

Chapter 9

Sorting

Data Structures and Algorithms in Java

Content

- Insertion Sort (**Day 1**) $\rightarrow O(n^2)$
- Selection sort (**Day 1**) $\rightarrow O(n^2)$
- Bubble Sort (**Day 1**) $\rightarrow O(n^2)$
- Decision Tree (**Day 1**)
- Shell Sort (Day 3) \rightarrow depends on gap sequence
- **Heap Sort (Day 2)** $\rightarrow O(n * \log n)$
- **Quick Sort (Day 2)** \rightarrow Worst: $O(n^2)$, average: $O(n * \log n)$
- Merge Sort (Day 3) $\rightarrow O(n \log n)$
- **Radix Sort (Day 2)** $\rightarrow O(n)$
- Java util. (Day 3)

Data Structures and Algorithms in Java

2

Objectives

- Sorting: a process that will swap elements in a group such that all elements satisfy pre-defined order based on some criteria.

- Natural criteria: numerical order, dictionary order

Discuss the following topics:

- Elementary Sorting Algorithms
- Decision Trees
- Efficient Sorting Algorithms
- Sorting in `java.util`
- Case Study: Adding Polynomials

Data Structures and Algorithms in Java

3

Chèn Trực Tiếp – Insertion Sort

- Giả sử có một dãy a_0, a_1, \dots, a_{n-1} trong đó phần tử đầu tiên a_0, a_1, \dots, a_{i-1} đã có thứ tự.
- Tìm cách chèn phần tử a_i vào **vị trí thích hợp** của đoạn đã được sắp để có dãy mới a_0, a_1, \dots, a_i trở nên có thứ tự. Vị trí này chính là vị trí giữa hai phần tử a_{k-1} và a_k thỏa $a_{k-1} < a_i < a_k$ ($1 \leq k \leq i$).

Data Structures and Algorithms in Java

4

Chèn Trực Tiếp – Insertion Sort

- **Bước 1:** $i = 1$; //giả sử có đoạn $a[1]$ đã được sắp
- **Bước 2:** $x = a[i]$; Tìm vị trí pos thích hợp trong đoạn $a[1]$ đến $a[i-1]$ để chèn $a[i]$ vào
- **Bước 3:** Dời chỗ các phần tử từ $a[pos]$ đến $a[i-1]$ sang phải 1 vị trí để dành chỗ cho $a[i]$
- **Bước 4:** $a[pos] = x$; //có đoạn $a[1]..a[i]$ đã được sắp
- **Bước 5:** $i = i + 1$;

Nếu $i < n$: Lặp lại Bước 2

Ngược lại : Dừng

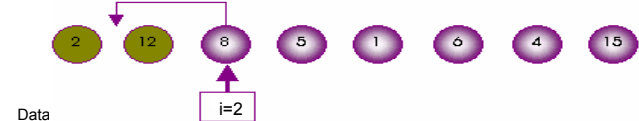
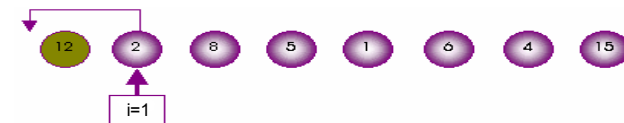
Data Structures and Algorithms in Java

5

Chèn Trực Tiếp – Insertion Sort

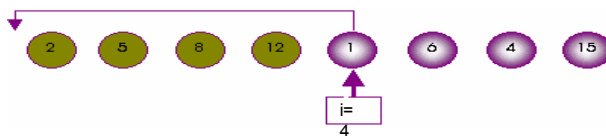
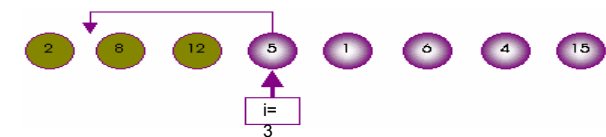
➤ Cho dãy số :

12 2 8 5 1 6 4 15

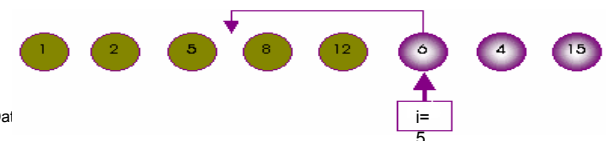


Data

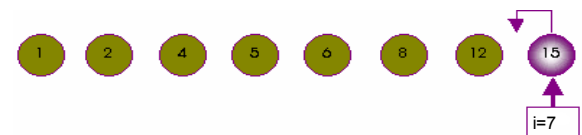
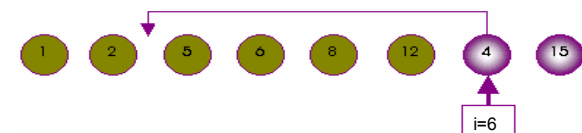
Chèn Trực Tiếp – Insertion Sort



Data



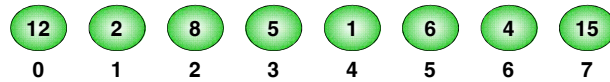
Chèn Trực Tiếp – Insertion Sort



Data Structures and Algorithms in Java

8

Minh Họa Thuật Toán Insertion Sort

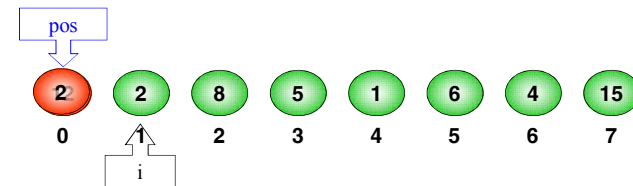


Data Structures and Algorithms in Java

9

Minh Họa Thuật Toán Insertion Sort

Insert $a[1]$ into (0,0)



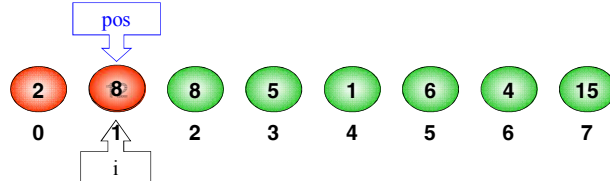
X

Data Structures and Algorithms in Java

10

Minh Họa Thuật Toán Insertion Sort

Insert $a[2]$ into (0, 1)



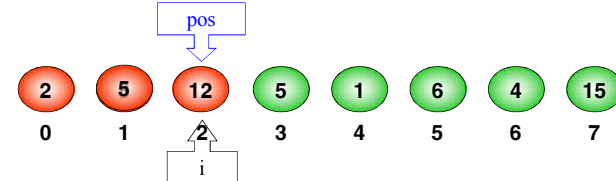
X

Data Structures and Algorithms in Java

11

Minh Họa Thuật Toán Insertion Sort

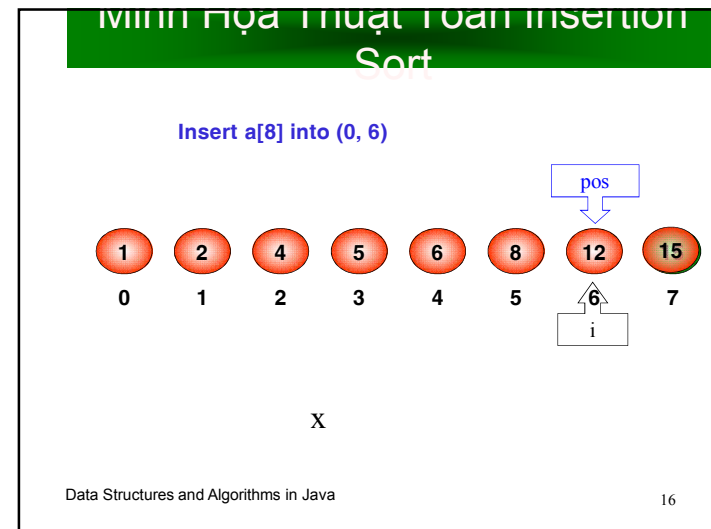
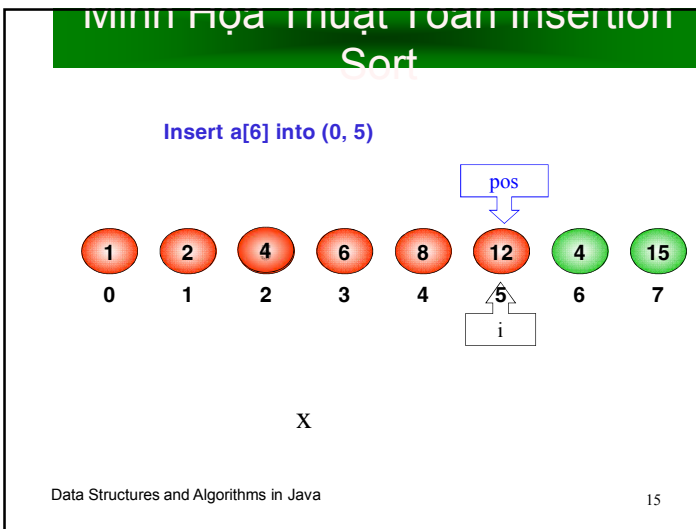
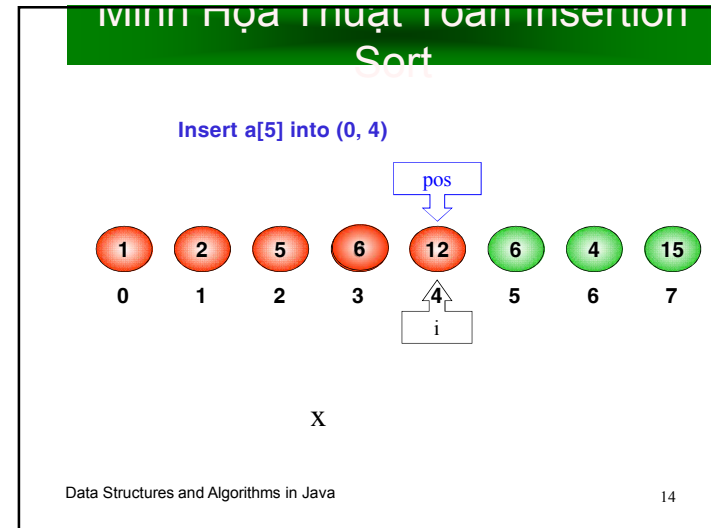
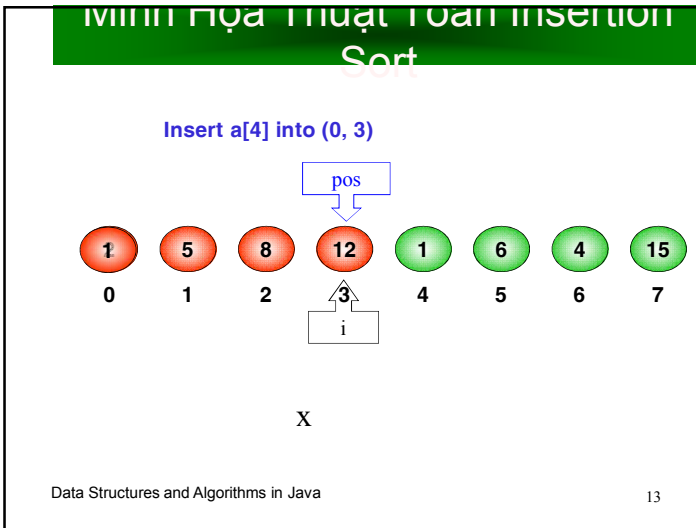
Insert $a[3]$ into (0, 2)



