list.subList(0,
               middle));
30         List<Integer> right = new ArrayList<>(list.subList
               (middle, list.size()));
31         return merge(mergeSort(left), mergeSort(right));
32     }
33     public static void main(String[] args) {
34         List<Integer> a = Arrays.asList(11, 12, 7, 41, 61, 13,
               16, 14);
35         System.out.println("Given List: " + a);
36         List<Integer> sortedA = mergeSort(a);
37         System.out.println("Sorted List: " + sortedA);
38     }
39 }
```

Sắp xếp trộn

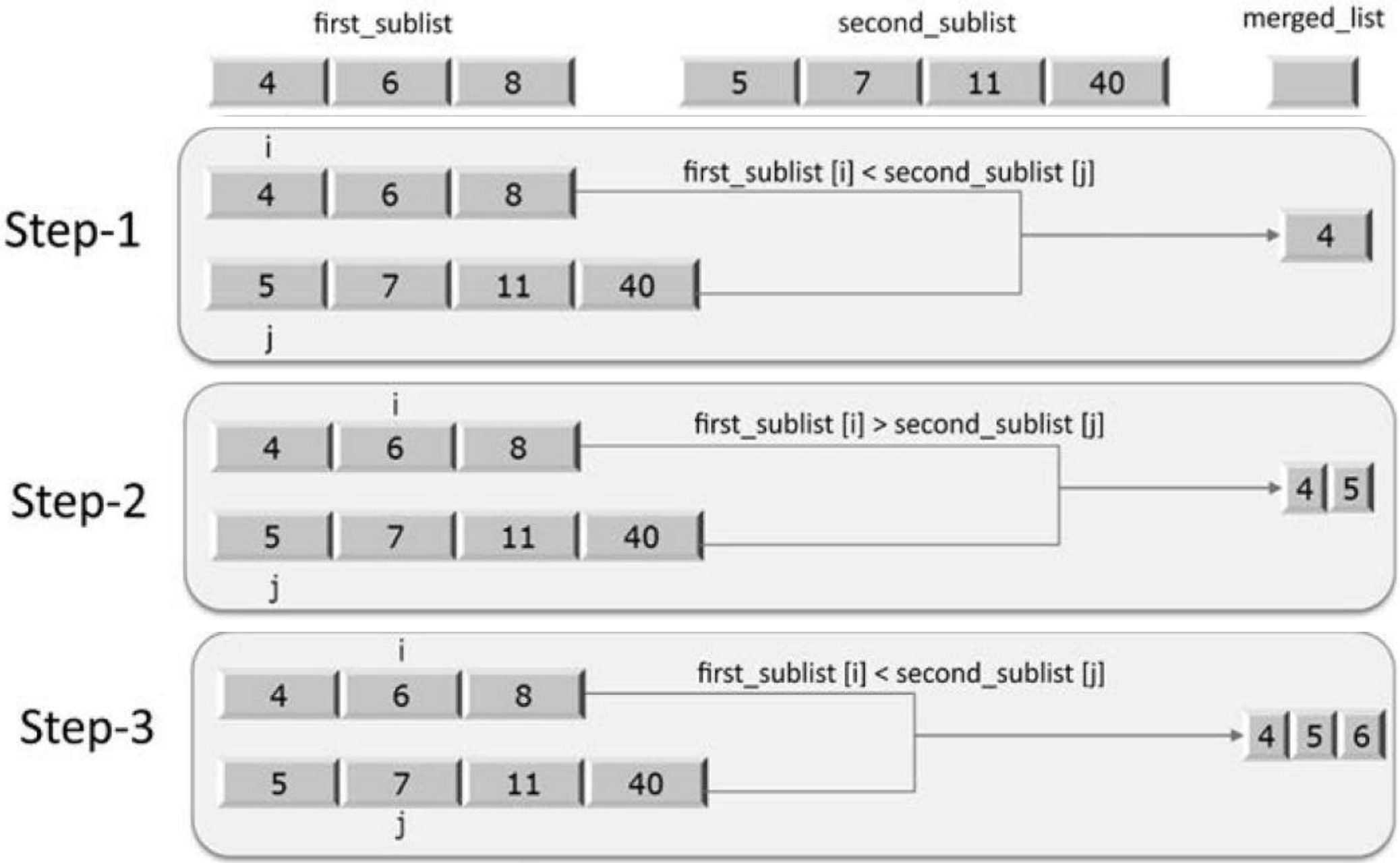
Ví dụ 2: Thực hiện thuật toán bằng cách trộn các danh sách [4, 6, 8] và [5, 7, 11, 40], như trong bảng sau:

Step	first_sublist	second_sublist	merged_list
0	[4 6 8]	[5 7 11 40]	[]
1	[6 8]	[5 7 11 40]	[4]
2	[6 8]	[7 11 40]	[4 5]
3	[8]	[7 11 40]	[4 5 6]
4	[8]	[11 40]	[4 5 6 7]
5	[ ]	[11 40]	[4 5 6 7 8]
6	[]	[ ]	[4 5 6 7 8 11 40]

©CMC Univ. 2022

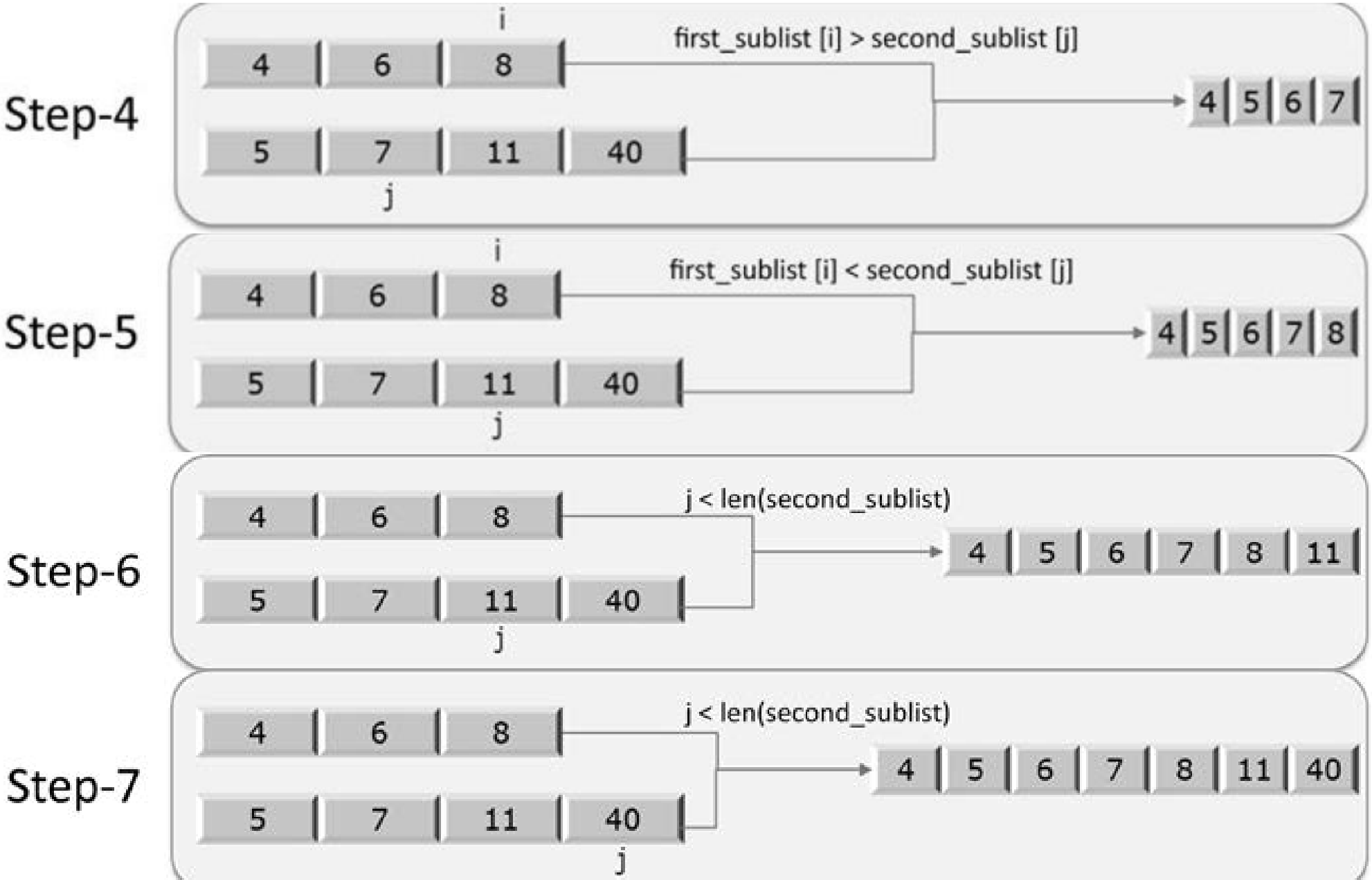
CMC UNIVERSITY

Sắp xếp trộn



CMC UNIVERSITY

Sắp xếp trộn



Sắp xếp trộn

Độ phức tạp thời gian trong trường hợp tồi nhất của thuật toán sắp xếp trộn sẽ phụ thuộc vào các bước sau :

1. Đầu tiên, bước chia sẽ có thời gian hằng số vì chỉ tính midpoint, tức là thời gian thực hiện là O(1)

2. Sau đó, trong mỗi lần lặp, ta chia danh sách thành một nửa theo cách đệ quy, tương tự như thuật toán tìm kiếm nhị phân, thời gian thực hiện là O(logn).

3. Hơn nữa, bước kết hợp sẽ hợp nhất tất cả n phần tử vào mảng ban đầu, việc này sẽ mất thời gian O(n).

Do vậy, thuật toán sắp xếp trộn có độ phức tạp thời gian:

$$O(n) * O(\log n) = O(n \log n).$$

©CMC Univ. 2022

CMC UNIVERSITY

Bài tập

1. Write a recursive algorithm and divide-and-conquer algorithm to calculate the power of a positive integer (Given two positive integers x, n. Let's calculate the value of x^n)

2. Write a recursive algorithm and divide-and-conquer algorithm to find the factorial of a positive integer ($n!$)

