Рекурентні співвідношення:

1. $T(n) = T(n-1) + 3n$ при $n > 0$, $T(0) = 1$

$T(n-1) = T(n-2) + 3(n-1) \Rightarrow T(n) = T(n-2) + 2*3n - 3$

$T(n-2) = T(n-3) + 3(n-2) \Rightarrow T(n) = T(n-3)\ 3*3n - 3 - 6$

$T(n-3) = T(n-4) + 3(n-3) \Rightarrow T(n) = T(n-4) + 4*3n - 3 - 6 - 9$

…

$T(n) = T(0) + (n+1)*3n - 3(1+2+3+..+n) = 1 + (3n^2+3n)/2$

$T(n) = O(n^2) \Rightarrow T(n) = 1 +(3/2)*n^2$

2. $T(n) = T(n-1) + 5$ при $n > 1$, $T(1) = 1$

$T(n-1) = T(n-2) + 5 \Rightarrow T(n) = T(n-1) +2*5$

$T(n-2) = T(n-3) + 5 \Rightarrow T(n) = T(n-1) +3*5$

…

$T(n) = T(n - (n -1)) + (n-1)*5 = 1 + 5n - 5 + 5n - 4$

$T(n) = O(n) \Rightarrow T(n) = 5n - 4$

3. $T(n) = T(n-1) + n/3$ при $n > 0$, $T(0) = 0$

$T(n-1) = T(n-2) + (n-1)/3 \Rightarrow T(n) = T(n-2) + ((n-1) + n)/3$

$T(n-2) = T(n-3) + (n-2)/3 \Rightarrow T(n) = T(n-3) + (n-2 + n-1 + n)/3$

…

$T(n) = T(n-n) + (1+2+..+n)/3 = (n^2+n)/6$

$T(n) = O(n^2) \Rightarrow T(n) = n^2/6$

4. $T(n) = 2T(n-1) + 3$ при $n > 1$, $T(1) = 1$.

$T(n-1) = 2T(n-2) + 3 \Rightarrow T(n) = 2*(2T(n-2) + 3) + 3 = 2^2T(n-2) + 2*3 + 3$

$T(n-2) = 2T(n-3) + 3 \Rightarrow T(n) = 2^2(2T(n-3) + 3) + 2*3 + 3 = 2^3T(n-3) + 2^2*3 + 2*3 + 3$

$T(n-3) = 2T(n-4) + 3 \Rightarrow T(n) = 2^3(2T(n-4) + 3) + 2^2*3 + 2*3 + 3 = 2^4(T-4) + 3(2^3+2^2+2^1+2^0)$

…

$T(n) = 2^{n-1}T(n-(n-1)) + 3*(2^0+2^1+2^2+..+2^{n-3}+2^{n-2}) = 2^{n-1} + 3*(2^{n-1}-1) = 4*2^{n-1} - 3 = 2^{n+1} - 3$

$T(n) = O(2^n) \Rightarrow T(n) = 2^{n+1} - 3$

5. $T(n) = 3T(n-1) + 1$ при $n > 0$, $T(0) = 1$.

$T(n-1) = 3T(n-2) + 1 \Rightarrow T(n) = 3*(3T(n-2)+1)+1=3^2T(n-2)+3^1+3^0$

$T(n-2) = 3T(n-3) + 1 \Rightarrow T(n)=3^2(3T(n-3)+1)+3^1+3^0 = 3^3T(n-3)+3^2+3^1+3^0$

…

$T(n) = 3^n*T(n-n) + 3^0+3^1+3^2+..+3^{n-2}+3^{n-1} = 3^n + (3^n-1)/2 = 3^{n+1}-1/2$

$T(n) = O(3^n) \Rightarrow T(n) = 3^{n+1}-1/2$

6. $T(n) = 3T(n-1)$ при $n > 1$, $T(1) = 4$.

