# Pyramide probleem

1, 2, 3.

static void Driehoek(int n)

{

Stopwatch sw = new Stopwatch();

sw.Start();

Console.WriteLine(iterative(n));

sw.Stop();

Console.WriteLine("iterative Elapsed={0}", sw.Elapsed);

sw = new Stopwatch();

sw.Start();

Console.WriteLine(recursive(n));

sw.Stop();

Console.WriteLine("recursive Elapsed={0}", sw.Elapsed);

sw = new Stopwatch();

sw.Start();

Console.WriteLine(functional(n));

sw.Stop();

Console.WriteLine("functional Elapsed={0}", sw.Elapsed);

Console.WriteLine();

}

static int iterative(int n)

{

int getal = 0;

for (int i = 1; i <= n; i++)

{

getal = getal + i;

}

return getal;

}

static int recursive(int n)

{

if (n > 0)

{

n = n + recursive(n - 1);

}

return n;

}

static int functional(int n)

{

n = (n / 2) \* (n + 1);

return n;

}

4.

Uit een test met een stopwacht zoals hierboven is aangegeven in de functie Driehoek, komt eruit dat de recursieve manier het snelst is en het iteratieve het langzaamst is.

# Fibonacci

1.

Bij de recursieve manier gebruik je op het begin net als bij de iteratieve manier als N 0/1 is dan return je die waarde. Bij de recursieve manier heb je als de waarde van N ander is dan 0/1 dan is de return waarde de functie N -1 + de functie N -2. Hierdoor krijg bij Fibonacci(0) = 0 Fibonacci(1) = 1 en bij Fibonacci(2) = 1, omdat je N(2) – 1 + N(2) – 2 doet wat uit komt op 1.

2.

static void Fibonacci(int n)

{

Stopwatch sw = new Stopwatch();

sw.Start();

Console.WriteLine(fiboIte(n));

sw.Stop();

Console.WriteLine("iterative Elapsed={0}", sw.Elapsed);

sw = new Stopwatch();

sw.Start();

Console.WriteLine(fiboRec(n));

sw.Stop();

Console.WriteLine("recursive Elapsed={0}", sw.Elapsed);

}

static int fiboRec(int n)

{

if (n == 0)

return 0;

else if (n == 1)

return 1;

else

return fiboRec(n - 1) + fiboRec(n - 2);

}

static int fiboIte(int n)

{

if(n == 0)

return 0;

else if (n == 1)

return 1;

int prev2 = 0;

int prev = 1;

int result = 0;

for (int i = 2; i <= n; i++)

{

result = prev + prev2;

prev2 = prev;

prev = result;

}

return result;

}

3. De iteratieve manier is bij 15 berekening of hoger sneller dan de recursieve manier. Dit komt doordat er op een gegeven moment een omschakeling is van het aantal functies die aangeroepen worden. Na 15x worden erbij de recursieve manier zoveel functies aangeroepen dat hij langzamer wordt.

# String omdraaien

1,3

static void omgedraaidRec(string n)

{

if (n.Length > 0)

{

Console.Write(n[n.Length - 1]);

n = n.Remove(n.Length - 1);

omgedraaidRec(n);

}

}

static bool omgedraaidPal(string n)

{

n = n.ToLower();

if (n.Length >= 2)

{

if (n[0] == n[n.Length - 1])

{

n = n.Remove(0,1);

n = n.Remove(n.Length - 1);

return omgedraaidPal(n);

}

else

{

return false;

}

}

else

{

return true;

}

}

2. wat wordt er met base bedoeld? Ik zou denken dat de base het removen van een char is?

# Tapijt van sierpinski

1.

private void DrawRectangle(Graphics gr, int niveau, RectangleF rect)

{

if (niveau == 0)

{

gr.FillRectangle(Brushes.Black, rect);

}

else

{

float wid = rect.Width / 3f;

float x0 = rect.Left;

float x1 = x0 + wid;

float x2 = x0 + wid \* 2f;

float hgt = rect.Height / 3f;

float y0 = rect.Top;

float y1 = y0 + hgt;

float y2 = y0 + hgt \* 2f;

DrawRectangle(gr, niveau - 1, new RectangleF(x0, y0, wid, hgt));

DrawRectangle(gr, niveau - 1, new RectangleF(x1, y0, wid, hgt));

DrawRectangle(gr, niveau - 1, new RectangleF(x2, y0, wid, hgt));

DrawRectangle(gr, niveau - 1, new RectangleF(x0, y1, wid, hgt));

DrawRectangle(gr, niveau - 1, new RectangleF(x2, y1, wid, hgt));

DrawRectangle(gr, niveau - 1, new RectangleF(x0, y2, wid, hgt));

DrawRectangle(gr, niveau - 1, new RectangleF(x1, y2, wid, hgt));

DrawRectangle(gr, niveau - 1, new RectangleF(x2, y2, wid, hgt));

}

}

2.

Ik het zien tot niveau 12 niveau 9 is haalbaar binnen bepaalde tijd maar vanaf daarboven is het niet te doen. Bij 9 moet hij 16777216