3. Write a recursive algorithm and divide-and-conquer algorithm that multiplies two polynomials

Given two polynomials $A(x) = \sum_{i=0}^{m} a_i x^i$ and $B(x) = \sum_{j=0}^{n} b_j x^j$, find polynomial $C(x) = A(x)B(x) = \sum_{k=0}^{m+n} c_k x^k$

The main task of this problem is to find $c_k = \sum_{i+j=k} a_i b_j, \forall k: 0 \le k \le m+n$

©CMC Univ. 2022

CMC UNIVERSITY

QUY HOẠCH ĐỘNG

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán quy hoạch động

Trong toán học và khoa học máy tính, ***quy hoạch động*** (dynamic programming) là một phương pháp hiệu quả giải những bài toán tối ưu có ba tính chất sau đây:

1. Dạng đệ qui: Bài toán lớn có thể phân rã thành những bài toán con đồng dạng, những bài toán con đó có thể phân rã thành những bài toán nhỏ hơn nữa …

2. Cấu trúc con tối ưu: Có thể sử dụng lời giải tối ưu của các bài toán con để tìm ra lời giải tối ưu của bài toán lớn.

3. Bài toán con chồng nhau: Hai bài toán con trong quá trình phân rã có thể có chung một số bài toán con khác.

Tính chất thứ nhất và thứ hai là điều kiện cần của bài toán quy hoạch động.

Tính chất thứ ba nêu lên đặc điểm của một bài toán mà cách giải bằng phương pháp quy hoạch động hiệu quả hơn hẳn so với phương pháp giải đệ quy thông thường

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán quy hoạch động

- Bài toán giải theo phương pháp quy hoạch động gọi là ***bài toán quy hoạch động*** (dynamic programming problem)

- Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là ***công thức truy hồi*** (recursive formula) của quy hoạch động.

- Tập các bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là ***cơ sở quy hoạch động*** (dynamic programming basis).

- Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là ***bảng phương án*** (solution table) của quy hoạch động.

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán quy hoạch động