$T(n-1) = 3T(n-2) \Rightarrow T(n) = 3^2T(n-2)$

$T(n-2) = 3T(n-3) \Rightarrow T(n) = 3^3T(n-3)$

…

$T(n) = 3^{n-1}T(n-(n-1)) = 3^{n-1}*4 = 3^n*4/3$

$T(n) = O(3^n) = 3^n*4/3$

Оцінка:

1. $T(n) = 4T(n/2) + n^2$

$a = 4$, $b = 2$, $f(n) =$

$n^{log\_b(a)} = n^{log\_2(4)} = O(n^2)$

Оскільки $f(n) = O(n^2)$, то отримали випадок 2 MasterTheorem:

Тому $T(n) = O(n^2*lg(n))$

2. $T(n) = 4T(n/2) + \sqrt{n}$

$a = 4$, $b = 2$, $f(n) =$

$n^{log\_b(a)} = n^{log\_2(4)} = O(n^2)$

Оскільки $f(n) = O(n^{2-e})$, де $e = 3/2$ то отримали випадок 1 MasterTheorem:

Тому $T(n) = O(n^2)$

3. $T(n) = 4T(n/2) + n^3$

$a = 4$, $b = 2$, $f(n) =$

$n^{log\_b(a)} = n^{log\_2(4)} = O(n^2)$

Оскільки $f(n) = \Omega(n^{2+e})$, де $e = 1$ то отримали випадок 3 MasterTheorem:

Перевіряємо умову регулярності:

$a*f(n/b) = 4*((n/2)^3) = n^3/2 <= c*f(n)$, виконується при $c = 1/2$.

Тому $T(n) = O(n^3)$

4. $T(n) = 3T(n/3) + n^2$

$a = 3$, $b = 3$, $f(n) = n^2$

$n^{log\_b(a)} = n^{log\_3(3)} = O(n^1)$

Оскільки $f(n) = \Omega(n^{1+e})$, де $e = 1$ то отримали випадок 3 MasterTheorem:

Перевіряємо умову регулярності:

$a*f(n/b) = 3*((n/3)^2) = n^2/3 <= c*f(n)$, виконується при $c = 1/3$.

Тому $T(n) = O(n^2)$

5. $T(n) = 3T(n/2) + n$

$a = 3$, $b = 2$, $f(n) = n$

$n^{log\_b(a)} = n^{log\_2(3)} = O(n^{1.585})$

Оскільки $f(n) = O(n^{1.585-e})$, де $e = 0.585$ то отримали випадок 1 MasterTheorem:

Тому $T(n) = O(n^{1.585})$

Алгоритми:
Завдання: Що може обчислювати алгоритм? Визначте інваріант циклу і за його допомогою покажіть коректність алгоритму. Якою є основна операція алгоритму? Скільки разів вона виконується? До якого класу ефективності належить цей алгоритм?

1. АЛГОРИТМ Mystery (s1, s2)

```
// Вхідні дані: слова-рядки s1 та s2
        if length(s1) ≠ length(s2)
                then return false
        for i <= 1 to length(s1) do
                if s1[i] ≠ s2[i]
                        then return false
        return true
```

Цей алгоритм обчислює чи є однаковими рядки s1 та s2.
Інваріант:
1) Інваріант циклу: На початку кожної ітерації циклу for змінна i містить значення індекса, який відповідає символу в рядку s1 та s2, який уже було перевірено на рівність, тобто для всіх j, де $1 \leq j < i$, виконується: $s1[j] = s2[j]$.
2) Ініціалізація: При $i = 1$, ми ще не перевірили жодних символів. Однак інваріант циклу не порушується, оскільки не існує жодних символів для перевірки твердження тривіально вірне.
3) Збереження: При виконанні циклу, коли i збільшується на 1.
Перш ніж перейти до наступної ітерації ми перевіряємо, чи $s1[i] \neq s2[i]$:
1. Якщо символи не однакові, алгоритм повертає false - на даному етапі інваріант виконується.
2. Якщо символи однакові, інваріант зберігається і стверджує, що всі символи s1[j] і s2[j] для $1 \leq j \leq i$ однакові. Тому перед початком наступної ітерації, при $i + 1$, ми можемо стверджувати, що $s1[j] = s2[j]$ для всіх j, де $1 \leq j \leq i$, що знову підтверджує інваріант.

