Introduction

Introduction

C

C -----i--

Conclusion

Sorting and Searching

Nguyễn Công Thành—BI9-210 Nguyễn Gia Phong—BI9-184 Nguyễn Văn Tùng—BI9-229 Trần Minh Vương—BI9-239 Trần Hồng Minh—BI8-114

University of Science and Technology of Hà Nội

November 30, 2019

Contents

- 1 Introduction
- Searching
 Linear Search
 Binary Search
- 3 Sorting
 Selection Sort
 Bubble Sort
 Heapsort
- 4 Comparing
 Comparable
 Comparator
- **5** Conclusion

Introduction

Sorting & Searching are important

C

Compari

C l

Introduction

- Sorting & Searching are important
- Object-Oriented Programming

Compani

Conclusi

Introduction

- Sorting & Searching are important
- Object-Oriented Programming
- Implementation in Java

Compan

Conclusi

Introduction

- Sorting & Searching are important
- Object-Oriented Programming
- Implementation in Java
- Generic Programming

Searching

Given a value x, return the [zero-based] index of x in the array, if such x exists. Otherwise, return NOT_FOUND (-1).

Introduction

Linear Search

Sortin

Comparin

Conclusio

Linear Search

Checks sequentially till

Comparin

Conclusio

- Checks sequentially till
 - match found

Introduction

Linear Search

Sortin

Comparir

Conclusio

- Checks sequentially till
 - match found
 - whole list searched

Comparin

Conclusio

- Checks sequentially till
 - match found
 - whole list searched
- Linear time complexity

- Checks sequentially till
 - match found
 - whole list searched
- Linear time complexity
- java.util.List.indexOf

Sortin

Comparii

- Checks sequentially till
 - match found
 - whole list searched
- Linear time complexity
- java.util.List.indexOf
- Example: Search for number 7
 - 4 20 6 9

Sortin

Comparii

- Checks sequentially till
 - match found
 - whole list searched
- Linear time complexity
- java.util.List.indexOf
- Example: Search for number 7
 - 4 20 6 9

Sortin

Comparin

- Checks sequentially till
 - match found
 - whole list searched
- Linear time complexity
- java.util.List.indexOf
- Example: Search for number 7
 - 4 20 6 9

Sortin

Compari

- Checks sequentially till
 - match found
 - whole list searched
- Linear time complexity
- java.util.List.indexOf
- Example: Search for number 7

Implementation

Introduction

Linear Search

Sorting

Comparin

```
import java.util.List;
public class Search
  public static final int NOT FOUND = -1;
  public static linear(List 1, Object o)
    for (int i = 0; i < 1.size(); ++i)
      if (o == null ? l.get(i) == null
                    : o.equals(l.get(i)))
        return i;
    return NOT FOUND;
 }
```

Introduction

Linear Search Binary Search

Sortin

Comparin

Conclusio

Binary Search

For sorted arrays only



Linear Search Binary Search

Sortin

Comparir

Conclusio

- For sorted arrays only
- Repeat halving interval cannot have x till

Comparin

Conclusio

- For sorted arrays only
- Repeat halving interval cannot have x till
 - match found

Introduction

Linear Search
Binary Search

Sortin

Comparia

Conclusio

- For sorted arrays only
- Repeat halving interval cannot have x till
 - match found
 - invalid interval

Compari

Conclusio

- For sorted arrays only
- Repeat halving interval cannot have x till
 - match found
 - invalid interval
- Logarithmic time complexity

Compari

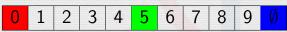
Conclusio

- For sorted arrays only
- Repeat halving interval cannot have x till
 - match found
 - invalid interval
- Logarithmic time complexity
- java.util.Collections.binarySearch

Comparin

Conclusio

- For sorted arrays only
- Repeat halving interval cannot have x till
 - match found
 - invalid interval
- Logarithmic time complexity
- java.util.Collections.binarySearch
- Example: Search for number 7



Linear Search
Binary Search

Sortin

Comparia

- For sorted arrays only
- Repeat halving interval cannot have x till
 - match found
 - invalid interval
- Logarithmic time complexity
- java.util.Collections.binarySearch
- Example: Search for number 7