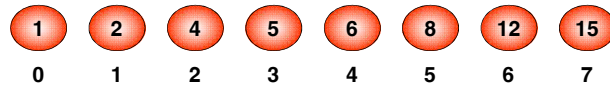
X

Data Structures and Algorithms in Java

12



Mô hình Thuật Toán Insertion Sort



Data Structures and Algorithms in Java

17

Elementary Sorting Algorithms

- An **insertion sort** starts by considering the two first elements of the array `data`, which are `data[0]` and `data[1]`
- Next, the third element, `data[2]`, is considered and inserted into its proper place

```
insertionsort(data[]) {
    for i = 1 to data.length-1
        tmp = data[i];
        move all elements data[j] greater than
        tmp by one position;
        place tmp in its proper position;_
}
```

Data Structures and Algorithms in Java

18

Cài Đặt Thuật Toán Chèn Trực Tiếp

```
void InsertionSort(int n, int []a )
{
    int pos, i;
    int x; //lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử.
    for(i=1 ; i<n ; i++) //đoạn a[0] đã sắp
    {
        x = a[i]; pos = i-1;
        // tìm vị trí chèn x
        while((pos >= 0)&&(a[pos] > x))
        {
            //kết hợp dời chỗ các phần tử sẽ đứng sau x trong dãy mới
            a[pos+1] = a[pos];
            pos--;
        }
        a[pos+1] = x; // chèn x vào dãy
    }
}
```

Data Structures and Algorithms in Java

19

Viết ra kết quả chạy từng bước

9	7	4	3	5
---	---	---	---	---

7	9	4	3	5
---	---	---	---	---

4	7	9	3	5
---	---	---	---	---

3	4	7	9	5
---	---	---	---	---

3	4	5	7	9
---	---	---	---	---

Data Structures and Algorithms in Java

20

Elementary Sorting Algorithms (continued)

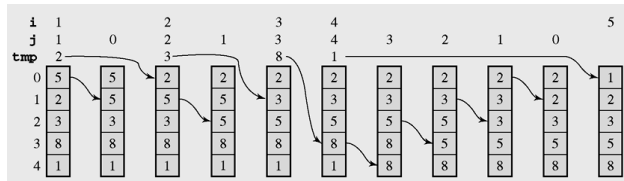


Figure 9-1 The array [5 2 3 8 1] sorted by insertion sort

Data Structures and Algorithms in Java

21

Selection Sort

- **Selection sort** is an attempt to localize the exchanges of array elements by finding a misplaced element first and putting it in its final place

```
selectionsort(data[])
for i = 0 to data.length-2
    select the smallest element among data[i], . . . ,
        data[data.length-1];
    swap it with data[i];
```

Data Structures and Algorithms in Java

22

Chọn Trực Tiếp – Selection Sort

➤ Ý tưởng:

- Chọn phần tử nhỏ nhất trong N phần tử trong dãy hiện hành ban đầu.
- Đưa phần tử này về vị trí đầu dãy hiện hành
- Xem dãy hiện hành chỉ còn N-1 phần tử của dãy hiện hành ban đầu
 - Bắt đầu từ vị trí thứ 2;
 - Lặp lại quá trình trên cho dãy hiện hành... đến khi dãy hiện hành chỉ còn 1 phần tử

Data Structures and Algorithms in Java

23

Các Bước Của Thuật Toán Chọn Trực Tiếp

- Bước 1: $i = 0$;
- Bước 2: Tìm phần tử **a[min]** nhỏ nhất trong dãy hiện hành từ $a[i]$ đến $a[N]$
- Bước 3: Đổi chỗ $a[\text{min}]$ và $a[i]$
- Bước 4: Nếu $i < N-1$ thì
 $i = i+1$; Lặp lại Bước 2;
 Ngược lại: Dừng.

Data Structures and Algorithms in Java

24

Cài Đặt Thuật Toán Chọn Trực Tiếp

```
void SelectionSort(int a[], int n)
{
    int vmin, i, j; // chỉ số phần tử nhỏ nhất trong dãy hiện hành
    for (i=0; i<n-1; i++) // chỉ số đầu tiên của dãy hiện hành
    {
        vmin = i;
        for (j = i+1; j < n; j++)
            if (a[j] < a[vmin])
                vmin = j; // lưu vị trí phần tử nhỏ nhất
        Swap(a[vmin], a[i]);
    }
}
```

25

Selection Sort (continued)

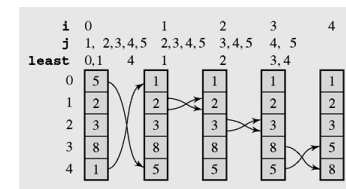


Figure 9-2 The array [5 2 3 8 1] sorted by selection sort

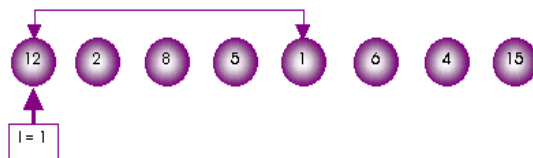
Data Structures and Algorithms in Java

26

Chọn Trực Tiếp – Selection Sort

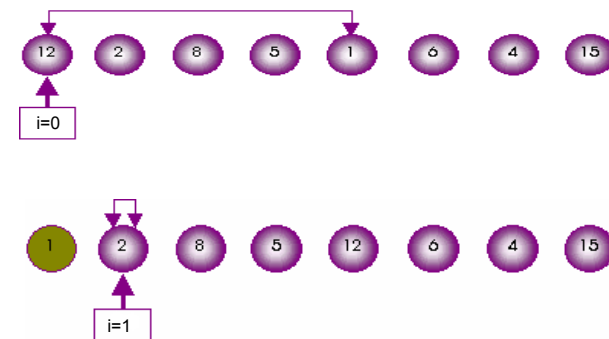
➤ Cho dãy số a:

12 2 8 5 1 6 4 15

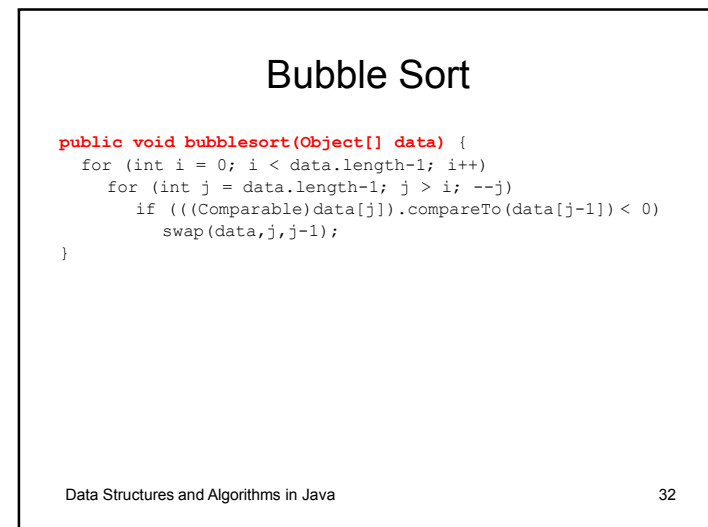
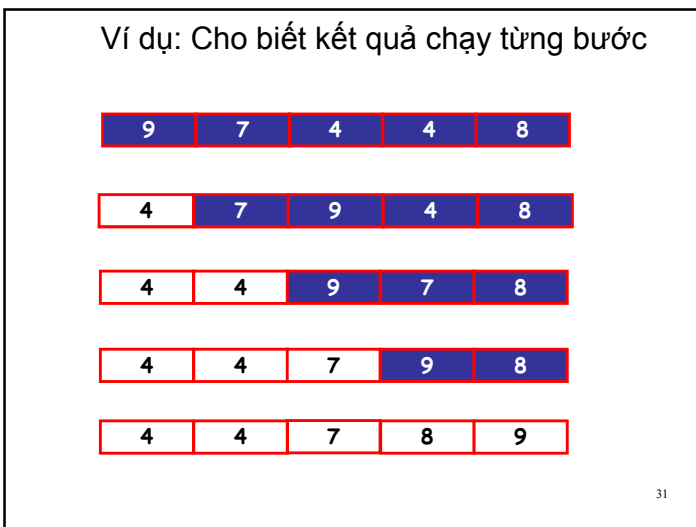
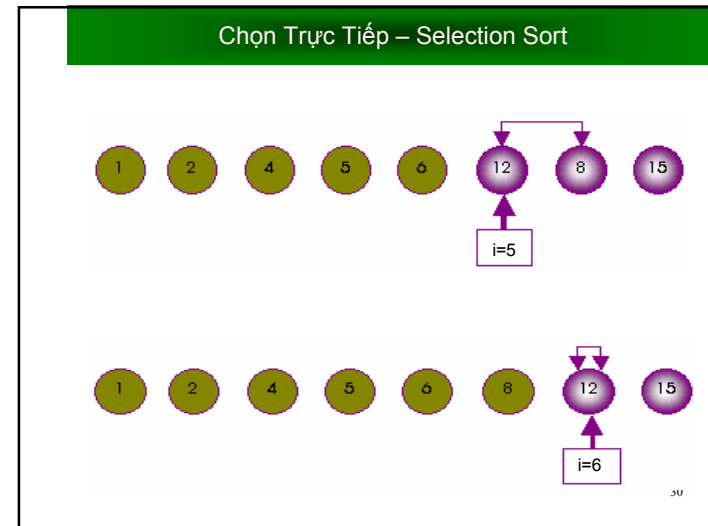
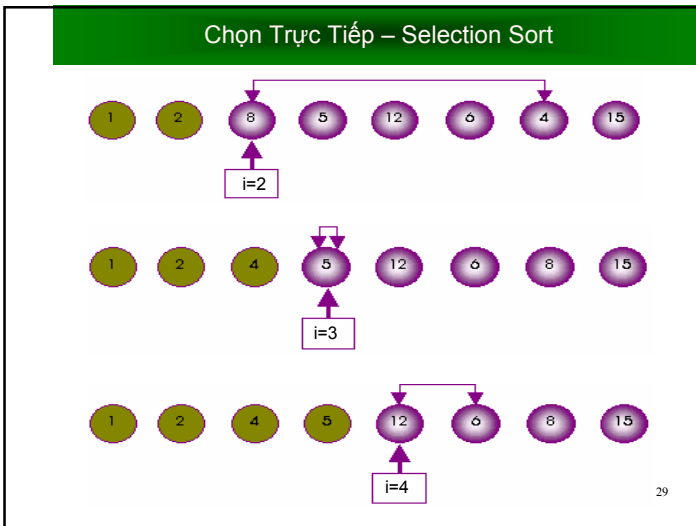


27

Chọn Trực Tiếp – Selection Sort



28



Nổi Bọt – Bubble Sort

Ý tưởng:

- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đúng đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i .
- Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.

Data Structures and Algorithms in Java

33

Nổi Bọt – Bubble Sort

- **Bước 1** : $i = 0$; // lần xử lý đầu tiên
- **Bước 2** : $j = N-1$; // Duyệt từ cuối dãy ngược về vị trí i
Trong khi ($j > i$) thực hiện:
Nếu $a[j] < a[j-1]$
Doicho($a[j], a[j-1]$);
 $j = j-1$;
- **Bước 3** : $i = i+1$; // lần xử lý kế tiếp
Nếu $i = N-1$: Hết dãy. Dừng
Ngược lại : Lặp lại Bước 2.