Основна операція: $s1[i] = s2[i]$ – порівняння символів
Скільки разів вона повторюється: У найкращому випадку вона повторюється 0, якщо довжини не однакові. Найгіршим випадком буде n = length(s1), якщо рядки s1 та s2 повінстю однакові або ж мають відмінним лише останній символ.
Клас ефективності: O(n)

2. АЛГОРИТМ CalcSmth (n)

```
// Вхідні дані: натуральне число n
if n = 0 return 0
if n = 1 return 1
a1 <= 0
a2 <= 1
res <= 1
for i <=2 to n do
        res <= a1 + a2
        a1 <= a2
        a2 <= res
return res
```

Алгоритм обчислює n-й член послідовності Фібоначчі.
Інваріант:
1) Інваріант циклу: На початку кожної ітерації циклу змінні a1 і a2 містять значення F_{i-1} та F_i числа Фібоначчі відповідно, де i — це поточний індекс в циклі.
2) Ініціалізація: При i = 2, a1 і a2 ініціалізуються значеннями F(0) = 0 та F(1) = 1. А це коректні початкові умови для обчислення Фібоначчі. Отже, інваріант циклу виконується.
3) Збереження: Кожна ітерація циклу обчислює res=a1+a2, що відповідає F(i).Потім a1 і a2 оновлюються: a1 отримує значення a2 (тобто F_{i-1}), а a2 отримує нове значення res (тобто F_i). Таким чином, інваріант циклу зберігається.
Основна операція: Обчислення res=a1+a2
Кількість виконань: n - 1 разів (для n≥2), якщо враховувати ініціалізацію, то кількість виконань дорівнює n.
Клас ефективності: O(n).

3. АЛГОРИТМ ModifyArray(A[1..n])

```
//Вхідні дані: масив з n дійсних чисел
for i <= 2 to n do
        if A[i-1] > A[i]
        then swap(A[i-1], A[i])
return A[n]
```

Алгоритм виконує перевірку сусідніх елементів масиву. Якщо елемент A[i-1] більший за A[i], він їх обмінює місцями.

1) Інваріант циклу: На початку кожної ітерації циклу for, масив A[1..i-1] містить початкові елементи, а елементи A[j] і A[j-1] (де 2≤j≤i2) гарантовано відсортовані.

2) Ініціалізація: На початку циклу при i=2, інваріант виконується тривіально, оскільки `A[1]` єдине, а обмін ще не відбувається.

3) Збереження: Кожна ітерація перевіряє пару сусідніх елементів і, якщо A[i-1] > A[i], виконується обмін. Після кожної ітерації, A[i-1] і A[i] вже не потребують обміну. Таким чином, інваріант циклу зберігається.

Основна операція: Перевірка умови A[i-1] > A[i] і, за потреби, виконання обміну swap(A[i-1], A[i])

Кількість виконань: n - 1 разів, оскільки цикл іде від 2 до n.

Клас ефективності: O(n).

4. АЛГОРИТМ Secret(A[0..n-1])

```
// Вхідні дані: масив з n дійсних чисел
 minval <= A[0]
 maxval <= A[0]
 for i <= 1 to n-1 do
          if A[i] < minval
               then minval <= A[i]
          if A[i] > maxval
               then maxval <= A[i]
     return  maxval – minval
```

Алгоритм знаходить різницю між найбільшим (`maxval`) і найменшим (`minval`) елементами масиву.

