Opgave 1

Skriv en rekursiv algoritme, som har et naturligt tal som parameter og returnerer summen af de ulige tals kvadrater fra 1 til N.

Eksempel: kaldt med parameteren 8 returneres $84\ (1^2 + 3^2 + 5^2 + 7^2)$.

Det er vigtigt at optimere algoritmen, så overflødige rekursive kald undgås.

Svar

```
private static string currentFormula = "";

public static void Main(string[] args)
{

   int result = SumOddSquares(8);
   Console.WriteLine($"Formel: {currentFormula.TrimEnd(' ', '+')}");
   Console.WriteLine($"Resultat: {result}");
}

public static int SumOddSquares(int n)
{
   if (n <= 0)
      return 0;
   if (n % 2 == 0)
      return SumOddSquares(n - 1);

   currentFormula = n + "^2 + " + currentFormula;
   return n * n + SumOddSquares(n - 2);
}
```

Jeg starter med at lave et string $\boldsymbol{currentFormula}$, dette variable vil lave den endelige formular. Int $\boldsymbol{result}$ får resultatet fra rekurasiv algoritme af 8, som er 84. I $\boldsymbol{SumOddSquares}$ funktionen, tjekker jeg først om n er mindre eller i ligemed 0. Hvis n er støre, så tjekker jeg om tallet er ulige. Hvis tallet er ulige, så tilføjer jeg $\boldsymbol{n}$ + "^**2** + " + $\boldsymbol{currentFormula}$ til den nuværende formular, og derefter retunerer $\boldsymbol{return\ n * n + SumOddSquares(n - 2)}$

Opgave 2

Hvad er Store-0 tidskompleksiteten for nedenstående algoritme. Begrund dit svar.

```
public static int myMethod( int N )
{
   int x = 0; int y = 0;
   for (int i = 0; i < N; i++)
   {
        for (int j = 0; j < N; j++)
        {
            for (int k = 0; k < N*Math.sqrt(N); k++)
            //square root; C++: #include <math.h>
            {
                x++;
            }
            j *= 2;
        }
        i += i;
   }
   for (int i = 0; i < N*N; i++)
        y++;
   return x+y;
}
```

Svar

Trin 1: Øverste loop: $for\ (int\ i\ =\ 0;\ i\ <\ N;\ i++)$

her bliver **i** dobblet med **i,** for hver iteration. Derfor kommer vi frem til $O(N)$

Trin 2: Midten loop: $for\ (int\ j\ =\ 0;\ j\ <\ N;\ j++)$

Her sker der det samme. Hvor $j\ *= 2$ bliver dobbelt hver gang. Derfor får vi $O(\log N)$

Trin 3: Inner loop: $for\ (int\ k\ =\ 0;\ k\ <\ N\ *\ Math.sqrt(N);$

Den indre loop kører $\boldsymbol{N^{\frac{3}{2}}}$ gange, da den er afhængig af N og kvadratrod af N. Derfor får vi $O(N^{\frac{3}{2}})$

Trin 4: Kombination

Vi ganger alle tidskompleksiteter sammen

$$O(\mathrm{N}) * O(logN) * O(N^{\frac{3}{2}}) = O(\left(N^{\frac{3}{2}}\right) * O(logN)^2)$$