Data Structures and Algorithms in Java

34

Cài Đặt Thuật Toán nổi Bọt

```
void BubbleSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i < n-1 ; i++)
        for (j = n-1; j > i ; j --)
            if(a[j] < a[j-1]) // nếu sai vị trí thì đổi chỗ
                Swap(a[j], a[j-1]);
}
```

35

Bubble Sort

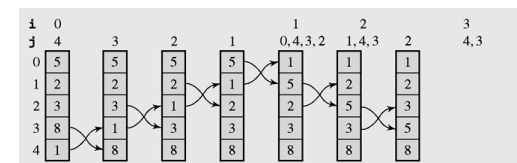


Figure 9-3 The array [5 2 3 8 1] sorted by bubble sort

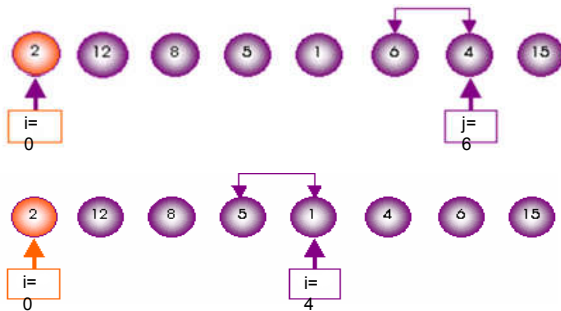
Data Structures and Algorithms in Java

36

Nổi Bọt – Bubble Sort

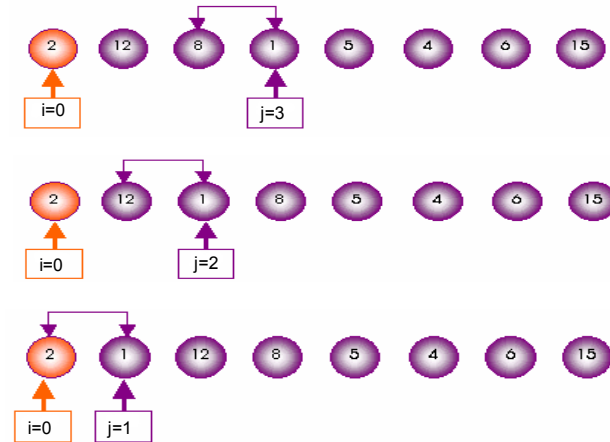
➤ Cho dãy số a:

2 12 8 5 1 6 4 15

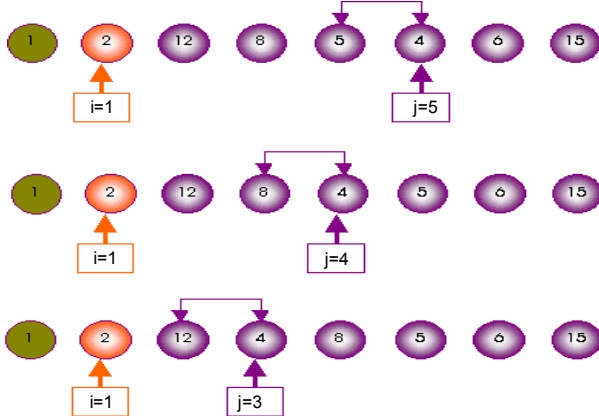


37

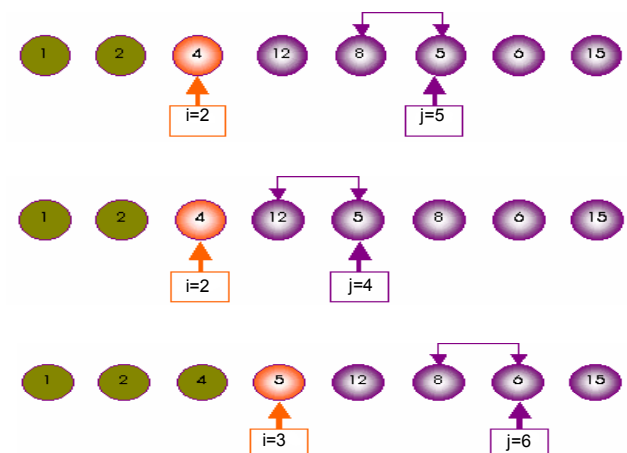
Nổi Bọt – Bubble Sort

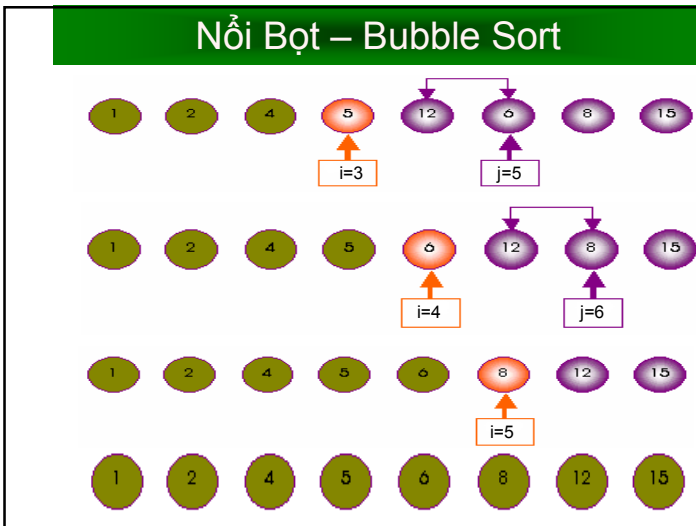


Nổi Bọt – Bubble Sort

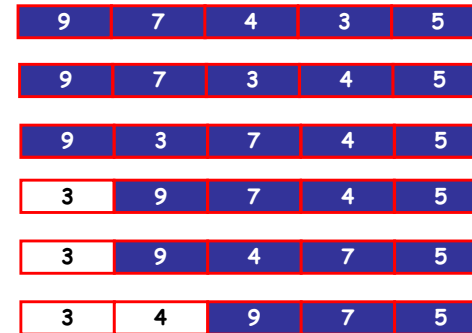


Nổi Bọt – Bubble Sort



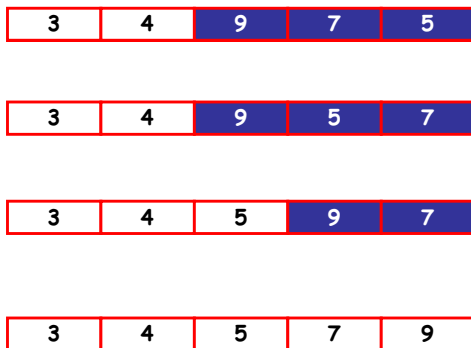


(1) Ví dụ : Ghi kết quả chạy từng bước



42

(2) Ví dụ : Ghi kết quả chạy từng bước



43

Decision Trees

- Every sorting algorithm can be expressed in terms of a binary tree in which the arcs carry the labels Y(es) or N(o)
- A **decision tree** is when nonterminal nodes of the tree contain conditions or queries for labels, and the leaves have all possible orderings of the array to which the algorithm is applied

Data Structures and Algorithms in Java

44

Decision Trees (continued)

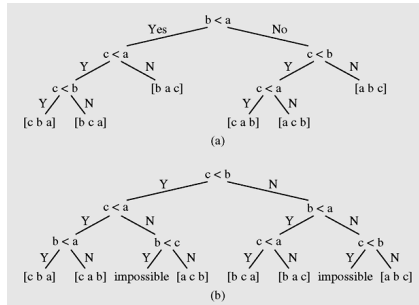


Figure 9-4 Decision trees for (a) insertion sort and (b) bubble sort as applied to the array [a b c]

Data Structures and Algorithms in Java

45

Decision Trees (continued)

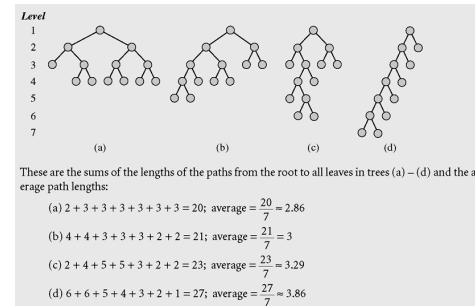


Figure 9-5 Examples of decision trees for an array of three elements

Data Structures and Algorithms in Java

46

Decision Trees (continued)

sort type	n	100	1,000	10,000
insertion	$\frac{n(n-1)}{4}$	2,475	249,750	24,997,500
selection, bubble	$\frac{n(n-1)}{2}$	4,950	499,500	49,995,000
expected	$n \lg n$	664	9,966	132,877

Figure 9-6 Number of comparisons performed by the simple sorting method and by an algorithm whose efficiency is estimated by the function $n \lg n$

Data Structures and Algorithms in Java

47

Shell Sort

- **Shell sort** divides the original array into subarrays, sorting them separately, and then dividing them again to sort the new subarrays until the whole array is sorted
- It is introduced by Donal L. Shell

Data Structures and Algorithms in Java

48

Ý tưởng

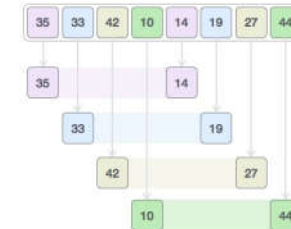
- Mang lại hiệu quả cao dựa trên giải thuật **sắp xếp chèn (Insertion Sort)**
- Giải thuật sắp xếp chèn trên các phần tử có khoảng cách xa nhau, sau đó sắp xếp các phần tử có khoảng cách hẹp hơn
- Khoảng cách này còn được gọi là **khoảng (interval)** – là số vị trí từ phần tử này tới phần tử khác. Khoảng này được tính dựa vào công thức Knuth như sau:

```
h = h * 3 + 1
trong đó:
h là Khoảng (interval) với giá trị ban đầu là 1
```

49

Ví dụ: [35 33 42 10 14 19 27 44]

- Cho $h = 4$

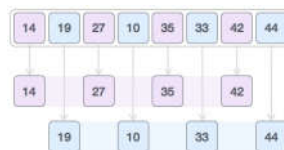


Kết quả: [14 19 27 10 35 33 42 44]

Data Structures and Algorithms in Java

50

$h = 2$

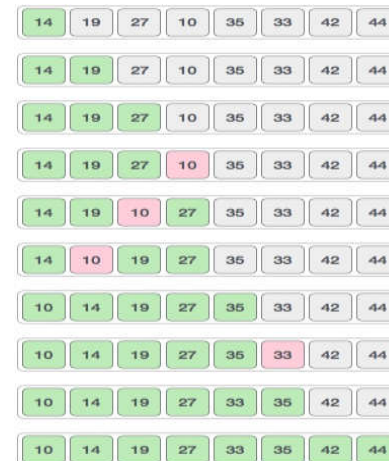


Kết quả: [14 19 27 10 35 33 42 44]

Data Structures and Algorithms in Java

51

$h = 1$ (ta thực hiện lại insertion sort)



52

Shell Sort (continued)

```

divide data into h subarrays;
for i = 1 to h; // h is the difference of indexes in initial array
    sort subarray datai using any simple method;
    // (insertion is usually used)
sort array data;

determine numbers h1 . . . hi of ways of dividing array data into subarrays;
for (h=hi; t > 1; t-- , h=hi)
    divide data into h subarrays;
    for i = 1 to h
        sort subarray datai;
sort array data

```

Data Structures and Algorithms in Java

53

Shell Sort (continued)

h=5	data before 5-sort	10	8	6	20	4	3	22	1	0	15	16
	Five subarrays before sorting	10	—	—	—	—	3	—	—	—	—	16
		8	—	—	—	—	—	22	—	—	—	—
			6	—	—	—	—	—	1	—	—	—
				20	—	—	—	—	—	0	—	—
					4	—	—	—	—	—	15	—
	Five subarrays after sorting	3	—	—	—	—	10	—	—	—	—	16
		8	—	—	—	—	—	22	—	—	—	—
			1	—	—	—	—	—	6	—	—	—
				0	—	—	—	—	—	20	—	—
					4	—	—	—	—	—	15	—