Giải bài toán quy hoạch động:

Quy hoạch động = Chia để trị + Ghi nhớ

1. Phân tích bài toán: kiểm tra ba tính chất của bài toán quy hoạch động
2. Chia bài toán đã cho thành các bài toán con tương tự
3. Phát triển cách tìm lời giải của bài toán lớn khi đã biết lời giải của các bài toán con - tìm công thức truy hồi
4. Chỉ ra cách hiệu quả để lưu trữ kết quả của các bài toán con

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán quy hoạch động

Ta cần một cách hiệu quả để lưu trữ kết quả của từng bài toán phụ, có các kỹ thuật sau:

- **Top-down**:

- Bắt đầu từ tập bài toán ban đầu và chia nó thành các bài toán con nhỏ
- Xác định lời giải của bài toán con và lưu trữ kết quả của nó (Sau này, khi gặp bài toán con này, ta chỉ trả về kết quả tính toán trước của nó).
- Nếu lời giải của một bài toán đã cho có thể được xây dựng một cách đệ quy bằng cách sử dụng lời giải của các bài toán con, thì lời giải của các bài toán con chồng nhau có thể dễ dàng được ghi nhớ.

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán quy hoạch động

Ghi nhớ có nghĩa là lưu trữ lời giải của bài toán con trong một mảng hoặc bảng băm. Bất cứ khi nào cần giải một bài toán con, trước tiên nó được tham chiếu đến các giá trị đã lưu nếu nó đã được tính toán và nếu nó chưa được lưu trữ thì nó sẽ được tính toán theo cách thông thường. Thủ tục này được gọi là ***ghi nhớ***, có nghĩa là nó "nhớ" kết quả của hoạt động đã được tính toán trước đó

- **Bottom-up**:

- Một bài toán đã cho được giải bằng cách chia nó thành các bài toán con một cách đệ quy, sau đó giải các bài toán con nhỏ nhất có thể.
- Lời giải của các bài toán con được kết hợp theo kiểu từ dưới lên để đi đến lời giải của bài toán con lớn hơn nhằm đạt đến lời giải cuối cùng một cách đệ quy.

©CMC Univ. 2022

CMC UNIVERSITY

Tính dãy Fibonacci

Chuỗi Fibonacci có thể được chứng minh bằng cách sử dụng quan hệ truy hồi