$$O(\left(N^{\frac{3}{2}}\right) * log\ N)$$

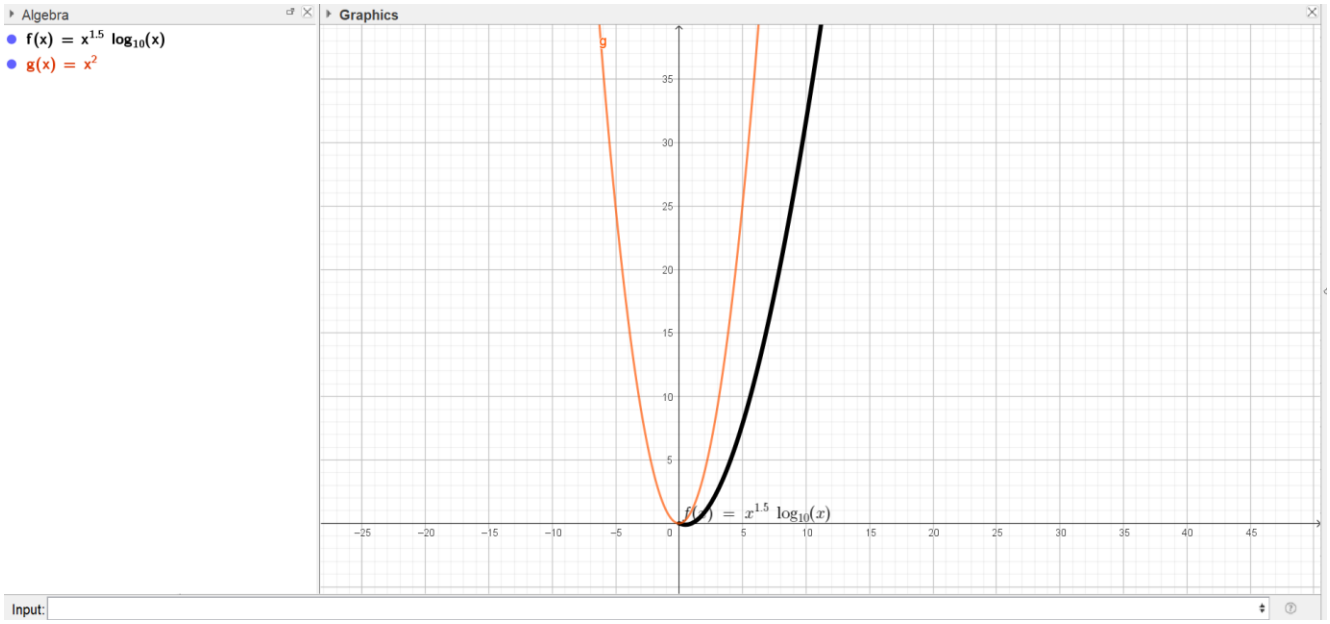
Trin 5: Sidste loop: $for\ (int\ i\ =\ 0;\ i\ <\ N\ *\ N;\ i++)$

Her er det simpel $O(N^2)$ notation, da vi har $N\ *\ N$

Trin 6: Total tidskompleksiteten

$$O\left(N^{\frac{3}{2}} * log\ N\right) + O(N^2)$$

Den dominerende term er $O(N^2)$, da den gror hurtigere end $O\left(N^{\frac{2}{3}} * log\ N\right)$
Derfor er *tidskompleksiteten* $O(N^2)$



Opgave 3

Skriv en *rekursiv* algoritme med følgende signatur:

```
bool additive(String s)
```

Parameteren indeholder en streng bestående af cifre, fx. “82842605”.

Algoritmen returnerer true, hvis parameteren indeholder en substring af tre på hinanden efterfølgende tal, hvor det tredje ciffer er lig med summen af de to forrige.

I ovenstående eksempel returneres *true,* fordi indeks 5 (6) er summen af indeks 3 og 4 (4 plus 2).

Tip: ASCll-værdien af karakteren ‘7’ er 55.

Svar

```
public static void Main(string[] args)
{
   Console.WriteLine(HarTreTalMedSum("82842605"));
}


public static bool HarTreTalMedSum(string value)
{
   for (int i = 0; i < value.Length - 2; i++)
   {
      int a = value[i] - '0';
      int b = value[i + 1] - '0';
      int c = value[i + 2] - '0';

      if (c == a + b)
```

```
        {
            return true;
        }
    }
    return false;
}
```

$stringValue[i] - '0'$ betyder at vi udnytter ASCII-værdierne. Så hvis 7 har ASCII værdien på 55 og vi subtraherer med ASCII værdien på 0, får vi $55 - 48$, som er det samme som 7.

Opgave 4

Skriv en algoritme, der har et array af usorterede, entydige naturlige tal som input og find de tre tal i arrayet, hvis sum er tættest på en potens af 2. Det samme tal kan kun bruges en gang.

Algoritmens returværdi skal være et heltalsarray, som først indeholder de tre tal og dernæst den tilhørende potens af to (fx 512).

Kaldt med arrayet {23,56,22,11,65,89,3,44,87,910,45,35,98}, returneres de tre tal 89, 3, 35 og potensen af 2: 128.

Hvad er Store-O tidskompleksiteten af din algoritme? Begrund dit svar og diskuter mulighederne for at optimere din løsning yderligere.