Figure 9-7 The array [10 8 6 20 4 3 22 1 0 15 16] sorted by Shell sort

Data Structures and Algorithms in Java

54

Shell Sort (continued)

h=3	data after 5-sort and before 3-sort	3	8	1	0	4	10	22	6	20	15	16
	Three subarrays before sorting	3	—	—	0	—	—	22	—	—	15	—
		8	—	—	4	—	—	—	6	—	—	16
			1	—	—	10	—	—	—	20	—	—
	Three subarrays after sorting	0	—	—	3	—	—	15	—	—	22	—
		4	—	—	6	—	—	8	—	—	—	16
			1	—	—	10	—	—	—	20	—	—
h=1	data after 3-sort and before 1-sort	0	4	1	3	6	10	15	8	20	22	16
	data after 1-sort	0	1	3	4	6	8	10	15	16	20	22

Figure 9-7 The array [10 8 6 20 4 3 22 1 0 15 16] sorted by Shell sort (continued)

Data Structures and Algorithms in Java

55

Shell Sort (continued)

h is the difference of two indexes in subarray

```

void Shellsort (Object[] data) {
    int i, j, k, h, hCnt, increments[] = new int[20];
    // create an appropriate number of increments h
    for (h = 1, i = 0; h < data.length; i++) {
        increments[i] = h;
        h = 3*h + 1;
    }
    // loop on the number of different increments h
    for (i = 0; i >= 0; i--) {
        h = increments[i];
        // loop on the number of subarrays h-sorted in ith pass
        for (hCnt = h; hCnt < 2*h; hCnt++) {
            // insertion sort for subarray containing every hth element
            // of array data
            for (j = hCnt; j < data.length; j++) {
                Comparable tmp = (Comparable) data[j];
                k = j;
                while ((k-h) >= 0 && tmp.compareTo(data[k-h]) < 0) {
                    data[k] = data[k-h];
                    k -= h;
                }
                data[k] = tmp;
                j += h;
            }
        }
    }
}

```

i is the increments .length

i-1 is the last index of the array increments

Figure 9-8 Implementation of Shell sort

Data Structures and Algorithms in Java

56

Heap Sort

- A **heap** is a binary tree with the following two properties:
 - The value of each node is not less than the values stored in each of its children.
 - The tree is perfectly balanced and the leaves in the last level are all in the leftmost positions.
- Elements in a heap are not perfectly ordered

Data Structures and Algorithms in Java

57

Heap Sort (continued)

- The largest element is in the root node and that, for each other node, all its descendants are not greater than the element in this node

```
heapsort(data[])
  transform data into a heap;
  for i = data.length-1 downto 2 (đi ngược)
    swap the root with the element in position i;
    restore the heap property for the tree data[0], . . . , data[i-1];
```

Data Structures and Algorithms in Java

58

Tóm tắt kỹ thuật Heap-Sort

- Bước 1: Chuyển mảng về dạng biểu diễn Max-Heap
- Bước 2: Sử dụng Heap-Sort để sắp xếp

Data Structures and Algorithms in Java

59

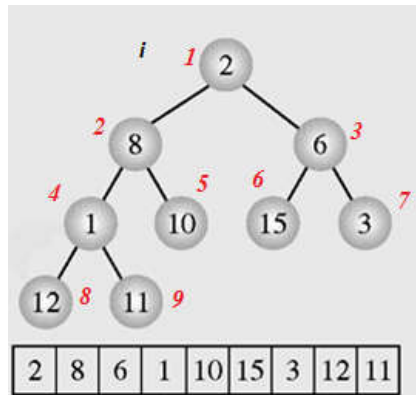
Bước 1: Xây dựng Heap từ mảng

- **Khởi đầu:** $i = \lfloor n/2 \rfloor$, mỗi nút $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$ là một lá, chúng là gốc của một max-heap
- **Duy trì:** MAX-HEAPIFY(A, i) đảm bảo nút i và các con của nó là các gốc của các max-heap, bất biến vòng lặp thỏa khi i giảm và trở về đầu vòng lặp
- **Kết thúc:** Khi $i = 0$, mỗi nút $1, 2, \dots, n$ là gốc của một max-heap

Data Structures and Algorithms in Java

60

Initial Tree



Data Structures and Algorithms in Java

61

Build Heap (continued)

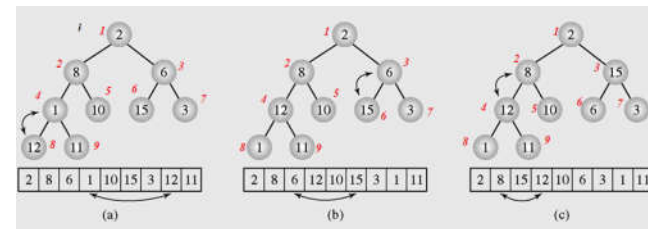


Figure 9-9 Transforming the array [2 8 6 1 10 15 3 12 11] into a heap

Data Structures and Algorithms in Java

62

Build Heap(continued)

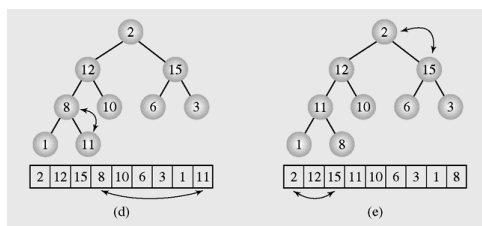


Figure 9-9 Transforming the array [2 8 6 1 10 15 3 12 11] into a heap (continued)

Data Structures and Algorithms in Java

63

Build Heap(continued)

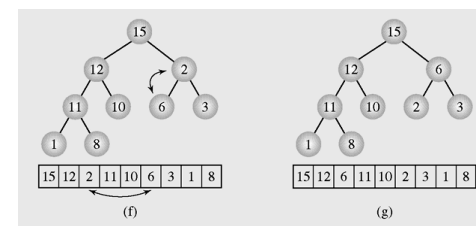


Figure 9-9 Transforming the array [2 8 6 1 10 15 3 12 11] into a heap (continued)

Data Structures and Algorithms in Java

64

Bước 2: Heap Sort

Heapsort sử dụng BUILD-MAX-HEAP để xây dựng một max-heap trên mảng input $A[1..n]$

Hoán đổi giá trị $A[1]$ với $A[n]$

Loại nút n ra khỏi heap và chuyển $A[1..(n-1)]$ thành một max-heap

Lặp lại các bước trên cho đến khi heap chỉ còn một phần tử

Data Structures and Algorithms in Java

65

Heap Sort (continued)

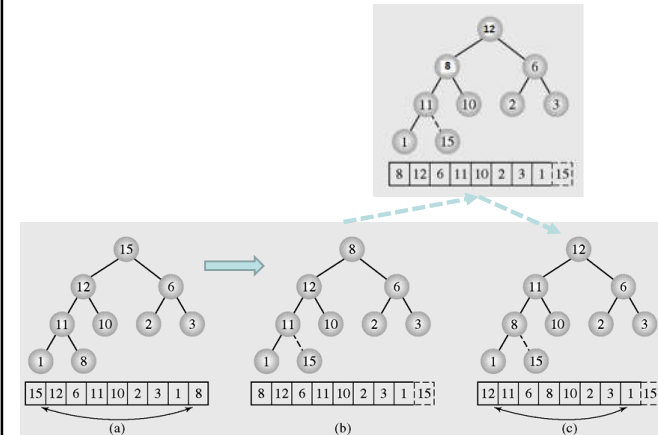


Figure 9-10 Execution of heap sort on the array $[15, 12, 6, 11, 10, 2, 3, 1, 8]$, which is the heap constructed in Figure 9.9

Heap Sort (continued)

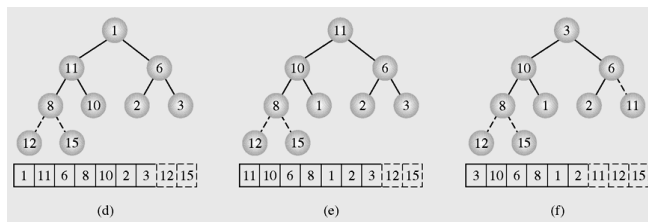


Figure 9-10 Execution of heap sort on the array $[15, 12, 6, 11, 10, 2, 3, 1, 8]$, which is the heap constructed in Figure 9.9 (continued)

Data Structures and Algorithms in Java

67

Heap Sort (continued)

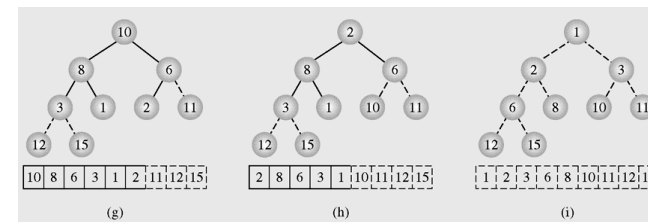


Figure 9-10 Execution of heap sort on the array $[15, 12, 6, 11, 10, 2, 3, 1, 8]$, which is the heap constructed in Figure 9.9 (continued)

Data Structures and Algorithms in Java

68

Bài tập 1.a

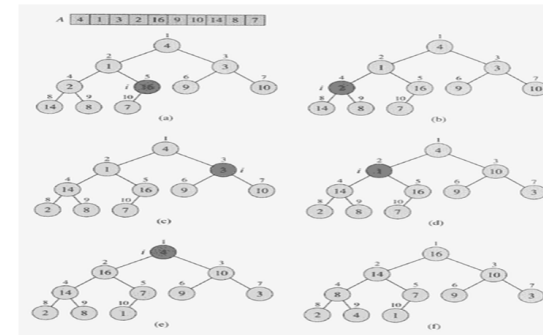
- Hãy xây dựng mảng sau thành dạng Heap Tree theo từng bước. Chú ý: Phải ghi rõ vị trí của chỉ số index (trong) mảng, ứng với các nút



Data Structures and Algorithms in Java

69

Đáp án



Data Structures and Algorithms in Java

70

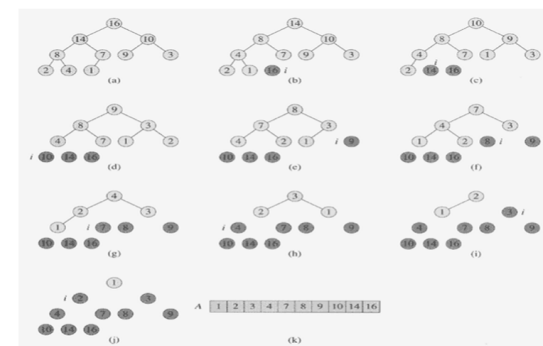
Bài tập 1.b

- Tiến hành thực hiện các bước Heap-Sort và cho biết cây và mảng lúc remove được 3 nodes

Data Structures and Algorithms in Java

71

Đáp án



Data Structures and Algorithms in Java

72

Bài tập 2.a

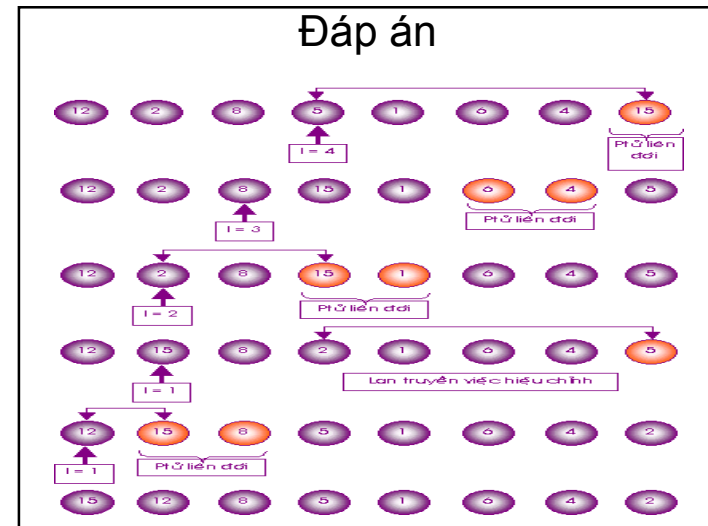
- Hãy xây dựng mảng sau thành dạng Heap Tree theo từng bước. Chú ý: Phải ghi rõ vị trí của chỉ số index (trong) mảng, ứng với các nút