Binary Search

Introduction

Linear Search
Binary Search

Sortin

Comparir

- For sorted arrays only
- Repeat halving interval cannot have x till
 - match found
 - invalid interval
- Logarithmic time complexity
- java.util.Collections.binarySearch
- Example: Search for number 7



Conclusio

- For sorted arrays only
- Repeat halving interval cannot have x till
 - match found
 - invalid interval
- Logarithmic time complexity
- java.util.Collections.binarySearch
- Example: Search for number 7



Searching Linear Search Binary Search

Sortin

Comparin

```
public class Search
 private static <T> int binary(
   List<? extends Comparable<? super T>> list,
   T key, int low, int high)
   if (high < low)
     return NOT FOUND;
   var mid = (low + high) / 2;
   var cmp = list.get(mid).compareTo(key);
   if (cmp < 0)
     return binary(list, key, mid + 1, high);
   if (cmp > 0)
     return binary(list, key, low, mid - 1);
   return mid;
```

Introduction

Linear Search Binary Search

Sortin

Comparin

```
public class Search
{
  public static <T> int binary(
    List<? extends Comparable<? super T>> list,
    T key)
  {
    return binary(list, key, 0, list.size());
  }
}
```

Selection Sor Bubble Sort

Comparing

Sorting

Given an array of n values, arrange the values into ascending order.

Selection Sort Bubble Sort

Comparin

Selection Sort

 Iterate through every position, select the minimum from there to array's end

Selection Sort Bubble Sort

Comparin

- Iterate through every position, select the minimum from there to array's end
- Quadratic time complexity

Bubble Sort Heapsort

Comparii

C l

- Iterate through every position,
 select the minimum from there to array's end
- Quadratic time complexity
- Example: 6 9 4 2 0

Comparir

- Iterate through every position,
 select the minimum from there to array's end
- Quadratic time complexity
- Example: 0 9 4 2 6

Comparir

_ . .

- Iterate through every position,
 select the minimum from there to array's end
- Quadratic time complexity
- Example: 0 2 4 9 6

Selection Sort Bubble Sort

Compari

- Iterate through every position, select the minimum from there to array's end
- Quadratic time complexity
- Example: 0 2 4 9 6

Comparin

Selection Sort

- Iterate through every position, select the minimum from there to array's end
- Quadratic time complexity
- Example: 0 2 4 6 9

Selection Sort
Bubble Sort

Comparir

Selection Sort

- Iterate through every position, select the minimum from there to array's end
- Quadratic time complexity
- Example: 0 2 4 6 9

Selection Sort

```
import static java.util.Collections.swap;
public class Sort
  public static <T extends Comparable<? super T>>
  void selection(List<T> list)
    int i, j, m, n = list.size();
    for (i = 0; i < n; ++i)
        for (m = j = i; j < n; ++j)
          if (list.get(j).compareTo(list.get(m)) < 0)</pre>
            m = j;
        swap(list, i, m);
```

Introduction

Searchin

Selection So Bubble Sort

Compari

Compani

 Repeatedly iterate through the array, swap adjacent elements in wrong order Sorting

Bubble Sort Heapsort

Compari

Compani

- Repeatedly iterate through the array, swap adjacent elements in wrong order
- Quadratic time complexity

Introduction

Selection So Bubble Sort

Compari

Compani

 Repeatedly iterate through the array, swap adjacent elements in wrong order

Quadratic time complexity

Example:

5 9

9

4

)

Sorting Selection So Bubble Sort

Compari

Сотпрати

 Repeatedly iterate through the array, swap adjacent elements in wrong order

Quadratic time complexity

• Example: 6 9 4 2 0

....

Selection So Bubble Sort

C

. . .