```
func(0) = 1
func(1) = 1
func(n) = func(n-1) + func(n-2) for n>1
```

Một chương trình kiểu đệ quy để tạo chuỗi sẽ như sau :

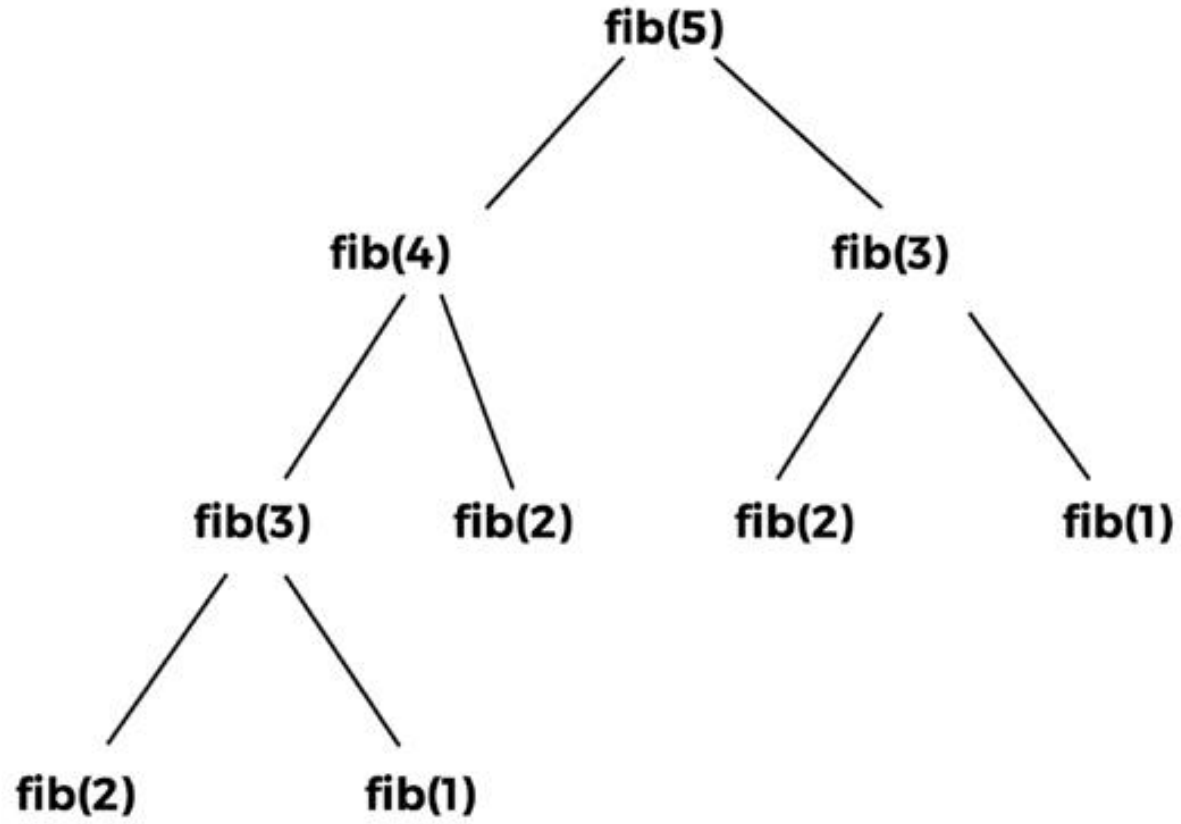
```java
public class Fibonacci {
    public static int fibonacci(int n) {
        if (n <= 1) {
            return n;
        }
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
    public static void main(String[] args) {
        int n = 5;
        System.out.println("Fibonacci sequence up to " + n +":");
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacci(i) + " ");
        }
    }
}
```

©CMC Univ. 2022

Tính dãy Fibonacci

Trong đoạn mã trên, có thể thấy rằng các lệnh gọi đệ quy đang được gọi để giải quyết bài toán.

- Khi gặp trường hợp cơ sở, $n \leq 1$, hàm fib() trả ra 1.