[12, 2, 8, 5, 1, 6, 4, 15]

Data Structures and Algorithms in Java

73

Đáp án



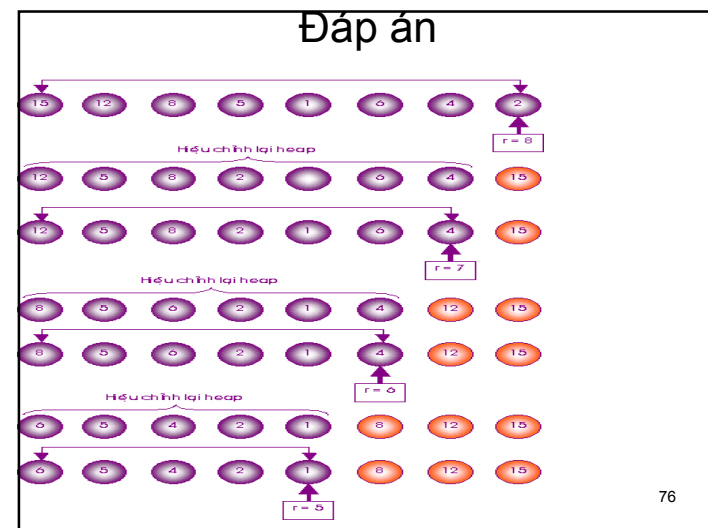
Bài tập 2.b

- Tiến hành thực hiện các bước Heap-Sort

Data Structures and Algorithms in Java

75

Đáp án



76

Bài tập

- Hãy xây dựng mảng sau thành dạng Heap Tree theo từng bước. Chú ý: Phải ghi rõ vị trí của chỉ số index (trong) mảng, ứng với các nút

A = [5 2 6 4 8 1]

- Hãy xây dựng từng bước giải thuật Heap-Sort cho mảng trên

Data Structures and Algorithms in Java

77

Heap Sort (continued)

```
void moveDown(T[] data, int first, int last) { // array[first..last] → heap
    int largest = 2*first + 1;
    while (largest <= last) {
        // first has two children (at 2*first+1 and 2*first+2);
        if (largest < last && data[largest].compareTo(data[largest+1]) < 0) largest++;
        if (data[first].compareTo(data[largest]) < 0) {
            swap(data, first, largest); // if necessary, swap values
            first = largest; // and move down;
            largest = 2*first + 1;
        }
        else largest = last + 1; // to exit the loop: the heap property isn't violated by data[first];
    }
}

void heapsort(T[] data) {
    for (int i = data.length/2 - 1; i >= 0; --i) moveDown(data, i, data.length-1); // array → heap
    for (int i = data.length-1; i >= 1; --i) {
        swap(data, 0, i); // move the largest value to the end of the array
        moveDown(data, 0, i-1); // array[0..i-1] → heap
    }
}
```

Data Structures and Algorithms in Java

78

Quick Sort

➤ Ý tưởng:

- Giải thuật QuickSort sắp xếp dãy a_1, a_2, \dots, a_N dựa trên việc phân hoạch dãy ban đầu thành 3 phần :
 - Phần 1: Gồm các phần tử có giá trị bé hơn x
 - Phần 2: Gồm các phần tử có giá trị bằng x
 - Phần 3: Gồm các phần tử có giá trị lớn hơn x
- với x là giá trị của một phần tử tùy ý trong dãy ban đầu.

79

Quick Sort - Ý Tưởng

- Sau khi thực hiện phân hoạch, dãy ban đầu được phân thành 3 đoạn:
 - 1. $a_k \leq x$, với $k = 0 \dots j$
 - 2. $a_k = x$, với $k = j+1 \dots i-1$
 - 3. $a_k \geq x$, với $k = i \dots N-1$

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

80

Quick Sort – Ý Tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 chỉ có 1 phần tử : đã có thứ tự
→ khi đó dãy con ban đầu đã được sắp.

81

Quick Sort – Ý Tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các đoạn 1, 3 được sắp.
- Để sắp xếp các đoạn 1 và 3, ta lần lượt tiến hành việc phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu vừa trình bày ...

82

Giải Thuật Quick Sort

- **Bước 1:** Nếu $left \geq right$ //dãy có ít hơn 2 phần tử
Kết thúc; //dãy đã được sắp xếp
- **Bước 2:** Phân hoạch dãy $a_{left} \dots a_{right}$ thành các đoạn: $a_{left} \dots a_j, a_{j+1} \dots a_{i-1}, a_i \dots a_{right}$
Đoạn 1 $\leq x$
Đoạn 2: $a_{j+1} \dots a_{i-1} = x$
Đoạn 3: $a_i \dots a_{right} \geq x$
- **Bước 3:** Sắp xếp đoạn 1: $a_{left} \dots a_j$
- **Bước 4:** Sắp xếp đoạn 3: $a_i \dots a_{right}$

83

Giải Thuật Quick Sort

- **Bước 1:** Chọn tùy ý một phần tử $a[k]$ trong dãy là giá trị mốc ($1 \leq k \leq r$):
 $x = a[k]; i = l; j = r;$
- **Bước 2:** Phát hiện và hiệu chỉnh cặp phần tử $a[i], a[j]$ nằm sai chỗ :
 - **Bước 2a:** Trong khi $(a[i] < x) i++;$
 - **Bước 2b:** Trong khi $(a[j] > x) j--;$
 - **Bước 2c:** Nếu $i < j$ Đổi chỗ $(a[i], a[j]);$
- **Bước 3:** Nếu $i < j$: Lặp lại Bước 2.
Ngược lại: Dừng

84

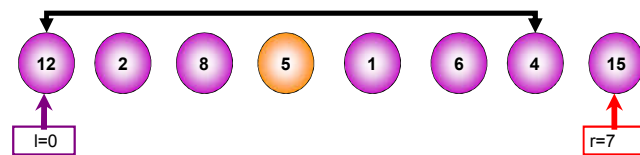
Quick Sort – Ví Dụ

➤ Cho dãy số a:

12 28 5 1 6 4 15

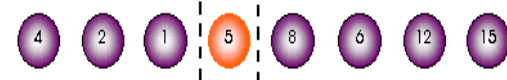
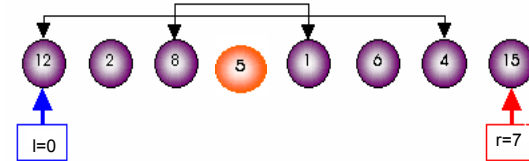
Phân hoạch đoạn $l=0, r=7$:

$x = a[3] = 5$



85

Quick Sort – Ví Dụ

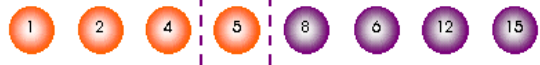
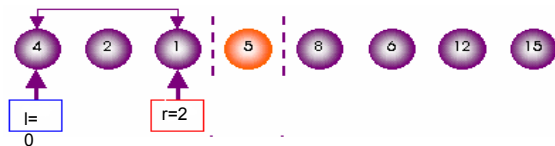


86

Quick Sort – Ví Dụ

➤ Phân hoạch đoạn $l=0, r=2$:

$x = a[2] = 2$

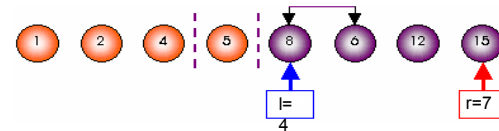


87

Quick Sort – Ví Dụ

➤ Phân hoạch đoạn $l=4, r=7$:

$x = a[5] = 6$



88

➤ Phân hoạch đoạn $l = 6, r = 7$:
 $x = a[6] = 6$

89

Quick Sort – Ví Dụ

Phân hoạch dãy

90

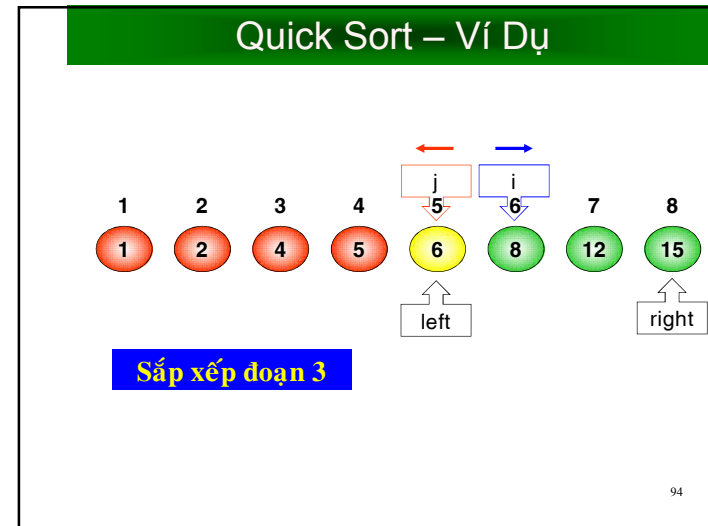
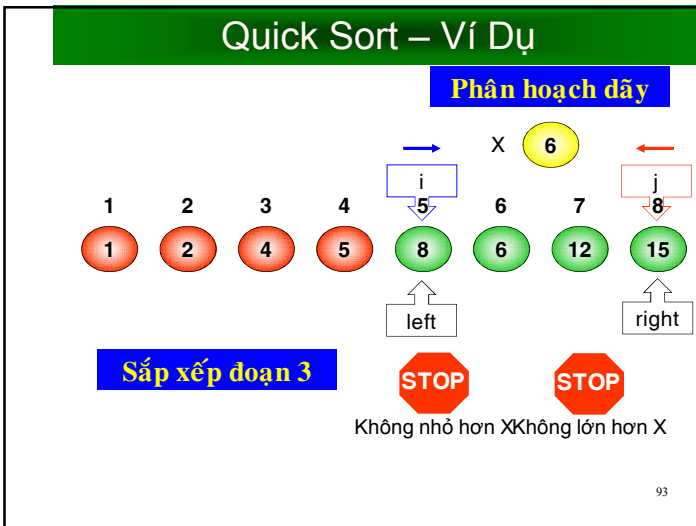
Quick Sort – Ví Dụ

Phân hoạch dãy

91

Quick Sort – Ví Dụ

92



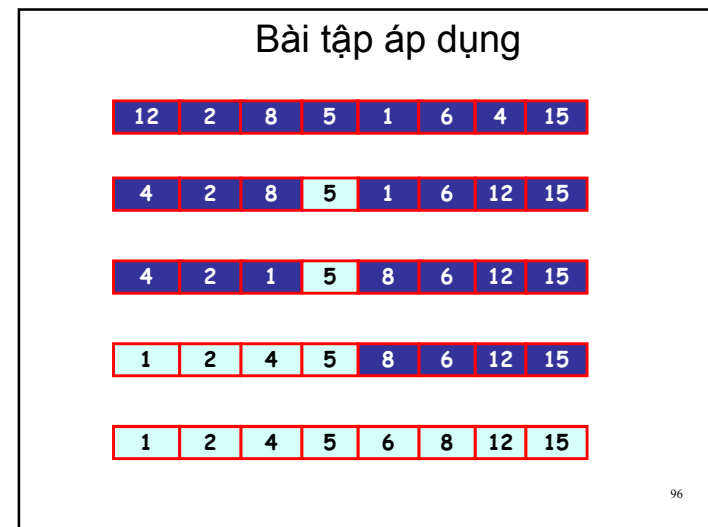
Quick Sort

```

void QuickSort(int a[], int left, int right)
{
    int i, j, x;
    x = a[(left+right)/2];
    i = left; j = right;
    while(i < j)
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Doicho(a[i], a[j]);
            i++; j--;
        }
    }
    if(left < j)
        QuickSort(a, left, j);
    if(i < right)
        QuickSort(a, i, right);
}