 Repeatedly iterate through the array, swap adjacent elements in wrong order

- Quadratic time complexity
- Example: 6 4 9 2 0

Selection So Bubble Sort

Compari

compani

- Repeatedly iterate through the array, swap adjacent elements in wrong order
- Quadratic time complexity
- Example: 6 4 2 9 0

Selection So Bubble Sort

Camanani

 Repeatedly iterate through the array, swap adjacent elements in wrong order

- Quadratic time complexity
- Example: 6 4 2 0 9

 Repeatedly iterate through the array, swap adjacent elements in wrong order

Quadratic time complexity

• Example: 4

miroductio

Selection So Bubble Sort

Compari

oompan.

 Repeatedly iterate through the array, swap adjacent elements in wrong order

Quadratic time complexity

• Example: 4 2 6 0 9

 Repeatedly iterate through the array, swap adjacent elements in wrong order

Quadratic time complexity

Example:

0

- Repeatedly iterate through the array, swap adjacent elements in wrong order
- Quadratic time complexity
- Example: 2

- Repeatedly iterate through the array, swap adjacent elements in wrong order
- Quadratic time complexity
- Example:

Introductio

Selection So Bubble Sort

Compari

compani

 Repeatedly iterate through the array, swap adjacent elements in wrong order

Quadratic time complexity

• Example: 0 2 4 6 9

Sorting Selection Sor

Bubble Sort Heapsort

Comparii

. . .

Implementation

```
public class Sort
  public static <T extends Comparable<? super T>>
  void bubble(List<T> list)
    for (int n = list.size(), m = 0;
         n > 1; n = m, m = 0
      for (int i = 1; i < n; ++i)
        if (list.get(i).compareTo(list.get(i-1)) < 0)</pre>
          swap(list, m = i, i - 1);
```

Bubble v Selection: Dawn of Sort

C. Thomas Wu (2010) claimed that On average, we expect the bubble sort to finish sorting sooner than the selection sort, because there will be more data movements for the same number of comparisons, and there is a test to exit the method when the array gets sorted.

Sort & Search

Group 11

Bubble Sort Heapsort

Bubble v Selection: Dawn of Sort



Comparin

Conclusion

BvS: Time Complexity

Case	Selection Sort		Bubble Sort	
Best Average	Comparisons $\Omega(n^2)$ $\Theta(n^2)$	Swaps $\Omega(n)$ $\Theta(n)$	Comparisons $\Omega(n)$ $\Theta(n^2)$	Swaps $\Omega(n)$ $\Theta(n^2)$
Worst	$O(n^2)$	O(n)	$O(n^2)$	$O(n^2)$

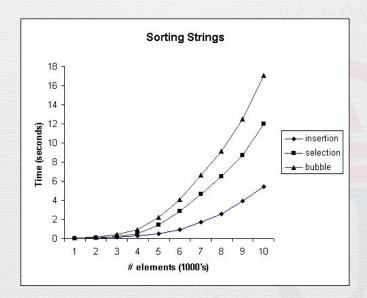
Introduction

Sorting
Selection Sort
Bubble Sort

Heapsort

Company

BvS: Average Case in Practice



Sortin

Selection Se

Heapsort

Comparin

Conclusio

Heapsort

Selection sort, but use heap for selection

Introduction

Sorti

Selection S

Heapsort

Comparin

- Selection sort, but use heap for selection
- Linearithmic time complexity

Bubble S

Heapsort

Compari

Conclusio

Using PriorityQueue (Min-Heap)

```
import java.util.PriorityQueue;
public class Sort
  public static <T extends Comparable<? super T>>
  void pq(List<T> list)
    var q = new PriorityQueue<T>(list);
    for (int i = 0; i < list.size(); ++i)
      list.set(i, q.poll());
```

Heapsort

Using PriorityQueue (Min-Heap)

```
import java.util.PriorityQueue;
public class Sort
  public static <T extends Comparable<? super T>>
  void pq(List<T> list)
    var q = new PriorityQueue<T>(list);
    for (int i = 0; i < list.size(); ++i)
      list.set(i, q.poll());
But hey, there is also List.sort!
```

Sorting

Bubble Sor Heapsort

C

Binary Max-Heap

Nearly complete binary tree

Heapsort

- Nearly complete binary tree
- Parent ≥ Children ⇒ Root is max!