SVAR:

Trin 1: Øverste loop: for (int i = 0; i < numbers.Length - 2; i++)

Loop kører $O(N)$

Trin 2: Midten loop: for (int j = i + 1; j < numbers.Length - 1; j++)

Loop kører $O(N)$

Trin 3: Inner loop: for (int k = j + 1; k < numbers.Length; k++)

Loop kører $O(N)$

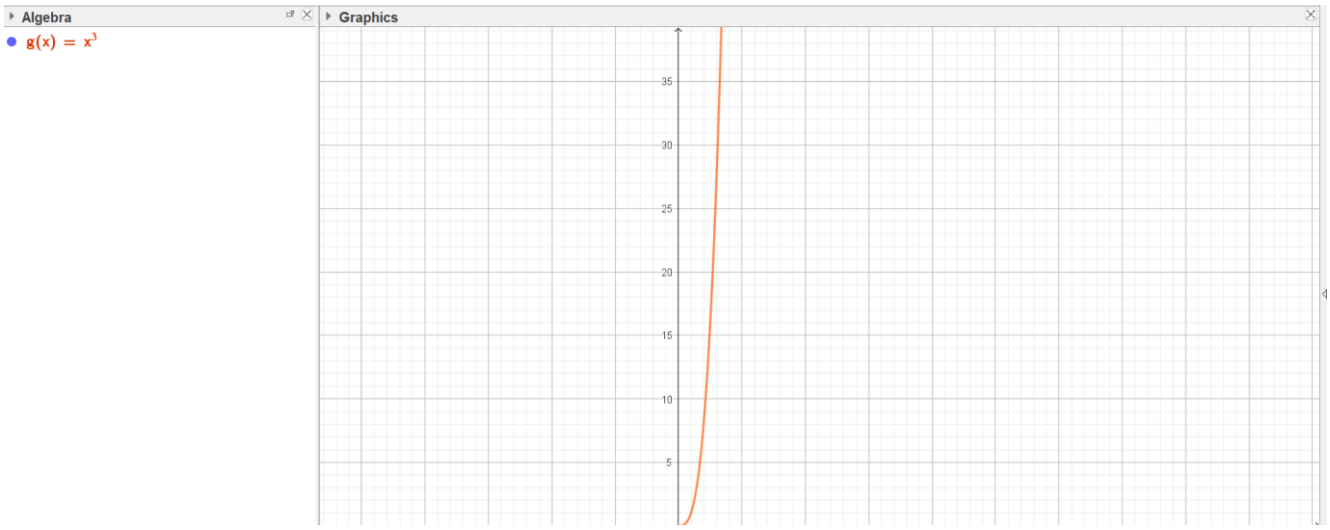
Trin 4: Kombination

Vi ganger alle tidskompleksiteter sammen

$$O(\mathrm{N}) * O(N) * O(N) = O(\mathrm{N}^3)$$

Trin 5: Total tidskompleksiteten

$$O(\mathrm{N}^3)$$



Den nuværende tidskompleksitet $O(\mathrm{N}^3)$ kan forbedres til $O(\mathrm{N}^2)$, ved at ikke anvende 3 nestede loops. Det kan man gøre ved at kun anvende 2 nestede loops, og derefter tilføje den trejdre værdi ind til den indre loop.

```
public static void Main(string[] args)
{
  int[] numbers = {23,56,22,11,65,89,3,44,87,910,45,35,98};
  var values = SumOfThreeNumbers(numbers);

    Console.WriteLine($"The three numbers are: {values[0]}, {values[1]}, {values[2]}");
    Console.WriteLine($"The nearest power of two is: {values[3]}");

}

public static int[] SumOfThreeNumbers(int[] numbers)
{
    int bestFirstNumber = 0, bestSecondNumber = 0, bestThirdNumber = 0;
    int bestPowerOfTwo = 0;
    int bestDiff = int.MaxValue;

    for (int i = 0; i < numbers.Length - 2; i++)
    {
        for (int j = i + 1; j < numbers.Length - 1; j++)
        {
            for (int k = j + 1; k < numbers.Length; k++)
            {
                int currentSum = numbers[i] + numbers[j] + numbers[k];
                int nearestPowerOfTwo = NearestPowerOfTwo(currentSum);
                int currentDiff = Math.Abs(nearestPowerOfTwo - currentSum);

                if (currentDiff < bestDiff)
                {
                    bestDiff = currentDiff;
                    bestFirstNumber = numbers[i];
                    bestSecondNumber = numbers[j];
                    bestThirdNumber = numbers[k];
                    bestPowerOfTwo = nearestPowerOfTwo;
                }
            }
        }
    }
    return new int[] { bestFirstNumber, bestSecondNumber, bestThirdNumber, bestPowerOfTwo };
}

private static int NearestPowerOfTwo(int n)
{
    int power = 1;
    while (power < n)
    {
        power *= 2;
    }

    return power;
}
```