```

95



Quick Sort

- It is introduced by C. A. R. Hoare

`quickSort (array[])`

if `array.length > 1`

choose `bound`; // value for partition array into 2 subarrays

while there are elements left in array

include element either in subarray1 = { `el`: `el` <= `bound` }

or in subarray2 = { `el`: `el` >= `bound` }

`quickSort (subarray1);`

`quickSort (subarray2);`

Quicksort

```

87 //QUICKSORT
88 public static void quickSort(Comparable[] a, int first, int last)
89 {
90     int lower = first + 1, upper = last;
91     swap(a, first, (first + last) / 2);
92     // giá đầu tiên là giá giữa ban đầu -> chọn làm giá đứng ở trục
93     Comparable bound = a[first];
94     // Đếm vị trí nhỏ và lớn, vị trí lớn về cuối
95     // partition mảng thành 2 phần: vị trí nhỏ ở trước, lớn ở sau
96     while (lower <= upper)
97     {
98         while (bound.compareTo(a[lower]) > 0) lower++;
99         while (bound.compareTo(a[upper]) < 0) upper--;
100         if (lower < upper) swap(a, lower++, upper--);
101         else lower++;
102     }
103     swap(a, upper, first);
104     if (first < upper-1) quickSort(a, first, upper-1);
105     if (upper+1 < last) quickSort(a, upper+1, last);
106 }
107
108 public static void quickSort(Comparable[] a)
109 {
110     if (a.length < 2) return;
111     int max = 0;
112     // find the largest element and put it at the end of data;
113     for (int i = 1; i < a.length; i++)
114         if (a[max].compareTo(a[i]) < 0) max = i;
115     swap(a, a.length-1, max); // largest el is now in its final position
116     quickSort(a, 0, a.length-2);
117 }

```

Figure 9-11 Implementation of quicksort

Quicksort (continued)(Bỏ slide này)

```

return;
int max = 0;
// find the largest element and put it at the end of data;
for (int i = 1; i < data.length; i++)
    if (((Comparable)data[max]).compareTo(data[i]) < 0)
        max = i;
swap(data, data.length-1, max); // largest el is now in its
quickSort(data, 0, data.length-2); // final position;
}

```

Figure 9-11 Implementation of quicksort (continued)

Quicksort (continued)

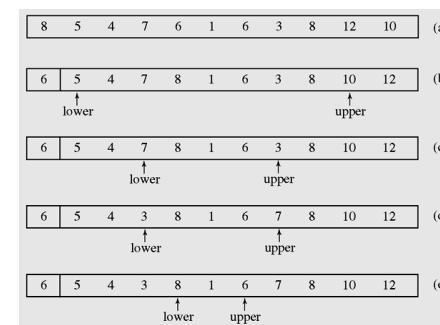


Figure 9-12 Partitioning the array [8 5 4 7 6 1 6 3 8 12 10] with quicksort ()

Quicksort (continued)

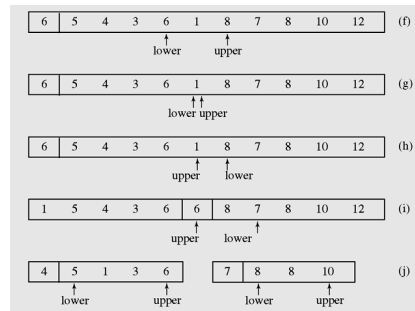


Figure 9-12 Partitioning the array [8 5 4 7 6 1 6 3 8 12 10] with quicksort () (continued)

Data Structures and Algorithms in Java

101

Quicksort (continued)

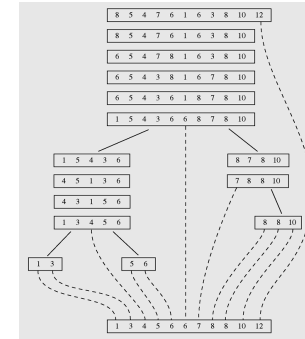


Figure 9-13 Sorting the array [8 5 4 7 6 1 6 3 8 12 10] with quicksort ()

Data Structures and Algorithms in Java

102

Kết luận

Trường hợp	Độ phức tạp
Tốt nhất	$n \cdot \log(n)$
Xấu nhất	n^2

Data Structures and Algorithms in Java

103

Bài tập (*)

- Cho lớp Student: code, name, tuoi
- Xây dựng thuật toán QuickSortForStudent để sắp xếp danh sách sinh viên tăng dần theo điểm số

Data Structures and Algorithms in Java

104

Mergesort

- **Mergesort** makes partitioning as simple as possible and concentrates on merging sorted halves of an array into one sorted array
- It was one of the first sorting algorithms used on a computer and was developed by John von Neumann

```
mergesort(data)
    if data have at least two elements
        mergesort(left half of data);
        mergesort(right half of data);
        merge(both halves into a sorted list);
```

Data Structures and Algorithms in Java

105

Ý tưởng

Chia để trị (Divide and conquer): Chia bài toán lớn thành những bài toán nhỏ hơn. Giải quyết những bài toán nhỏ sau đó gộp lại để được lời giải cho bài toán lớn.

Ý tưởng merge sort: Để sắp xếp một mảng $A[start...end]$, ta chia mảng A thành 2 mảng con $A1$ và $A2$. Sắp xếp $A1$ và $A2$, sau đó hòa nhập chúng thành một để được mảng A đã sắp xếp.

Mô tả thuật toán:

Bước 1:

- $Mid = (start + end) / 2$
- Sắp xếp hai nửa mảng $A[start...mid]$ và $A[(mid + 1)...end]$. Việc sắp xếp hai nửa mảng được thực hiện bằng cách gọi đệ quy thủ tục sắp xếp hòa nhập

Bước 2: Hòa nhập hai nửa mảng $A[start...mid]$ và $A[(mid + 1)...end]$ để thu được mảng A trong đó các phần tử đã được sắp xếp.

Ví dụ:

$A = (7, 3, 9, 1)$

Sắp xếp hai nửa mảng: $A = (3, 7, 1, 9)$

Hòa nhập hai nửa mảng: $A = (1, 3, 7, 9)$

106

Cài đặt

```
void MergeSort( Item A[ ], int start, int end) {
    if (start < end) {
        int mid = (start + end)/2;
        MergeSort ( A, start, mid );
        MergeSort ( A, mid+1, end);
        Merge ( A, start, mid, end);
    }
}
```

Data Structures and Algorithms in Java

107

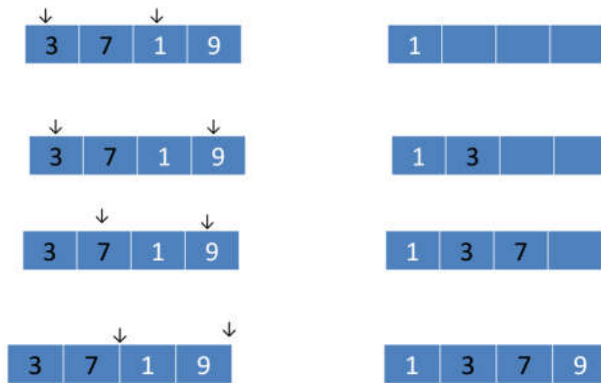
Hòa nhập 2 mảng? (Merge)

- + Đối với mỗi mảng con chúng ta có một con trỏ trỏ tới phần tử đầu tiên
- + Đặt phần tử nhỏ hơn của các phần tử đang xét ở hai mảng vào mảng mới
- + Di chuyển con trỏ của các mảng tới vị trí thích hợp
- + Lặp lại các bước thực hiện trên cho tới khi mảng mới chứa hết các phần tử của hai mảng con

Data Structures and Algorithms in Java

108

Ví dụ - hòa nhập 2 mảng



Ví dụ: Mergesort (continued)

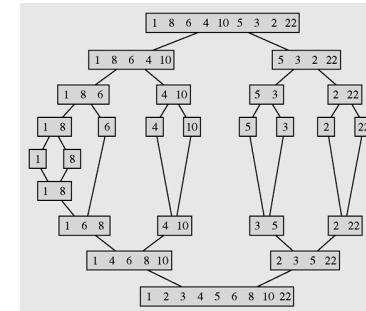
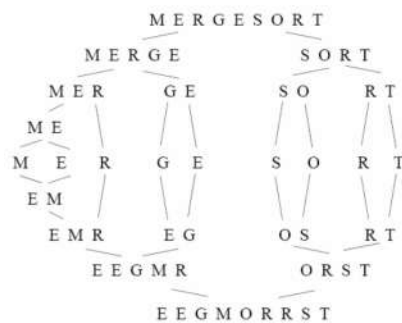


Figure 9-14 The array [1 8 6 4 10 5 3 2 22] sorted by mergesort

Data Structures and Algorithms in Java

110

Ví dụ



Data Structures and Algorithms in Java

111

Bài tập

- Hãy vẽ cây hòa nhập (Merge Sort)

7 2 9 4 3 8 6 1

Data Structures and Algorithms in Java

112

Bài tập

- Hãy vẽ cây hòa nhập (Merge Sort)

C D A B G H I J K A B F E

Data Structures and Algorithms in Java

113

Radix Sort

- A technique to sort integers by proceeding right to left
- A technique that looks at each number as a string of bits so that all integers are of equal length

```
radixsort()
for d = 1 to the position of the leftmost digit of longest number
    distribute all numbers among piles 0 through 9 according to the dth digit;
    put all integers on one list;
```

Data Structures and Algorithms in Java

114

Ý tưởng

Bước 1 : // k cho biết chữ số dùng để phân loại hiện hành

k = 0; // k = 0: hàng đơn vị; k = 1: hàng chục; |

Bước 2 : //Tạo các lô chứa các loại phần tử khác nhau

Khởi tạo 10 lô B0, B1, .., B9 rỗng;

Bước 3 :

For i = 1 .. n do

Đặt ai vào lô Bt với t = chữ số thứ k của ai;

Bước 4 :

Nối B0, B1, .., B9 lại (theo đúng trình tự) thành a.

Bước 5 :

k = k+1;

Nếu k < m thì trở lại bước 2.

