Quicksort e Mergesort: Implemetações

Wagner Emanoel Costa

Precisaremos

- Arquivos usados em aulas passadas
  - StdIn.java
  - StdOut.java
  - StdRandom.java
- Arquivos de Testes
  - tiny.txt
  - words3.txt

O que veremos ?

- Uma forma de implementar o Quicksort
  - Aleatorizado
- Uma forma de implementar o Mergesort

Quicksort

Shuffle – Std Random

```
public static void shuffle(Object[] a) {
    if (a == null) throw new NullPointerException("argument array is null");
    int n = a.length;
    for (int i = 0; i < n; i++) {
        int r = i + uniform(n-i);     // between i and n-1
        Object temp = a[i];
        a[i] = a[r];
        a[r] = temp;
    }
}
```

Shuffle – Std Random – Embaralha o vetor

```
public static void shuffle(Object[] a) {
    if (a == null) throw new NullPointerException("argument array is null");
    int n = a.length;
    for (int i = 0; i < n; i++) {
        int r = i + uniform(n-i);     // between i and n-1
        Object temp = a[i];
        a[i] = a[r];
        a[r] = temp;
    }
}
```

Quicksort

```
public class Quick {

    // This class should not be instantiated.
    private Quick() { }
```

```
  public static void sort(Comparable[] a) {
      StdRandom.shuffle(a);
      sort(a, 0, a.length - 1);
      assert isSorted(a);
  }
```

```
// quicksort the subarray from a[lo] to a[hi]
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
    assert isSorted(a, lo, hi);
}
```

Quicksort

```
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
    while (true) {

        // find item on lo to swap
        while (less(a[++i], v))
            if (i == hi) break;

        // find item on hi to swap
        while (less(v, a[--j]))
            if (j == lo) break;      // redundant since a[lo] a

        // check if pointers cross
        if (i >= j) break;

        exch(a, i, j);
    }

    // put partitioning item v at a[j]
    exch(a, lo, j);

    // now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
    return j;
}
```

Quicksort

```
/***************************************************************
 *  Helper sorting functions.
 ***************************************************************

 // is v < w ?
 private static boolean less(Comparable v, Comparable w) {
     return v.compareTo(w) < 0;
 }

 // exchange a[i] and a[j]
 private static void exch(Object[] a, int i, int j) {
     Object swap = a[i];
     a[i] = a[j];
     a[j] = swap;
 }
```

Quicksort

```
/***************************************************************************
 *  Check if array is sorted - useful for debugging.
 ***************************************************************************
private static boolean isSorted(Comparable[] a) {
    return isSorted(a, 0, a.length - 1);
}

private static boolean isSorted(Comparable[] a, int lo, int hi) {
    for (int i = lo + 1; i <= hi; i++)
        if (less(a[i], a[i-1])) return false;
    return true;
}
```

Quicksort

```
// print array to standard output
private static void show(Comparable[] a) {
    for (int i = 0; i < a.length; i++) {
        StdOut.println(a[i]);
    }
}
```

Quicksort

```
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        Quick.sort(a);
        show(a);

        // shuffle
        StdRandom.shuffle(a);

        // display results again using select
        StdOut.println();
        for (int i = 0; i < a.length; i++) {
            String ith = (String) Quick.select(a, i);
            StdOut.println(ith);
        }
    }

}
```

Fim da Classe

Quicksort - Teste

- java Quick < tiny.txt
- java Quick < words3.txt

MergeSort

Mergesort

```
public class Merge {

    // This class should not be instantiated.
    private Merge() { }
```

Mergesort

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
    // precondition: a[lo .. mid] and a[mid+1 .. hi] are sorted subarrays
    assert isSorted(a, lo, mid);
    assert isSorted(a, mid+1, hi);

    // copy to aux[]
    for (int k = lo; k <= hi; k++) {
        aux[k] = a[k];
    }

    // merge back to a[]
    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        if      (i > mid)              a[k] = aux[j++];
        else if (j > hi)               a[k] = aux[i++];
        else if (less(aux[j], aux[i])) a[k] = aux[j++];
        else                           a[k] = aux[i++];
    }