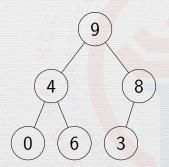
Heapsort

Comparin

Conclusio

Binary Max-Heap

- Nearly complete binary tree
- Parent ≥ Children ⇒ Root is max!
- Example:



Sorting

Selection So

Heapsort

Comparin

Conclusion

Linear Binary Max-Heap

length of inner representation

Comparin

. . .

Linear Binary Max-Heap

- length of inner representation
- size of heap $(0 \le size \le length)$

Conclusion

Linear Binary Max-Heap

- length of inner representation
- size of heap $(0 \le size \le length)$
- Index within [0.. size)

$$parent(i) = \left\lfloor \frac{i-1}{2} \right\rfloor$$
$$left(i) = 2i + 1$$
$$right(i) = 2i + 2$$

_ ...

Sorting Selection So

Heapsort

Comparin

Heap Declaration

```
public class Heap<T extends Comparable<? super T>>
{
   private List<T> list;
   private int size;

   public int getSize() { return size; }
   public int getLength() { return list.size(); }
   public T get(int i) { return list.get(i); }
}
```

Sorting Selection Sor

Heapsort

Compari

Conclusio

void Heap::heapify(int i)

```
int right = i + 1 \ll 1;
int left = right - 1;
int largest = i;
if (left < size
   && get(left).compareTo(get(largest)) > 0)
 largest = left;
if (right < size
   && get(right).compareTo(get(largest)) > 0)
 largest = right;
if (largest != i)
    swap(list, i, largest);
   heapify(largest);
 }
```

Sorting

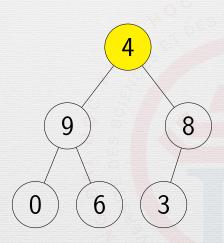
Sorting

Heapsort

Comparing

Canalinatan

Heapification



Sorting

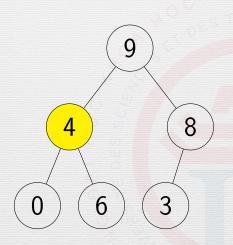
Selection 5

Heapsort

Comparing

Conclusion

Heapification



Sorting

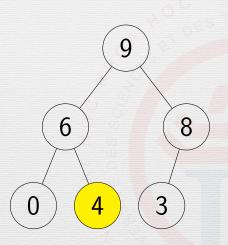
Soloction

Heapsort

Comparing

Conclusion

Heapification



The Loop Invariant

For i = |n/2| - 1 downto 0, heapify(i):

• **Initialization**: For every array, each node $\lfloor n/2 \rfloor ... n-1$ is trival max-heap (leaf).

Heapsort

The Loop Invariant

For i = |n/2| - 1 downto 0, heapify(i):

- **Initialization**: For every array, each node $\lfloor n/2 \rfloor ... n-1$ is trival max-heap (leaf).
- Maintenance: If nodes i+1...n-1 are max-heaps, after heapify(i), all nodes $i \dots n-1$ are max-heaps.

The Loop Invariant

For $i = \lfloor n/2 \rfloor - 1$ downto 0, heapify(i):

- Initialization: For every array, each node $\lfloor n/2 \rfloor ... n-1$ is trival max-heap (leaf).
- Maintenance: If nodes i+1..n-1 are max-heaps, after heapify(i), all nodes i..n-1 are max-heaps.
- **Terminination**: After heapify(0), the whole array is a max-heap.

Bubble S Heapsort

Compari

Conclusio

Heap Constructor

```
public class Heap<T extends Comparable<? super T>>
  public Heap(List<T> a)
    list = a:
    size = a.size();
    for (int i = size >> 1; i-- > 0;)
     heapify(i);
```

Sorting Selection So

Bubble Sort Heapsort

Comparia

Maximum Selection

```
public class Heap<T extends Comparable<? super T>>
  public T pop() throws RuntimeException
    if (size < 1)
      throw new RuntimeException("heap underflow");
    swap(list, 0, --size);
    heapify(0);
    return get(size);
```