Ngược lại: Dừng

115

Ví dụ 1

- Cho dãy số

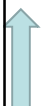
A = [701 1725 999 9170 3252 4518 7009 1424
428 1239 8425 7013]

Data Structures and Algorithms in Java

116

Bước 1: Phân lô hàng đơn vị

A = [701 1725 999 9170 3252 4518 7009 1424 428 1239 8425 7013]



12	0701										
11	1725										
10	0999										
9	9170										
8	3252										
7	4518										
6	7009										
5	1424										
4	0428										
3	1239									0999	
2	8425					1725			4518	7009	
1	7013	9170	0701	3252	7013	1424	8425		0428	1239	
CS	A	0	1	2	3	4	5	6	7	8	9

117

Bước 2: Phân lô hàng chục

A = [701 1725 999 9170 3252 4518 7009 1424 428 1239 8425 7013]

12	0999										
11	7009										
10	1239										
9	4518										
8	0428										
7	1725										
6	8425										
5	1424										
4	7013			0428							
3	3252			1725							
2	0701	7009	4518	8425							
1	9170	0701	7013	1424	1239		3252		9170		0999
CS	A	0	1	2	3	4	5	6	7	8	9

118

Bước 1: Phân lô hàng trăm

A = [701 1725 999 9170 3252 4518 7009 1424 428 1239 8425 7013]

12	0999										
11	9170										
10	3252										
9	1239										
8	0428										
7	1725										
6	8425										
5	1424										
4	4518										
3	7013				0428						
2	7009	7013		3252	8425		1725				
1	0701	7009	9170	1239	1424	4518	0701		0999		
CS	A	0	1	2	3	4	5	6	7	8	9

119

Bước 1: Phân lô hàng nghìn

A = [701 1725 999 9170 3252 4518 7009 1424 428 1239 8425 7013]

12	0999										
11	1725										
10	0701										
9	4518										
8	0428										
7	8425										
6	1424										
5	3252										
4	1239										
3	9170	0999	1725								
2	7013	0701	1424						7013		
1	7009	0428	1239		3252	4518			7009	8425	9170
CS	A	0	1	2	3	4	5	6	7	8	9

120

Radix Sort (Ví dụ 2)

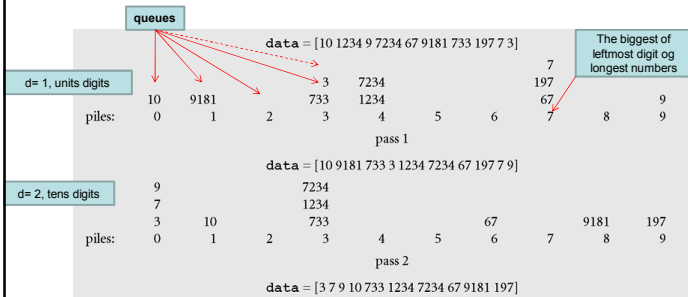


Figure 9-15 Sorting the list 10, 1234, 9, 7234, 67, 9181, 733, 197, 7, 3 with radix sort

Data Structures and Algorithms in Java

121

Radix Sort (Ví dụ 2)

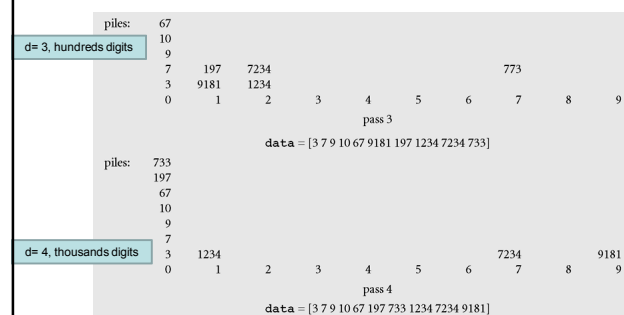


Figure 9-15 Sorting the list 10, 1234, 9, 7234, 67, 9181, 733, 197, 7, 3 with radix sort (continued)

Data Structures and Algorithms in Java

122

Radix Sort (continued)

```
private final int radix = 10;
private final int digits = 10;
public void radixsort(int[] data) {
    int d, j, k, factor;
    Queue<Integer>[] queues = new Queue[radix]; // array of queues
    for (d = 0; d < radix; d++) queues[d] = new Queue<Integer>();
    for (d = 1, factor = 1; d <= digits; factor *= radix, d++) {
        // Move elements in array to suitable queues base on the dth digits
        for (j = 0; j < data.length; j++) queues[(data[j] / factor) % radix].enqueue(data[j]);
        // Move values in queues to the array
        for (j = k = 0; j < radix; j++)
            while (!queues[j].isEmpty()) data[k++] = queues[j].dequeue();
    }
}
```

Evaluation

- Time for divide and division modulo are not appropriate.
- More additional space for queues
- Move elements: $\text{digits} \times 2 \times O(n)$

Data Structures and Algorithms in Java

123

Bit Radix Sort – Improvement

```
private final int bits = 31;
public void bitRadixsort(int[] data, int b) {
    int pow2b = 1 << b; // 010000000 00000000 00000000 00000000
    int i, j, k, pos = 0, mask = pow2b - 1;
    int last = (bits % b == 0) ? (bits / b) : (bits / b + 1);
    Queue<Integer>[] queues = new Queue[pow2b];
    for (i = 0; i < pow2b; i++) queues[i] = new Queue<Integer>();
    // Move elements in array to suitable queues base on the dth digits
    for (i = 0; i < last; i++) {
        for (j = 0; j < data.length; j++)
            queues[(data[j] & mask) >> pos].enqueue(data[j]);
        mask <<= b;
        pos = pos + b;
        for (j = k = 0; j < pow2b; j++) // Move values in queues to the array
            while (!queues[j].isEmpty()) data[k++] = queues[j].dequeue();
    }
}
```

Data Structures and Algorithms in Java

124

Radix Sort (continued)

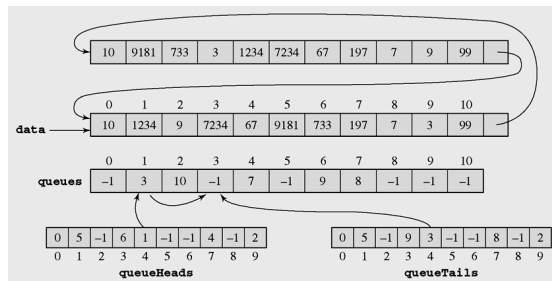


Figure 9-16 An implementation of radix sort

Data Structures and Algorithms in Java

125

Sorting in `java.util`

- Java provides two sets of versions of sorting methods: one for arrays and one for lists
- The utility class `Arrays` includes a method for:
 - Searching arrays for elements with binary search
 - Filling arrays with a particular value
 - Converting an array into a list, and sorting methods
- The sorting methods are provided for arrays with elements of all elementary types except Boolean

Data Structures and Algorithms in Java

126

Sorting in `java.util` (continued)

- For each type of sorting method there are two versions:
 - One for sorting an entire array
 - One for sorting a subarray

```
public static void sort(int[] a);
public static void sort(int[] a, int first, int last);
```

Data Structures and Algorithms in Java

127

Sorting in `java.util` (continued)

```
import java.io.*;
import java.util.*;

class Person implements Comparable {
    private String name;
    public int age;
    public Person(String s, int i) {
        name = s; age = i;
    }
    public Person() {
        this("", 0);
    }
    public String toString() {
        return "(" + name + ", " + age + ")";
    }
    public int compareTo(Object p) {
        return name.compareTo(((Person)p).name);
    }
}
```

Figure 9-17 Demonstration of sorting functions

Data Structures and Algorithms in Java

128

Sorting in java.util (continued)



```
class PersonComparator implements Comparator {
    public int compare(Object ob1, Object ob2) {
        if (ob1 == ob2)
            return 0;
        else if (ob1 == null)
            return -1;
        else if (ob2 == null)
            return 1;
        return ((Person)ob1).age - ((Person)ob2).age;
    }
}

class testSorts {
    public static void main(String[] ar) {
        int[] intArr1 = {4, 6, 7, 4, 2};
        int[] intArr2 = {4, 6, 7, 4, 2};
        char[] charArr = {'a', 'n', 'd', 'v', 'a'};
        Arrays.sort(intArr1); // intArr1 = [2, 4, 4, 6, 7]
        for (int i = 0; i < intArr1.length; i++)
            System.out.print(intArr1[i] + " ");
        System.out.println();

        class testSorts {
            public static void main(String[] ar) {
                int[] intArr1 = {4, 6, 7, 4, 2};
                int[] intArr2 = {4, 6, 7, 4, 2};
                char[] charArr = {'a', 'n', 'd', 'v', 'a'};
                Arrays.sort(intArr1); // intArr1 = [2, 4, 4, 6, 7]
                for (int i = 0; i < intArr1.length; i++)
                    System.out.print(intArr1[i] + " ");
                System.out.println();
            }
        }
    }
}
```

Figure 9-17 Demonstration of sorting functions (continued)

Data Structures and Algorithms in Java

129

Sorting in java.util (continued)

```
Arrays.sort(intArr2, 1, intArr2.length-2); // intArr2 = [4, 6, 7, 4, 2]
Arrays.sort(charArr); // charArr = ['a', 'a', 'd', 'n', 'v']
Person[] persons1 = {new Person("Tom", 50), new Person("Lili", 29),
    new Person("Jeff", 44)};
Person[] persons2 = {new Person("Tom", 50), new Person("Lili", 29),
    new Person("Jeff", 44), null};

Vector personVector = new Vector();
LinkedList personList1 = new LinkedList();
LinkedList personList2 = new LinkedList();
ArrayList personArrayList = new ArrayList();
for (int i = 0; i < persons1.length; i++) {
    personVector.add(persons1[i]);
    personList1.add(persons1[i]);
    personList2.add(persons1[i]);
    personArrayList.add(persons1[i]);
}
```

Figure 9-17 Demonstration of sorting functions (continued)

Data Structures and Algorithms in Java

130

Sorting in java.util (continued)

```
Collections.sort(personVector);
System.out.println("personVector = " + personVector);
// personVector = [(Jeff, 44), (Lili, 29), (Tom, 50)]
Collections.sort(personList1);
// personList1 = [(Jeff, 44), (Lili, 29), (Tom, 50)]
Collections.sort(personList2, new PersonComparator());
// personList2 = [(Lili, 29), (Jeff, 44), (Tom, 50)]
Collections.sort(personArrayList);
// personArrayList = [(Jeff, 44), (Lili, 29), (Tom, 50)]
Arrays.sort(persons1);
// persons1 = [(Jeff, 44), (Lili, 29), (Tom, 50)]
Arrays.sort(persons2, new PersonComparator());
// persons2 = [null, (Lili, 29), (Jeff, 44), (Tom, 50)]

Integer[] a = {new Integer(1), new Integer(4),
    new Integer(2), new Integer(5)};
(new Sorts()).insertionSort(a); // a = [1, 2, 4, 5]
Person[] persons3 = {new Person("Tom", 50), new Person("Lili", 29),
    new Person("Jeff", 44)};
(new Sorts()).insertionSort(persons3);
// persons3 = [(Lili, 29), (Jeff, 44), (Tom, 50)]
}
```

Figure 9-17 Demonstration of sorting functions (continued)

Data Structures and Algorithms in Java

131

Bài tập (*)

Cho một danh sách gồm các sinh viên sau:

STT	MSSV	Họ và tên	Năm sinh
1	1005	Trần Minh Thành	1991
2	1001	Trần Thị Bích	1988
3	1003	Trần Minh Thành	1990
4	1000	Vô Quang Vinh	1990
5	1008	Nguyễn Văn An	1990

- . Tạo một cấu trúc dữ liệu để xử lý danh sách trên.
- . Sắp xếp danh sách tăng dần theo mã số tăng dần.
- . Sắp xếp danh sách tăng dần theo tên (thứ tự bảng chữ cái) và năm sinh (nếu trùng tên thì sắp theo năm sinh tăng dần).