1) Інваріант циклу: На початку кожної ітерації циклу for, змінні minval і maxval містять найменше і найбільше значення серед елементів масиву масив A[1..i-1]

2) Ініціалізація: При i=1, підмасив A[0..0] містить один елемент, тому на початку алгоритму обидві змінні minval і maxval ініціалізуються значенням A[0], що є правильним мінімумом і максимумом для підмасиву довжиною 1

3) Збереження: Кожна ітерація порівнює елемент масиву A[i] з поточними значеннями minval та maxval. Якщо A[i] < minval, то значення minval оновлюється на A[i]. Якщо A[i] > maxval, то значення maxval оновлюється на A[i]. Після кожної ітерації цикл гарантує, що minval і maxval є мінімальним і максимальним значеннями для підмасиву A[0..i], що зберігає інваріант.

Основна операція: Порівняння елемента масиву A[i] з minval і maxval та, за потреби, оновлення відповідних значень.

Кількість виконань: Операція порівняння виконується 2 рази для кожного елемента масиву (один раз для перевірки мінімуму і один раз для максимуму). Оскільки масив має nnn елементів, основна операція виконується 2(n - 1) разів.

Клас ефективності: O(n).

5. АЛГОРИТМ Enigma(A[0..n], x)
// Вхідні дані: масив дійсних чисел A[0..n] та дійсне значення x

```
result <= A[0] + A[1]*x
xPower <= x
for i <= 1 to n do
        xPower <= xPower*x
        result <= result + A[i]*xPower
return  result
```

Алгоритм обчислює значення полінома зі степенями змінної x і коефіцієнтами з масиву A.
1) Інваріант циклу: На початку кожної ітерації циклу for, змінна result містить значення часткової суми полінома до степеня x^{i-1}, а змінна xPower містить значення x^{i}.
2) Ініціалізація: До початку циклу змінна result ініціалізована як сума перших двох членів: result=A[0]+A[1]∗x. Змінна xPower ініціалізована значенням x^1=x, що відповідає другому члену полінома. Ці значення є правильними початковими умовами для обчислення полінома.
3) Збереження: Кожна ітерація циклу оновлює `xPower`, множачи його на `x` (тобто збільшуючи його на один степінь: x^{i}. Потім алгоритм додає новий член полінома до змінної `result`: result=result+A[i]∗x^{i}. Таким чином, на кожній ітерації алгоритм додає новий член полінома зі степенем x^{i}, що зберігає інваріант.
Основна операція: Множення змінної xPower на x і додавання нового члена до змінної result.
Кількість виконань: Множення та додавання виконуються на кожній ітерації циклу for, починаючи з i=2 до i=n. Тобто основні операції в циклі виконуються n - 1 разів. Враховуючи ініціалізацію отримає, що основна операція виконується n разів.
Клас ефективності: O(n).

6. АЛГОРИТМ CheckSmth (n)

```
// Вхідні дані: натуральне число n
if n < 2
        then return 0
i <= 2
count <= 0
while n > 1 do
        if n mod i = 0
                then count <= count + 1
                        while n mod i = 0 do n <= n / i
        i <= i + 1
return count
```

Алгоритм обчислює кількість унікальних простих дільників числа n.
1) Інваріант циклу: На початку кожної ітерації зовнішнього циклу while, змінна count містить кількість унікальних простих дільників, знайдених до поточного значення i, а змінна n — залишок після поділу на всі попередні прості дільники.
2) Ініціалізація: Значення змінної i ініціалізоване як 2 (перше просте число), і count дорівнює 0, що відповідає початковому стану, коли не знайдено жодних простих дільників.
3) Збереження:
Під час кожної ітерації зовнішнього циклу:
Якщо поточне значення n ділиться на i, це означає, що i є простим дільником числа n. У цьому випадку:

- Збільшуємо count, оскільки знайшли новий простий дільник.
- Потім за допомогою внутрішнього циклу ділимо n на i, поки це можливо, щоб позбутися всіх кратних цього простого дільника.