Heapsort

Heapsort Implementation

```
public class Sort
  public static <T extends Comparable<? super T>>
  void heap(List<T> list)
  {
    var heap = new Heap<T>(list);
    for (int i = 1; i < list.size(); ++i)
     heap.pop();
```

Introductio

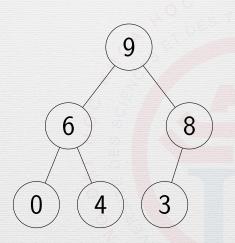
Sorting

Jorting

Heapsort

Comparing

Conclusion



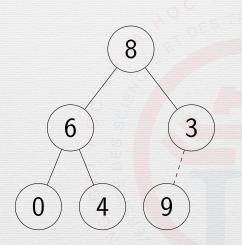
Sorting

Sorting

Heapsort

Comparing

Conclusion



Sorting

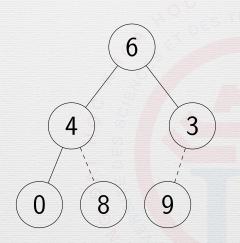
Sorting

Bubble So

Heapsort

Comparing

Conclusion



Sorting

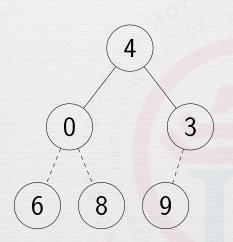
Sorting

Bubble So

Heapsort

Comparing

Conclusion



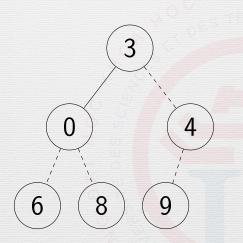
Sorting

Selection

Heapsort

Comparing

Conclusion



IIItroductio

. . .

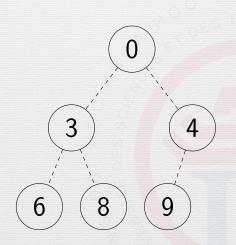
Sorting

Bubble So

Heapsort

Comparing

Conclusion



C l. t

Sorting

Comparing

Comparable

• <,
$$\leq$$
, =, \geq , > or \neq ?

....

Sorting

Comparing

Comparable

- <, \leq , =, \geq , > or \neq ?
- e.g. 420 > 69

THEOGRACIO

Sorting

Comparing

Comparato

- <, \leq , =, \geq , > or \neq ?
- e.g. 420 > 69
- But "420" < "69"!

Comparing

- <, \leq , =, \geq , > or \neq ?
- e.g. 420 > 69
- But "420" < "69"!
- How do we sort any collection of data?

Group 11

Introduction

Introduction

Contin

Sortin

Comparable

Conclusio

Comparable

Natural increasing order

C -----

30111116

Comparator

Conclusio

Comparable

- Natural increasing order
- Define int compareTo(T other)

Introduction

Sorting

Compar

Comparator Comparator

- Natural increasing order
- Define int compareTo(T other)
- Negative: less than; Zero: equal;
 Positve: greater than.

c ..

Sorting

Comparable

Conclusio

Example Element

```
public class Person implements Comparable < Person>
  private String name;
  private Integer age;
  private Character gender;
  public int compareTo(Person other)
    return this.name.compareTo(other.name);
```

Sorting

Comparable

Conclusio

Example Element (misc.)

```
public class Person implements Comparable < Person>
  public Person(String name, Integer age,
                Character gender)
    this.name = name;
    this.age = age;
    this.gender = gender;
  7
  public String toString()
    return String.format("%s (%d%c)",
                         name, age, gender);
```

```
Introduction
```

meroduction

Sorting

Comparable Comparator

Conclus

```
import static java.util.Collections.swap;
public class Sort
  public static <T extends Comparable<? super T>>
  void selection(List<T> list)
    int i, j, m, n = list.size();
    for (i = 0; i < n; ++i)
        for (m = j = i; j < n; ++j)
          if (list.get(j).compareTo(list.get(m)) < 0)</pre>
            m = j;
        swap(list, i, m);
```