Opgave 5

Hvad er Store-0 tidskompleksiteten af nedenstående metode? Begrund dit svar.

```
int myMethod(int N)
{
        int x = 0;
        for (int i = 1; i <= Math.sqrt(N); i++) //square root; C++: #include <math.h>
        {
                for (int j = 1; j <= N; j++)
                        for (int k = 1; k < N;)
                        {
                                x++;
                                k = k * 2;
                        }
        }
        return x;
 }
```

Trin 1: Øverste loop: $for(int\ i\ =\ 1;\ i\ <=\ Math.sqr(N)\ i++)$

Loop kører $\sqrt{(N)}$ gange: $O\left(\sqrt{(N)}\right)$

Trin 2: Midten loop: $for\ (int\ j\ =\ 1;\ j\ <=\ N;\ j++)$

Loop kører N gange for hver i: $O(N)$

Trin 3: Inner loop: $for\ (int\ k\ =\ 1;\ k\ <\ N;\ )\ x++;\ k\ =\ k\ *\ 2$

K bliver ganget med 2, hver iteration: $O(LogN)$

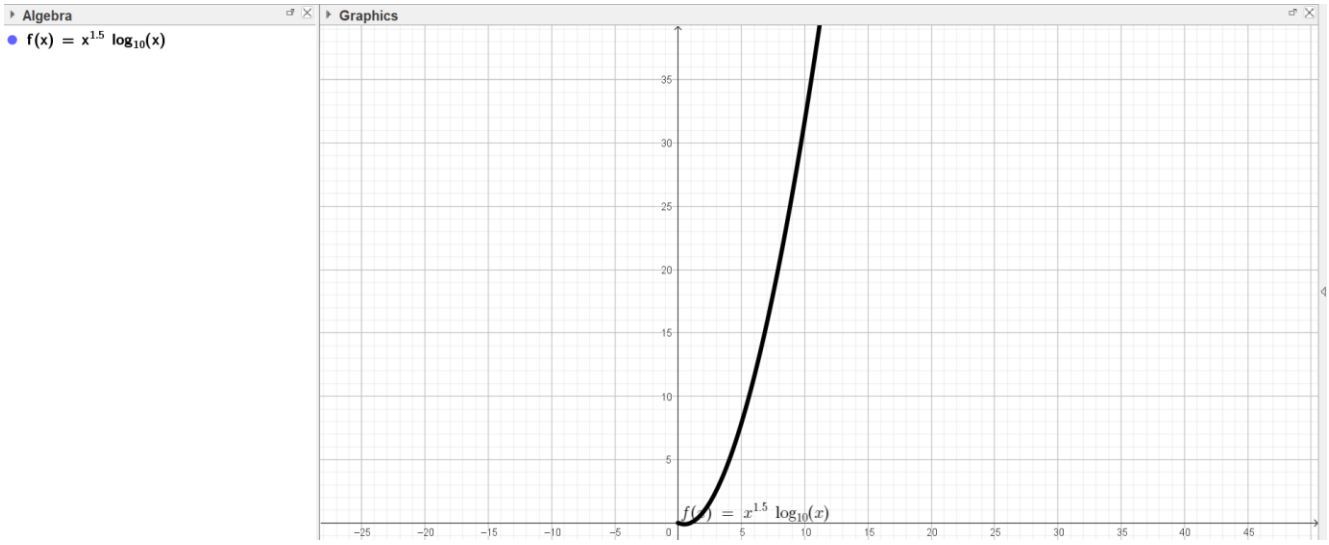
Trin 4: Kombination

Vi ganger alle tidskompleksiteter sammen

$$O(\sqrt{\mathrm{N}}) * O(N) * O(LogN) = O(\mathrm{N}^{\frac{3}{2}} * \mathrm{LogN})$$

Trin 5: Total tidskompleksiteten

$$O(\mathrm{N}^{\frac{3}{2}} * \mathrm{LogN})$$



Opgave 6

Skriv en rekursiv algoritme med følgende signatur:

```
int sumDivisibleBy3(int N)
```

Algoritmen returnerer summen af heltal større end 0 og mindre end eller lig med N, som er dividér bare med 3.