- Nếu không gặp trường hợp cơ sở, gọi lại hàm fib(). Cây đệ qui giải tới số hạng thứ 5 trong dãy Fibonacci được chỉ ra trong hình sau:



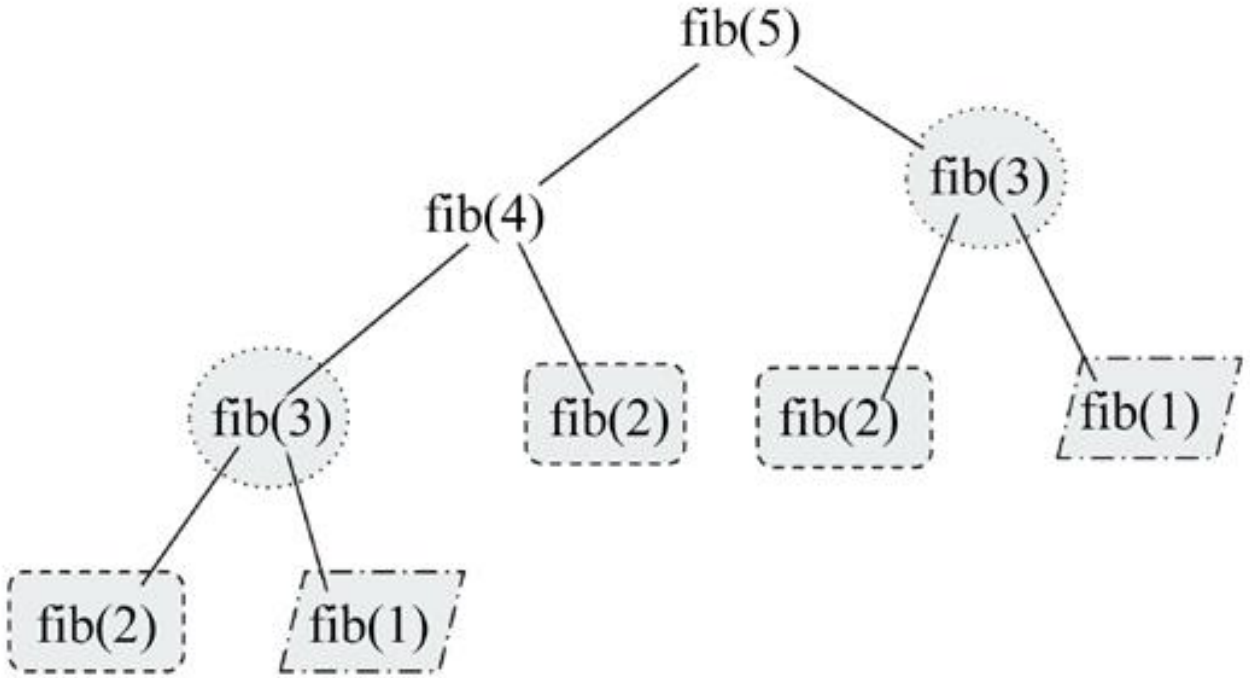
©CMC Univ. 2022

Tính dãy Fibonacci

Các bài toán con chồng nhau từ cây đệ qui được chỉ ra trong hình dưới: lệnh gọi fib(1) xảy ra 2 lần, lệnh gọi fib(2) xảy ra 3 lần, và gọi fib(3) xuất hiện 2 lần. Các giá trị trả về của cùng một lệnh gọi hàm luôn không đổi.

Như vậy, fib(1), fib(2), fib(3) là các bài toán chồng nhau. Thời gian tính toán sẽ bị lãng phí nếu mỗi lần gọi một hàm ta lại phải tính lại nó.

Các lệnh gọi lặp lại này tới một hàm có cùng tham số và đầu ra cho thấy có sự trùng lặp. Một số phép tính nhất định lặp lại trong các bài toán con nhỏ hơn



