Tentamen: lösning

Uppgifter: lösningar

Uppgift 1 (2 poäng + 4 poäng)

a) (2 poäng)

*6*

*3*

*u*

*1*

b) (4 poäng)

*6*

*5*

*4*

*v[0]*

*v*

*3*

*2*

*v[1]*

*v[2]*

*1*

Uppgift 2 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)

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 poäng)

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

{

// antalet rektanglar av den givna färgen

int countRectangles = 0;

for (Rectangle r : rectangles)

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

countRectangles++;

// de rektanglar som är av den givna färgen

Rectangle[] rect = new Rectangle[countRectangles];

int pos = 0;

for (Rectangle r : rectangles)

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

rect[pos++] = r;

return rect;

}

c) (3 poäng)

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);

Uppgift 3 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)

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 poäng)

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 poäng)

*chars*

*e*

*d*

*c*

*B*

*a*

*5*

*cs*

Uppgift 4 (4 poäng + 5 poäng)

a) (4 poäng)

ABE

ABE--

b) (5 poäng)

def

f--de

Uppgift 5 (5 poäng + 4 poäng)

a) (5 poäng)

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 poäng)

Bästa fall uppstår när sekvensen med element är redan sorterad. I detta fall jämförs varje element, utom det första, med det element som finns framför det. Det blir precis en jämförelse per element, utom det första elementet. Totalt är det *n - 1* jämförelser. Motsvarande komplexitetsfunktion är:

*b(n) = n – 1*

*b(n) ϵ Θ(n)*

I bästa fall är algoritmen linjär när det gäller jämförelser.

Värsta fall uppstår när sekvensen med element är omvänt sorterad. I detta fall jämförs varje element, utom det första, med alla element framför det. Det totala antalet jämförelser är:

1 + 2 + … + (*n – 2) + (n – 1)*

Det betyder att algoritmens tidskomplexitet i värsta fall, när det gäller jämförelser, kan ges med följande komplexitetsfunktion:

*w(n) = n (n – 1) / 2*

Denna funktion kan skrivas även så här:

*w(n) = n2/2 – n/2*

Termen med *n2* dominerar för tillräckligt stora värden på *n*, och därför:

*w(n) ϵ Θ(n2)*

I värsta fall är algoritmen kvadratisk när det gäller jämförelser.