Searching

Sorting

Comparable

```
var list = java.util.Arrays.asList(
  new Person("Mahathir Mohamad", 94, 'M'),
  new Person("Elizabeth II", 93, 'F'),
  new Person("Paul Biya", 86, 'M'),
  new Person("Michel Aoun", 84, 'M'),
  new Person("Mahmoud Abbas", 83, 'M'),
  new Person("Francis", 82, 'M'));
Sort.selection(list);
list.forEach(System.out::println);
```

Sorting

Sorting

Comparable

Comparat

Conclusio

Sort by Name Output

Elizabeth II (93F)
Francis (82M)
Mahathir Mohamad (94M)
Mahmoud Abbas (83M)
Michel Aoun (84M)
Paul Biya (86M)

Sorting

Comparable Comparator

Conclusio

Comparator

• How about reverse order?

Sorting

Comparable Comparator

Conclusion

Comparator

- How about reverse order?
- Sort by another key?

Sorting

Comparable

Conclusion

Comparator

- How about reverse order?
- Sort by another *key*?
- compareTo (or any other) method cannot be overriden without subclassing.

- -- -

Sorting

Comparable

Comparati

java.util.Comparator

• Define int compare(T one, T another)

Sorting

Comparable

Conclusio

java.util.Comparator

- Define int compare(T one, T another)
- Negative: less than; Zero: equal;
 Positve: greater than.

Refactored Selection Sort

Comparator

```
public class Sort
  public static <T>
  void selection(List<T> list,
                 Comparator<T> comparator)
    int i, j, m, n = list.size();
    for (i = 0; i < n; ++i)
        for (m = j = i; j < n; ++j)
          if (comparator.compare(list.get(j),
                                  list.get(m)) < 0)
            m = j;
        swap(list, i, m);
```

Sorting

Compari

Comparable Comparator

Conclusio

Exposing Attributes

```
public class Person implements Comparable<Person>
{
   public String getName() { return name; }
   public Integer getAge() { return age; }
   public Character getGender() { return gender; }
}
```

Searching

Sorting

Comparable

```
Sort.heap(list, new Comparator<Person>()
{
   public int compare(Person a, Person b)
   {
     return a.getAge().compareTo(b.getAge());
   }
});
list.forEach(System.out::println);
```

Sorting

Sorting

Comparab

Comparator

Conclusio

Sorting by Age Output

Francis (82M)
Mahmoud Abbas (83M)
Michel Aoun (84M)
Paul Biya (86M)
Elizabeth II (93F)
Mahathir Mohamad (94M)

Backward Compatibility

Introduction

C 1.

Sorting

Comparable Comparator

```
public class Compare<T extends Comparable<? super T>>
implements Comparator<T>
  public int compare(T a, T b)
    return a.compareTo(b);
public class Sort
  public static <T extends Comparable<? super T>>
  void selection(List<T> list)
    selection(list, new Compare<T>());
```

Though the topic is more algorithmic than OOP:

 Encapsulation: Intuitive interface and concise code, e.g. binary search, heap.

Though the topic is more algorithmic than OOP:

- Encapsulation: Intuitive interface and concise code, e.g. binary search, heap.
- Polymorphism: Generic, convenient libraries, thus more code reuse and more effective development.

Though the topic is more algorithmic than OOP:

- **Encapsulation**: Intuitive interface and concise code, e.g. binary search, heap.
- Polymorphism: Generic, convenient libraries, thus more code reuse and more effective development.
- **Inheritance**: Extend objects' functionalities, hence even more generalization.

Though the topic is more algorithmic than OOP:

- Encapsulation: Intuitive interface and concise code, e.g. binary search, heap.
- Polymorphism: Generic, convenient libraries, thus more code reuse and more effective development.
- **Inheritance**: Extend objects' functionalities, hence even more generalization.
- However, shoving every self-contained function into a class is rather redundant.

Sorting

Comparing

Conclusion

• For the list of references, see our report.

Compani

Conclusion

Copying

- For the list of references, see our report.
- The report also contains more explainations and examples.

- For the list of references, see our report.
- The report also contains more explainations and examples.
- The documents as well as Java source files are licensed under CC BY-SA 4.0.