Data Structures and Algorithms in Java

132

Đề mẫu Practical Exam (40%)

- Đọc danh sách nhân viên từ file theo định dạng
- Sắp xếp theo mã nhân viên
- Sắp xếp theo tên, nếu trùng tên thì xếp tăng dần theo tuổi
- Xuất ra file theo định dạng

NV000,Nguyen Van An,32,200
 NV001,Nguyen Thi Ly,42,200
 NV002,Le Thanh Tung,28,200
 NV003,Ho Van Hiep,21,200
 NV004,Thi Anh Khoa,20,200

NV000- Nguyen Van An-32-200
 NV001- Nguyen Thi Ly-42-200
 NV002- Le Thanh Tung-28-200
 NV003- Ho Van Hiep-21-200
 NV004- Thi Anh Khoa-20-200

Data Structures and Algorithms in Java

133

Case Study: Adding Polynomials

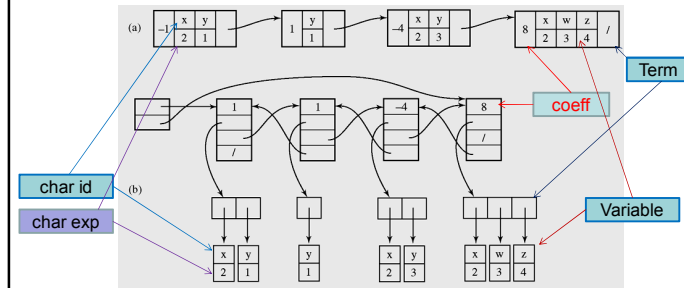


Figure 9-19 A linked list representation of the expression $-x^2y^3 + y - 4x^2y^3 + 8x^2w^3z^4$

Data Structures and Algorithms in Java

Read yourself
Click to Summary

134

Case Study: Adding Polynomials (continued)

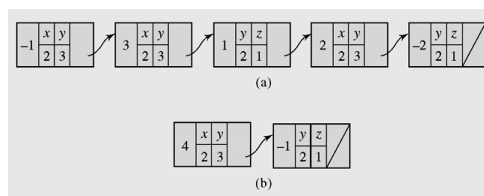


Figure 9-20 Transforming (a) a list representing the expression $-x^2y^3 + 3x^2y^3 + y^2z + 2x^2y^3 - 2y^2z$ into (b) a list representing a simplified version of this expression, $4x^2y^3 - y^2z$

Data Structures and Algorithms in Java

135

Case Study: Adding Polynomials (continued)

```
import java.io.*;
import java.util.*;

class Variable implements Comparable, Cloneable {
    public char id;
    public int exp;
    public Variable() {
    }
    public Variable(char c, int i) {
        id = c; exp = i;
    }
    public int compareTo(Object v) {
        return id - ((Variable)v).id;
    }
    public boolean equals(Object v) {
        return id == ((Variable)v).id && exp == ((Variable)v).exp;
    }
    public Object clone() {
        return new Variable(id, exp);
    }
}

class Term implements Comparable, Cloneable {
    public Term() {
    }
}
```

Figure 9-21 Implementation of the program to add polynomials

Data Structures and Algorithms in Java

136

Case Study: Adding Polynomials (continued)

```
public int coeff;
public ArrayList vars = new ArrayList();
public Object clone() {
    Term t = new Term();
    t.coeff = coeff;
    t.vars = (ArrayList) vars.clone();
    for (int i = 0; i < vars.size(); i++)
        t.vars.set(i, ((Variable)vars.get(i)).clone());
    return t;
}
/** two terms are equal if all variables are the same and
 * corresponding variables are raised to the same power;
 * the first cell of the node containing a term is excluded
 * from comparison, since it stores coefficient of the term;
 */
```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

137

Case Study: Adding Polynomials (continued)

```
public boolean equals(Object term) {
    int i;
    for (i = 0; i < Math.min(vars.size(), ((Term)term).vars.size()) &&
        vars.get(i).equals(((Term)term).vars.get(i)); i++);
    return i == vars.size() && vars.size() == ((Term)term).vars.size();
}
public int compareTo(Object term2) {
    if (vars.size() == 0)
        return 1; // this is just a coefficient;
    else if (((Term)term2).vars.size() == 0)
        return -1; // term2 is just a coefficient;
    Variable var1, var2;
```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

138

Case Study: Adding Polynomials (continued)

```
for (int i = 0; i < Math.min(vars.size(), ((Term)term2).vars.size()); i++) {
    var1 = (Variable)vars.get(i);
    var2 = (Variable)((Term)term2).vars.get(i);
    if (var1.id < var2.id)
        return -1; // this precedes term2;
    else if (var2.id < var1.id)
        return 1; // term2 precedes this;
    else if (var1.exp < var2.exp)
        return -1; // this precedes term2;
    else if (var2.exp < var1.exp)
        return 1; // term2 precedes this;
    return vars.size() - ((Term)term2).vars.size();
}
}
```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

139

Case Study: Adding Polynomials (continued)

```
class Polynomial {
    private LinkedList terms = new LinkedList();
    public Polynomial() {
    }
    private void error(String s) {
        System.out.println(s);
        Runtime.getRuntime().exit(-1);
    }
    public Polynomial add(Polynomial polyn2) {
        ListIterator p1, p2;
        Polynomial result = new Polynomial();
        int i;
        for (p1 = terms.listIterator(); p1.hasNext(); ) // create new polynomial
            result.terms.add(((Term)p1.next()).clone()); // out of copies
```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

140

Case Study: Adding Polynomials (continued)

```

for (p2 = polyn2.terms.listIterator(); p2.hasNext(); ) // of this
    result.terms.add(((Term)p2.next()).clone()); // polynomial and polyn2;
for (i = 0, p1 = result.terms.listIterator(); p1.hasNext();
    i++, p1 = result.terms.listIterator(i)) {
    Term term1 = (Term) p1.next();
    for (p2 = p1; p2.hasNext(); ) {
        Term term2 = (Term) p2.next();
        if (term1.equals(term2)) {
            term2.coeff += term1.coeff;
            result.terms.remove(term1);
            if (term2.coeff == 0) // remove terms with zero
                result.terms.remove(term2); // coefficients;
            i = -1; // to become i = 0 after autoincrement;
            break;
        }
    }
}

```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

141

Case Study: Adding Polynomials (continued)

```

    }
    Collections.sort(result.terms);
    return result;
}

public void get(InputStream fin) {
    int ch = ' ', i, sign, exp;
    boolean coeffUsed;
    char id;
    Term term = new Term();

```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

142

Case Study: Adding Polynomials (continued)

```

try {
    while (ch > -1) {
        coeffUsed = false;
        while (true)
            if (ch > -1 && Character.isWhitespace((char)ch)) // skip
                ch = fin.read(); // blanks;
            else break;
        if (!Character.isLetterOrDigit((char)ch) &&
            ch != ',' && ch != '-' && ch != '+')
            error("Wrong character entered2");
        if (ch == -1)
            break;
        sign = 1;
        while (ch == '-' || ch == '+') { // first get sign(s) of Term
            if (ch == '-')
                sign *= -1;
            ch = fin.read();
            while (Character.isWhitespace((char)ch))
                ch = fin.read();

```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

143

Case Study: Adding Polynomials (continued)

```

    }
    if (Character.isDigit((char)ch)) { // and then its coefficient;
        String number = "";
        while (Character.isDigit((char)ch)) {
            number += (char) ch;
            ch = fin.read();
        }
        while (Character.isWhitespace((char)ch))
            ch = fin.read();
        term.coeff = sign * Integer.valueOf(number).intValue();
        coeffUsed = true;

```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

144

Case Study: Adding Polynomials (continued)

```

    }
    else term.coeff = sign;
    for (i = 0; Character.isLetterOrDigit((char)ch); i++) {
        id = (char) ch;        // process this term:
        ch = fIn.read();        // get a variable name
        if (Character.isDigit((char)ch)) { // and an exponent
            String number = "";        // (if any);
            while (Character.isDigit((char)ch)) {
                number += (char) ch;
                ch = fIn.read();
            }
            exp = Integer.valueOf(number).intValue();
        }
    }

```

Figure 9-21 Implementation of the program to add polynomials
(continued)

Data Structures and Algorithms in Java

145

Case Study: Adding Polynomials (continued)

```

        while (Character.isWhitespace((char)ch))
            ch = fIn.read();

    }
    else exp = 1;
    term.vars.add(new Variable(id,exp));

    terms.add(term.clone());
    term.vars = new ArrayList();
    while (Character.isWhitespace((char)ch))
        ch = fIn.read();
    if (ch == ';') // finish if a semicolon is entered;
        if (coeffUsed || i > 0)
            break;
        else error("Term is missing"); // e.g., 2x - ; or just ';'
    else if (ch != '-' && ch != '+') // e.g., 2x 4y;
        error("wrong character entered");
    }
}

```

Figure 9-21 Implementation of the program to add polynomials
(continued)

Data Structures and Algorithms in Java

146

Case Study: Adding Polynomials (continued)

```

    } catch (IOException io) {
    }
    for (Iterator p = terms.iterator(); p.hasNext(); ) {
        term = (Term) p.next(); // order alphabetically variables
        if (term.vars.size() > 1) // in each term separately;
            Collections.sort(term.vars);
    }
}

public void display() {
    boolean afterFirstTerm = false;
    for (Iterator it = terms.iterator(); it.hasNext(); ) {

```

Figure 9-21 Implementation of the program to add polynomials
(continued)

Data Structures and Algorithms in Java

147

Case Study: Adding Polynomials (continued)

```

    Term term = (Term) it.next();
    System.out.print(" ");
    if (term.coeff < 0) // put '-' before polynomial
        System.out.print("-"); // and between terms (if needed);
    else if (afterFirstTerm) // don't put '+' in front of
        System.out.print("+"); // polynomial;
    afterFirstTerm = true;
    System.out.print(" "); // print a coefficient if
    if (term.vars.size() == 0 || // the term has only
        Math.abs(term.coeff) != 1) // a coefficient or coefficient
        System.out.print(Math.abs(term.coeff)); // is not 1 or -1;
    for (int i = 1; i <= term.vars.size(); i++) {

```

Figure 9-21 Implementation of the program to add polynomials
(continued)

Data Structures and Algorithms in Java

148

Case Study: Adding Polynomials (continued)

```

        Variable var = (Variable) term.vars.get(i-1);
        System.out.print(var.id); // print a variable name
        if (var.exp != 1) // and an exponent, only
            System.out.print(var.exp); // if it is not 1.
    }
    System.out.println();
}
}

```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

149

Case Study: Adding Polynomials (continued)

```

class AddPolyn {
    static public void main(String[] a) {
        Polynomial polyn1 = new Polynomial(), polyn2 = new Polynomial();
        System.out.println("Enter two polynomials, each ended with a semicolon:");
        polyn1.get(System.in);
        polyn2.get(System.in);
        System.out.println("The result is:");
        polyn1.add(polyn2).display();
    }
}

```

**Figure 9-21 Implementation of the program to add polynomials
(continued)**

Data Structures and Algorithms in Java

150

Summary

- Sorting is a two step process: Choose the criteria that will be used to order data and determine how to put a set of data in order using that criterion
- An insertion sort starts by considering the two first elements of the array `data`, which are `data[0]` and `data[1]`
- Selection sort is an attempt to localize the exchanges of array elements by finding a misplaced element first and putting it in its final place

Data Structures and Algorithms in Java

151

Summary (continued)

- A decision tree is when nonterminal nodes of the tree contain conditions or queries for labels, and the leaves have all possible orderings of the array to which the algorithm is applied
- Shell sort divides the original array into subarrays, sorting them separately, and then dividing them again to sort the new subarrays until the whole array is sorted

Data Structures and Algorithms in Java

152

Summary (continued)

- Mergesort makes partitioning as simple as possible and concentrates on merging sorted halves of an array into one sorted array
- Radix sort is a technique to sort integers by proceeding right to left
- Java provides two sets of versions of sorting methods: one for arrays and one for lists