Kaldt med `N = 12` er den korrekte returværdi 30 (3+6+9+12). Kaldt med `N = 14` er den korrekte returværdi også 30.

Din algoritme skal optimeres således, at overfledige rekursive kald undgås.

SVAR:
Vores endlige O notation bliver til O(N)

```
public static void Main(string[] args)
{
   int N = 12;
   int result = SumDivisibleBy3(N);
   Console.WriteLine($"The sum of all numbers less than {N} that are divisible by 3 is: {result}");
}

public static int  SumDivisibleBy3(int N)
{
```

```
    if (N < 3) return 0;
    int sum = 0;
    for (int i = N - N % 3; i >= 3; i -= 3)
    {
        sum += i;
    }

    return sum;
}
```

Opgave 7

6561 er et eksempel på et naturligt tal, som kan skrives som X^y hvor X og Y er heltal, dvs. 9^4. Et andet eksempel er 3125 (5^5).

Skriv en algoritme, som kan afgøre om et givent naturligt tal Z < 100,000 (algoritmens parameter) kan skrives som $Z = X^y$ hvor X og Y er heltal, hvorom det gælder, at X > 2 og Y > 2.

Algoritmens returværdi skal designes således, at felgende information kan udledes fra den:

- Værdien af X (i tilfældet 6561 er X=9).
- Værdien af Y (i tilfældet 6561 er Y=4).
- For det givne Z kan der være flere Iøsningspar (X,Y). For 3125 er der kun et løsningspar; men for 6561 er X=3 og Y=8 også en mulighed. Hvis der er mere end en løsning, skal den med den største X-værdi returneres.
- Hvis det givne Z ikke har noget lesningspar (X,Y) eller Z indeholder en ulovlig værdi, returneres en dummy eller default værdi.

Algoritmen skal optimeres under antagelsen af, at maksimumværdien for Z og minimumsværdierne for X og Y aldrig ændrer sig.

SVAR:

Funktionen er ret simpel. Først definere jeg to variabler for X og Y som begge er -1. Derefter kører jeg en for loop, hvor der for hvert iteration bliver lavet en ny x værdi. For example $27\text{^}1/3 = 3$. Derefter tjekker if statement, hvis $x^y = Z$, og $x > bestX$ så ændre værdierne for bestX og bestY. Til sidst retuneres værdierne bestX og bestY

```
public static void Main(string[] args)
{
   int Z = 6561;
   int[] result = PotentAlgorithm(Z);

   if (result[0] == -1 && result[1] == -1)
   {
      Console.WriteLine($"No valid (x, y) found for Z={Z}");
      return;
   }
   Console.WriteLine($"Result for Z={Z}: x={result[0]}, y={result[1]}");
}

public static int[] PotentAlgorithm(int Z)
{
   int bestX = -1, bestY = -1;
   for (int y = 2; y <= 10; y++)
   {
      int x = (int)Math.Round(Math.Pow(Z, 1.0 / y));

      if (Math.Pow(x, y) == Z)
      {
         if (x > bestX)
         {
            bestX = x;
            bestY = y;
```

```
            }
        }
    }

    return new[] { bestX, bestY };
}
```

Opgave 8

Tabellen nedenfor er en hash tabel, hvor der anvendes quadratic probing til collision resolution.

Indeks	Værdi
0	
1	
2	V
3	R
4	
S	
6	P
7	
8	E
9	
10	F

Indeks 0, 1, 4, 5, 7 og 9 er ledige. Vis hvordan tabellen ser ud, efter at elementerne Q C and H, som hasher til henholdsvis indeks 7, 8 and 2 er indsat. Begrund dit svar.