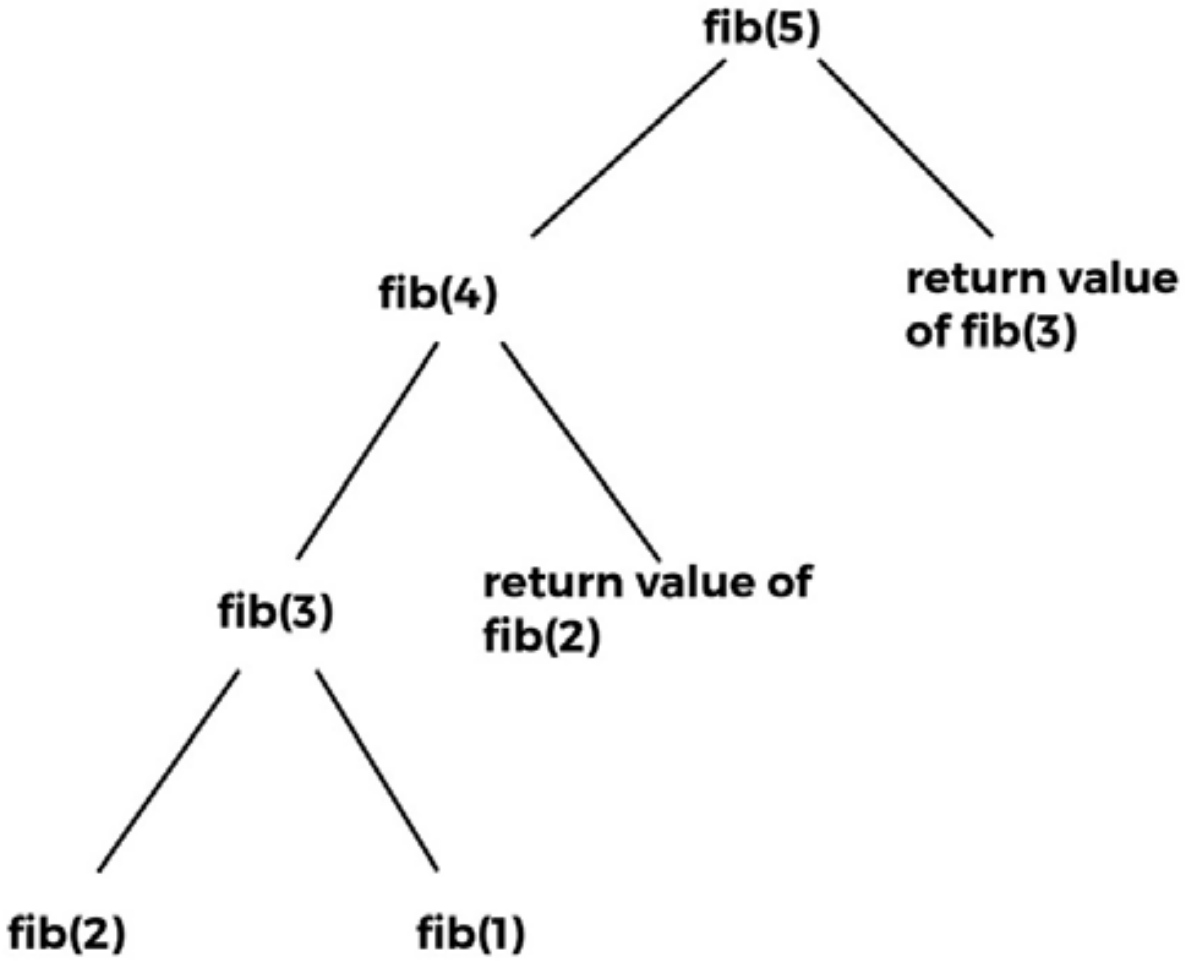
©CMC Univ. 2022

Tính dãy Fibonacci

Trong qui hoạch động sử dụng kỹ thuật ghi nhớ:

- Lưu các kết quả tính toán fib(1), fib(2) và fib(3) lần đầu tiên gặp chúng

- Sau đó, mỗi khi có lời gọi đến fib(1), fib(2), hoặc fib(3) chỉ cần trả về kết quả tương ứng của chúng. Sơ đồ cây đệ qui được chỉ ra trong hình bên:



©CMC Univ. 2022

CMC UNIVERSITY

Tính dãy Fibonacci

Khi thực hiện phương pháp qui hoạch động tính dãy Fibonacci, ta lưu các kết quả của các bài toán con đã được giải trong HashMap có tên memo.

Đầu tiên kiểm tra Fibonacci của một số bất kỳ đã được tính chưa:

- Nếu nó đã được tính, thì trả ra giá trị được lưu trong memo.get(n).
- Sau khi tính nghiệm của bài toán con, lại lưu nó nó trong memo.