    // postcondition: a[lo .. hi] is sorted
    assert isSorted(a, lo, hi);
}
```

Mergesort

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
    // precondition: a[lo .. mid] and a[mid+1 .. hi] are sorted subarrays
    assert isSorted(a, lo, mid);
    assert isSorted(a, mid+1, hi);

    // copy to aux[]
    for (int k = lo; k <= hi; k++) {
        aux[k] = a[k];
    }

    // merge back to a[]
    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        if      (i > mid)              a[k] = aux[j++];
        else if (j > hi)               a[k] = aux[i++];
        else if (less(aux[j], aux[i])) a[k] = aux[j++];
        else                           a[k] = aux[i++];
    }

    // postcondition: a[lo .. hi] is sorted
    assert isSorted(a, lo, hi);
}
```

Observe que é necessário copiar

Mergesort

```
// mergesort a[lo..hi] using auxiliary array aux[lo..hi]
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid + 1, hi);
    merge(a, aux, lo, mid, hi);
}
```

```
public static void sort(Comparable[] a) {
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length-1);
    assert isSorted(a);
}
```

Mergesort

```
// mergesort a[lo..hi] using auxiliary array aux[lo..hi]
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid + 1, hi);
    merge(a, aux, lo, mid, hi);
}
```

```
public static void sort(Comparable[] a) {
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length-1);
    assert isSorted(a);
}
```

Observe que é necessário copiar

Mergesort

```
// is v < w ?
private static boolean less(Comparable v, Comparable w) {
    return v.compareTo(w) < 0;
}

// exchange a[i] and a[j]
private static void exch(Object[] a, int i, int j) {
    Object swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```

Mergesort

```
// print array to standard output
private static void show(Comparable[] a) {
    for (int i = 0; i < a.length; i++) {
        StdOut.println(a[i]);
    }
}
```

```
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        Merge.sort(a);
        show(a);
    }
}
```

Mergsort - Teste

- java Merge < tiny.txt
- java Merge < words3.txt

Dúvidas?

Selection Sort

Selection

```java
import java.util.Comparator;
```

```java
public class Selection {

    // This class should not be instantiated.
    private Selection() { }
```

```java
 public static void sort(Comparable[] a) {
     int N = a.length;
     for (int i = 0; i < N; i++) {
         int min = i;
         for (int j = i+1; j < N; j++) {
             if (less(a[j], a[min])) min = j;
         }
         exch(a, i, min);
         assert isSorted(a, 0, i);
     }
     assert isSorted(a);
 }
```

Selection

```java
/**
 * Rearranges the array in ascending order, using a comparator.
 * @param a the array
 * @param c the comparator specifying the order
 */
public static void sort(Object[] a, Comparator c) {
    int N = a.length;
    for (int i = 0; i < N; i++) {
        int min = i;
        for (int j = i+1; j < N; j++) {
            if (less(c, a[j], a[min])) min = j;
        }
        exch(a, i, min);
        assert isSorted(a, c, 0, i);
    }
    assert isSorted(a, c);
}
```

Selection

```
/***************************************************************************
 *  Helper sorting functions.
 ***************************************************************************/

 // is v < w ?
 private static boolean less(Comparable v, Comparable w) {
     return v.compareTo(w) < 0;
 }

 // is v < w ?
 private static boolean less(Comparator c, Object v, Object w) {
     return c.compare(v, w) < 0;
 }


 // exchange a[i] and a[j]
 private static void exch(Object[] a, int i, int j) {
     Object swap = a[i];
     a[i] = a[j];
     a[j] = swap;
 }
```

Selection

```
/***************************************************************************
 *  Check if array is sorted - useful for debugging.
 ***************************************************************************/

 // is the array a[] sorted?
 private static boolean isSorted(Comparable[] a) {
     return isSorted(a, 0, a.length - 1);
 }

 // is the array sorted from a[lo] to a[hi]
 private static boolean isSorted(Comparable[] a, int lo, int hi) {
     for (int i = lo + 1; i <= hi; i++)
         if (less(a[i], a[i-1])) return false;
     return true;
 }

 // is the array a[] sorted?
 private static boolean isSorted(Object[] a, Comparator c) {
     return isSorted(a, c, 0, a.length - 1);
 }