SVAR:

Jeg har lavet et algoritme i C#, som anvender quadratic probing. Hvis Q skal hashe til indeks 7, så kan vi bare indsætte den, så pladsen er tom. C har ikke plads i række 8, og derfor udnytter vi quadratic probing collision resolution. $((8 + (1)^2)\ \%\ 11 = 9$, dvs C skal på række 9. H har ikke plads på række 2, igen bruger vi probing. $((2 + (3)^2)\ \%\ 11 = 0$. H skal på række 0.

Indeks	Værdi
0	H
1	
2	V
3	R
4	
S	Q
6	P
7	
8	E
9	C
10	F

Opgave 9

Hvad er Store-O tidskompleksiteten for nedenstående metode? Dit svar må godt være baseret på udførte tests. Hvis du vælger at give et svar uden tests, skal svaret begrundes.

```
public static long myMethod(int n)
{
    if (n <= 1)
        return 1;
    else
        return myMethod(n-1) + myMethod(n-2);
}
```

SVAR:

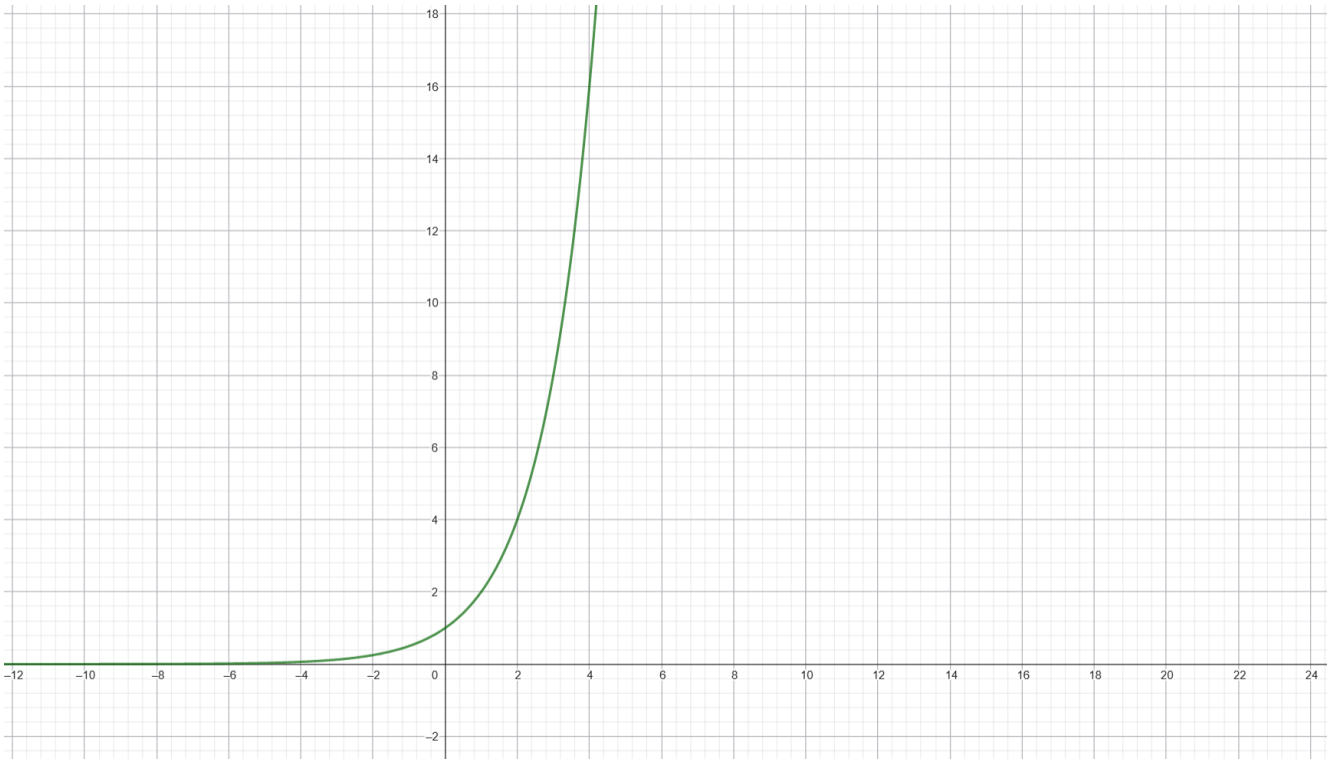
O tidskompleksiteten ender med at være $O(2^n)$, fordi algoritmen køre igennem de samme resultater mange gange, som betyder den gror exponentielt. Man kan tænke på den som et træ. Den føste gange man køre funktionen, splitter den i to andre funktioner, som derefter bliver splittet igen og igen, indtil de når $n \leq 1$

```
public static int counter = 0;
public static void Main(string[] args)
{
   int[] tests = {5, 10, 15, 20, 25, 30, 35, 40};
   foreach (var test in tests)
   {
      counter = 0;
      var result = MyMethod(test);
      Console.WriteLine($"MyMethod({test}) = {result}, calls = {counter}");
   }
}

public static long  MyMethod(int n)
{
   counter++;
   if (n <= 1)
   {
      return 1;
   }
   else
   {
      return MyMethod(n - 1) + MyMethod(n - 2);
   }
}
```

```
MyMethod(5) = 8, calls = 15
MyMethod(10) = 89, calls = 177
MyMethod(15) = 987, calls = 1973
MyMethod(20) = 10946, calls = 21891
```

```
MyMethod(25) = 121393, calls = 242785
MyMethod(30) = 1346269, calls = 2692537
MyMethod(35) = 14930352, calls = 29860703
MyMethod(40) = 165580141, calls = 331160281
```