Після кожної ітерації внутрішнього циклу зовнішній цикл збільшує значення i, щоб перевірити наступний потенційний простий дільник.
Основна операція: Перевірка, чи ділиться число n на поточне значення i.
Кількість виконань: Перевірка виконується для кожного числа i, починаючи з 2 до sqrt{n}, оскільки після поділу на прості числа, всі інші можливі дільники будуть меншими за корінь з n.
Клас ефективності: O(sqrt(n)).

7. АЛГОРИТМ MinDistance(A[0..n-1])
//Вхідні дані: масив чисел A[0..n-1]
//Вихідні дані: мінімальна різниця між двома елементами масиву A

```
dmin <= inf
for i <= 0 to n-1 do
    for j <= 0 to n-1 do
        if i != j and  (A[i] – A[j]) < dmin
            dmin <= A[i] – A[j]

return   dmin
```

8. АЛГОРИТМ MaxElement(A[0..n-1])
//Вхідні дані: масив чисел A[0..n-1]
//Вихідні дані: повертає значення найбільшого елемента в масиві A

```
dmin <= inf
for i <= 0 to n-1 do
    for j <= 0 to n-1 do
        if i != j and  (A[i] – A[j]) < dmin
            dmin <= A[i] – A[j]

return   dmin
```

9. АЛГОРИТМ UniqueElements (A[0 .. n − 1])
// Вхідні дані: масив дійсних чисел A[0 .. n − 1]
// Вихідні дані: повертається значення “true” , якщо всі
// елементи масиву A різні, та “false” інакше.

```
for i <= 0 to n − 2 do
    for j <= i + 1 to n − 1 do
        if A[i] = A[j] return false
return true
```

10. АЛГОРИТМ MatrixMultiplication (A[0 .. n − 1, 0 .. n − 1], B[0 .. n − 1, 0 .. n − 1])
// Виконується множення двох квадратних матриць розміром n x n.
// В основі алгоритму лежить визначення цієї операції
// Вхідні дані: дві квадратні матриці A та B розміром n x n
// Вихідні дані: матриця C = AB

```
for i <= 0 to n − 1 do
    for j <= 0 to n − 1 do
        C[i, j] <= 0.0
        for k <= 0 to n − 1 do
            C[i, j] <= C[i, j] + A[i, k] * B[k, j]
return C
```

11. АЛГОРИТМ Binary (n)
// Вхідні дані: ціле додатне число n
// Вихідні дані: кількість розрядів в двійковому представленні n

```
count <= 1
while n > 1 do
```

```
                count <= count + 1
                n <= n/2
        return count
```

12. АЛГОРИТМ Insertion_Sort (A)

```
        for j <= 2 to length[A] do
                key <= A[j]
                // Вставка елемента A[j] у відсортовану послідовність A[1.. j–1]
                i <= j – 1
                while i > 1 and A[i] > key do
                        A[i + 1] <= A[i]
                        i <= i – 1
                A[i + 1] <= key
```

13. АЛГОРИТМ Merge (A, p, q, r)

```
        n1 <= q – p + 1
        n2 <= r – q 3
        L[1.. n1+1]
        R[1.. n2+1]
        for i <= 1 to n1 do
                L[i] <= A[p + i – 1]
        for j <= 1 to n2 do
                R[i] <= A[q + j]
        L[n1+1] <= ∞
        R[n2+1] <= ∞
        i <= 1
        j <= 1
        for k <= p to r do
                if L[i] ≤ R[j]
                then A[k] <= L[i]
                        i <= i + 1
                else A[k] <= R[j]
```

14. АЛГОРИТМ SequentialSearch (A[0 .. n − 1], K)

```
// Вхідні дані: масив чисел A[0 .. n − 1] та ключ пошуку K
// Вихідні дані: повертається значення найбільшого елемента
// масиву A, що дорівнює K, або –1, якщо шуканий елемент не знайдено
        i <= 0
        while i <= n and A[i] ≠ K do i <= i + 1 if i < n return i else return –
        j <= j + 1
```