```java
import java.util.HashMap;
import java.util.Map;

public class FibonacciDynamic {
    private static Map<Integer,Integer> memo = new HashMap<>();

    public static int fibonacci(int n) {
        if (n <= 1) {
            return n;
        }
        if (memo.containsKey(n)) {
            return memo.get(n);
        }
        int result = fibonacci(n - 1) + fibonacci(n - 2);
        memo.put(n, result);
        return result;
    }
    public static void main(String[] args) {
        int n = 5;
        System.out.println("Fibonacci series up to " + n +":");
        for (int i = 0; i <= n; i++) {
            System.out.print(fibonacci(i) + " ");
        }
    }
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Tính dãy Fibonacci

Quy hoạch động cải thiện độ phức tạp về thời gian chạy của thuật toán.

- Trong phương pháp đệ qui, hai hàm sẽ được gọi với mỗi giá trị

  Ví dụ, fib(5) gọi fib(4) và fib(3),

  fib(4) gọi fib(3) và fib(2),

  ...

  Như vậy, độ phức tạp thời gian cho phương pháp đệ qui là $O(2^n)$

- Trong phương pháp quy hoạch động, ta không tính lại các bài toán con, do vậy với fib(n), ta có tổng số n giá trị được tính, gồm fib(0), fib(1), fib(2), …, fib(n).

  Do vậy, độ phức tạp thời gian tổng là O(n).

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán cái túi (Knapsack)

Bài toán: Một nhà thám hiểm cần mang một cái túi trọng lượng không quá m. Có n đồ vật có thể mang theo. Đồ vật thứ i có trọng lượng w_i và giá trị sử dụng v_i. Những đồ vật nào sẽ được mang theo sao cho tổng giá trị sử dụng của chúng là lớn nhất ?

Giải:

Gọi $f[i, j]$ là giá trị lớn nhất có thể bằng cách chọn trong số các đồ vật $\{1,2, \ldots, i\}$ có giới hạn trọng lượng j. Mục đích của ta là tìm $f[n, m]$: giá trị lớn nhất có thể bằng cách chọn trong n đồ vật đã cho với giới hạn trọng lượng m.

Với trọng lượng giới hạn j, chọn từ các đối tượng $\{1,2, \ldots, i\}$ để đạt giá trị lớn nhất sẽ có một trong hai khả năng:

- Nếu đồ vật thứ i không được chọn, thì $f[i, j]$ là giá trị lớn nhất có thể bằng cách chọn trong số các đồ vật $\{1,2, \ldots, i-1\}$ với trọng số giới hạn j. Tức là, $f[i, j] = f[i - 1, j]$

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán cái túi

- Nếu đồ vật thứ i được chọn (chỉ xét khi $w_i \leq j$), thì $f[i, j]$ bằng với giá trị của đồ vật thứ i ($= v_i$) cộng với giá trị lớn nhất có thể đạt được bằng cách chọn trong số các đồ vật $\{1,2, \ldots, i-1\}$ với giới hạn trọng số $j - w_i$ ($= f[i-1, j-w_i]$). Tức là,

$$f[i, j] = f[i-1, j-w_i] + v_i.$$

Bằng cách xây dựng $f[i, j]$, ta có công thức truy hồi:

$$f[i,j] = \begin{cases} f[i-1, j], & if\ i < w_i \\ \max\{f[i-1, j], f[i-1, j-w_i] + v_i\}, & if\ j \geq w_i \end{cases}$$

Để tính các phần tử trên hàng i của bảng f, ta cần biết các phần tử ở hàng trước đó (hàng i-1). Do vậy, nếu biết hàng 0 của bảng f, có thể tính được mọi phần tử trong bảng. Từ đó suy ra cơ sở quy hoạch động $f[0, j]$, theo định nghĩa là giá trị lớn nhất có thể bằng cách chọn trong số 0 đồ vật, do vậy hiển nhiên $f[0, j] = 0$.