Opgave 10

Skriv en rekursiv metode med følgende signatur:

$$int\ logTo(int\ N)$$

Algoritmen returner totals-logaritmen af N, og det er en forudsætning, at N er et naturligt tal og en potens af 2.

Kaldt med $N = 32$ returneres 5, og med $N = 4096$ returneres 12.

SVAR:

```
public static int n = 4096;
public static void Main(string[] args)
{
   int result = logTo(n);
   Console.WriteLine("Log2 of " + n + " is: " + result);
}

public static int  logTo(int N)
{
   return N <= 1 ? 0 : 1 + logTo(N / 2);
}
```

Her er der lavet en meget simpel funktion, som klarer opgaven. Funktionen $LogTo$ bruger en one-line if statement som tjekker, om N er mindre end eller lig med 1. Hvis det er tilfældet, returneres 0, ellers returneres $1 + LogTo\left(\frac{N}{2}\right)$ For hvert rekursivt kald deles N med 2, og der lægges 1 til resultatet. Det betyder, at funktionen tæller, hvor mange gange N kan halveres, indtil det bliver 1. For eksempel giver $LogTo(4096)$ resultatet 12, fordi 4096 kan halveres 12 gange før det når 1.

Opgave 11

Tabellen nedenfor repræsenterer de afgivne stemmer ved et valg.

$$(7,4,3,5,3,1,6,4,5,1,7,5)$$

I dette eksempel er der 7 kandidater (1-7), og der er afgivet 12 stemmer. Kandidat 6 fik 1 stemme, kandidaterne 1, 3, 4 og 7 fik hver 2 stemmer, kandidat 5 fik 3 stemmer, og kandidat 2 fik 0 stemmer.

Opgaven går ud på at skrive en algoritme, som kaldt med tabellen og eventuelt tabellens længde, kan afgøre om en kandidat fik mere end 50 % af stemmerne. I så fald returneres kandidatens nummer. Hvis ingen kandidat fik over 50 % af stemmerne, returneres -1.

I eksemplet opnåede ingen af kandidaterne flertal, og der returneres -1.

Hvad er din algoritmes tidskompleksitet?

SVAR:

Funktionen har kun en loop som kører N gange. Dvs O- tidskompleksitet

er $O(N)$

```
int[] votes = { 7, 5, 3, 5, 5, 5, 6, 5, 5, 5, 7, 5 };
Dictionary<int, int> candidatesDic = new Dictionary<int, int>();


foreach (var vote in votes)
{
   if (candidatesDic.ContainsKey(vote))
   {
      candidatesDic[vote]++;
   }
   else
   {
      candidatesDic[vote] = 0;
   }
}
```

```
    int winner = CheckVotes(candidatesDic, votes.Length);
    Console.WriteLine("The winner is candidate: " + winner);
}
public static int CheckVotes(Dictionary<int, int> candidates, int
voteLength)
{
    foreach (var candidate in candidates)
    {
        if(candidate.Value > voteLength / 2)
        {
            return candidate.Key;
        }

    }
    return -1;
}
```