 // is the array sorted from a[lo] to a[hi]
 private static boolean isSorted(Object[] a, Comparator c, int lo, int hi) {
     for (int i = lo + 1; i <= hi; i++)
         if (less(c, a[i], a[i-1])) return false;
     return true;
 }
```

Selection

```
    // print array to standard output
    private static void show(Comparable[] a) {
        for (int i = 0; i < a.length; i++) {
            StdOut.println(a[i]);
        }
    }

    /**
     * Reads in a sequence of strings from standard input; selection sorts them;
     * and prints them to standard output in ascending order.
     */
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        Selection.sort(a);
        show(a);
    }
}
```

Selectionsort - Teste

- java Selection < tiny.txt
- java Selection < words3.txt

Dúvidas?

Insertion Sort

Insertion Sort

```java
import java.util.Comparator;
```

```java
public class Insertion {

    // This class should not be instantiated.
    private Insertion() { }

    /**
     * Rearranges the array in ascending order, using the natural order.
     * @param a the array to be sorted
     */
    public static void sort(Comparable[] a) {
        int N = a.length;
        for (int i = 0; i < N; i++) {
            for (int j = i; j > 0 && less(a[j], a[j-1]); j--) {
                exch(a, j, j-1);
            }
            assert isSorted(a, 0, i);
        }
        assert isSorted(a);
    }
```

Insertion Sort

```
public static void sort(Comparable[] a, int lo, int hi) {
    for (int i = lo; i <= hi; i++) {
        for (int j = i; j > lo && less(a[j], a[j-1]); j--) {
            exch(a, j, j-1);
        }
    }
    assert isSorted(a, lo, hi);
}
```

```
public static void sort(Object[] a, Comparator comparator) {
    int N = a.length;
    for (int i = 0; i < N; i++) {
        for (int j = i; j > 0 && less(a[j], a[j-1], comparator); j--) {
            exch(a, j, j-1);
        }
        assert isSorted(a, 0, i, comparator);
    }
    assert isSorted(a, comparator);
}
```

Insertion Sort

```
public static void sort(Object[] a, int lo, int hi, Comparator comparator) {
    for (int i = lo; i <= hi; i++) {
        for (int j = i; j > lo && less(a[j], a[j-1], comparator); j--) {
            exch(a, j, j-1);
        }
    }
    assert isSorted(a, lo, hi, comparator);
}
```

```
// is v < w ?
private static boolean less(Comparable v, Comparable w) {
    return v.compareTo(w) < 0;
}

// is v < w ?
private static boolean less(Object v, Object w, Comparator comparator) {
    return comparator.compare(v, w) < 0;
}
```

Insertion Sort

```
// exchange a[i] and a[j]
private static void exch(Object[] a, int i, int j) {
    Object swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}

// exchange a[i] and a[j]  (for indirect sort)
private static void exch(int[] a, int i, int j) {
    int swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```

Insertion Sort

```
/***************************************************************************
 *  Check if array is sorted - useful for debugging.
 ***************************************************************************/
private static boolean isSorted(Comparable[] a) {
    return isSorted(a, 0, a.length - 1);
}

// is the array sorted from a[lo] to a[hi]
private static boolean isSorted(Comparable[] a, int lo, int hi) {
    for (int i = lo+1; i <= hi; i++)
        if (less(a[i], a[i-1])) return false;
    return true;
}

private static boolean isSorted(Object[] a, Comparator comparator) {
    return isSorted(a, 0, a.length - 1, comparator);
}

// is the array sorted from a[lo] to a[hi]
private static boolean isSorted(Object[] a, int lo, int hi, Comparator comparator) {
    for (int i = lo + 1; i <= hi; i++)
        if (less(a[i], a[i-1], comparator)) return false;
    return true;
}
```

Insertion Sort

```
   // print array to standard output
    private static void show(Comparable[] a) {
        for (int i = 0; i < a.length; i++) {
            StdOut.println(a[i]);
        }
    }

    /**
     * Reads in a sequence of strings from standard input; insertion sorts them;
     * and prints them to standard output in ascending order.
     */
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        Insertion.sort(a);
        show(a);
    }
}
```

Mergsort - Teste

- java Insertion < tiny.txt
- java Insertion < words3.txt

Dúvidas?

Obrigado pela Atenção