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán cái túi

Sau khi tính xong bảng phương án, ta quan tâm đến $f[n, m]$, là giá trị lớn nhất thu được khi chọn trong tất cả n đồ vật với trọng lượng giới hạn m. Việc còn lại là chỉ ra phương án chọn các đối tượng.

- Nếu $f[n, m] = f[n-1, m]$ (phương án tối ưu chọn đồ vật n), truy tiếp $f[n-1, m]$
- Nếu $f[n, m] \neq f[n-1, m]$ (phương án tối ưu không chọn đồ vật n), truy tiếp $f[n-1, m-w_i]$.

Tiếp tục truy vết cho đến khi đến được hàng 0 của bảng phương án

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán cái túi

```java
public class KnapsackDP {

    public static int knapsack(int W, int[] wt, int[] v, int n)
        {
        int[][] dp = new int[n + 1][W + 1];

        // Build the table in bottom-up manner
        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= W; w++) {
                if (i == 0 || w == 0) {
                    dp[i][w] = 0;
                } else if (wt[i - 1] <= w) {
                    dp[i][w] = Math.max(v[i - 1] + dp[i - 1]
                        [w - wt[i - 1]], dp[i - 1][w]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }
        return dp[n][W];
    }
```

```java
    public static void main(String[] args) {
        int[] v = {3, 4, 5, 6, 9, 15, 20};
        int[] wt = {2, 3, 4, 5, 6, 7, 8};
        int W = 20;
        int n = val.length;
        System.out.println("Maximum value in Knapsack = " +
            knapsack(W, wt, v, n));
    }
}
```

```
Maximum value: 42
```

0, 1, 5, 6

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán cái túi

Inputs:

- W: Trọng lượng tối đa của cái túi.
- wt[]: Mảng biểu diễn trọng lượng của các đồ vật.
- v[]: Mảng biểu diễn giá trị của các đồ vật.
- n: Số lượng đồ vật.

Bảng qui hoạch động:

Tạo mảng 2 chiều dp, trong đó dp[i][w] biểu diễn giá trị lớn nhất có thể đạt được với i đồ vật đầu tiên và trọng lượng túi là w.

©CMC Univ. 2022

CMC UNIVERSITY

Bài toán cái túi

Điền giá trị vào bảng:

- Lặp lại từng đồ vật (i từ 0 đến n) và mỗi trọng lượng có thể (w từ 0 đến W).
- Nếu đồ vật hiện tại i không được chọn (i.e., i == 0 hoặc w == 0), giá trị là 0.
- Nếu trọng lượng của đồ vật hiện tại wt[i-1] <= w, ta xem xét giá trị lớn nhất giữa việc chọn đồ vật đó và việc không chọn đồ vật đó.
- Nếu trọng lượng của đồ vật hiện tại wt[i-1] > w, không thể chọn đồ vật này, và giá trị vẫn là dp[i-1][w].

Kết quả:

Giá trị dp[n][W] là giá trị lớn nhất có thể đưa vào túi có trọng lượng W.

©CMC Univ. 2022

CMC UNIVERSITY

Bài tập

1. Cho số tự nhiên $n <= 400$. Hãy cho biết có bao nhiêu cách phân tích số n thành tổng của dãy các số nguyên dương, các cách phân tích là hoán vị của nhau chỉ tính là một cách.

Ví dụ, $n = 5$ có 7 cách phân tích:

1. 5 = 1 + 1 + 1 + 1 + 1
2. 5 = 1 + 1 + 1 + 2
3. 5 = 1 + 1 + 3
4. 5 = 1 + 2 + 2
5. 5 = 1 + 4
6. 5 = 2 + 3
7. 5 = 5

©CMC Univ. 2022

Bài tập

2. Cho dãy số nguyên $A = (a_1, a_2, \ldots, a_n)$. Một dãy con của A là một cách lựa chọn từ A một số phần tử giữ nguyên thứ tự. Như vậy, có 2^n dãy con.

Hãy tìm dãy con đơn điệu tăng của A có độ dài lớn nhất.

Tức là, tìm số nguyên lớn nhất k và dãy chỉ số $i_1 < i_2 < \ldots < i_k$ sao cho $a_{i_1} < a_{i_2} < \ldots < a_{i_k}$

Ví dụ:

Sample Input	Sample Output
12 1 2 3 8 9 4 5 6 2 3 9 10	Length = 8 a[1] = 1 a[2] = 2 a[3] = 3 a[6] = 4 a[7] = 5 a[8] = 6 a[11] = 9 a[12] = 10

©CMC Univ. 2022


CMC UNIVERSITY
THANK YOU