Exam: solution

Tasks: solutions

Task 1 (2 points + 4 points)

a) (2 points)

*6*

*3*

*u*

*1*

b) (4 points)

*6*

*5*

*4*

*v[0]*

*v*

*3*

*2*

*v[1]*

*v[2]*

*1*

Task 2 (3 points + 3 points + 3 points)

a) (3 points)

public static Rectangle minRectangle (Rectangle[] rectangles)

{

if (rectangles.length == 0)

throw new java.lang.IllegalArgumentException ("empty array");

Rectangle minRect = rectangles[0];

double minArea = minRect.area ();

for (int pos = 1; pos < rectangles.length; pos++)

if (rectangles[pos].area () < minArea)

{

minRect = rectangles[pos];

minArea = minRect.area ();

}

return minRect;

}

b) (3 points)

public static Rectangle[] selectRectangles (Rectangle[] rectangles, String color)

{

// the number of rectangles of the given color

int countRectangles = 0;

for (Rectangle r : rectangles)

if (r.getColor ().equals (color))

countRectangles++;

// the rectangles that are of the given color

Rectangle[] rect = new Rectangle[countRectangles];

int pos = 0;

for (Rectangle r : rectangles)

if (r.getColor ().equals (color))

rect[pos++] = r;

return rect;

}

c) (3 points)

Rectangle[] rectangles = { new Rectangle (3, 4, "yellow"),

new Rectangle (1, 2, "blue"),

new Rectangle (5, 6, "red"),

new Rectangle (4, 5, "yellow")};

Rectangle[] selectedRectangles = selectRectangles (rectangles, "yellow");

Rectangle minSelectedRectaangle = minRectangle (selectedRectangles);

Task 3 (3 points + 3 points + 3 points)

a) (3 points)

public void add (char c)

{

if (charCount == chars.length)

{

char[] ca = new char [2 \* chars.length];

for (int pos = 0; pos < chars.length; pos++)

ca[pos] = chars[pos];

chars = ca;

}

chars[charCount++] = c;

}

b) (3 points)

public void replace (int index, char c)

throws java.lang.IndexOutOfBoundsException

{

if (index < 0 || index >= charCount)

throw new java.lang.IndexOutOfBoundsException ("bad index");

chars[index] = c;

}

c) (3 points)

*chars*

*e*

*d*

*c*

*B*

*a*

*5*

*cs*

Task 4 (4 points + 5 points)

a) (4 points)

ABE

ABE--

b) (5 points)

def

f--de

Task 5 (5 points + 4 points)

a) (5 points)

public static void sort (String[] elements)

{

String e = "";

int holePos = 0;

for (int pos = 1; pos < elements.length; pos++)

{

e = elements[pos];

holePos = pos;

while (holePos > 0 && e.compareTo (elements[holePos - 1]) < 0)

{

elements[holePos] = elements[holePos - 1];

holePos--;

}

elements[holePos] = e;

}

}

b) (4 points)

The best case appears when the sequence of elements is already sorted. In this case each element, except the first one, is compared with the element ahead of it. There will be precisely one comparison per element, except for the first element. In total there are *n - 1* comparisons. The corresponding complexity function is:

*b(n) = n – 1*

*b(n) ϵ Θ(n)*

In the best case the algorithm is linear in terms of comparisons.

The worst case appears when the sequence of elements is sorted in reverse. In this case every element, except for the first one, is compared to all elements ahead of it. The total number of comparisons is:

1 + 2 + … + (*n – 2) + (n – 1)*

This means that the time complexity of the algorithm in the worst case, in terms of comparisons, can be given by the following complexity function:

*w(n) = n (n – 1) / 2*

This function can also be written like this:

*w(n) = n2/2 – n/2*

The term with *n2* dominates for sufficiently large values of *n*, and therefore:

*w(n) ϵ Θ(n2)*

In the worst case the algorithm is quadratic in terms of comparisons.