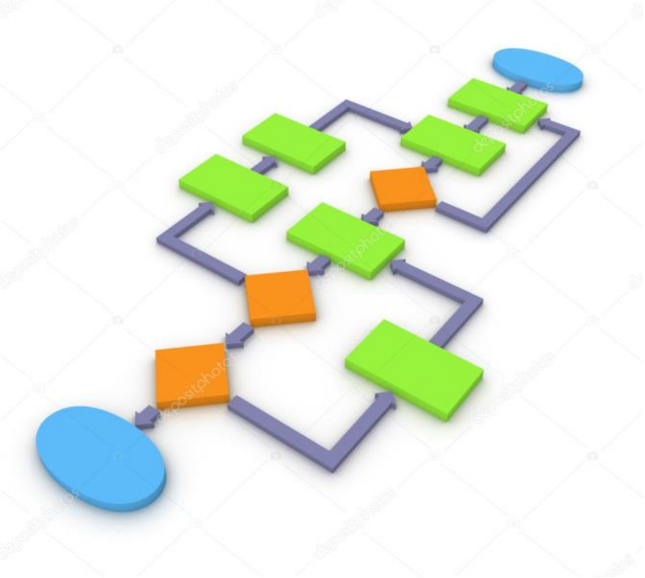
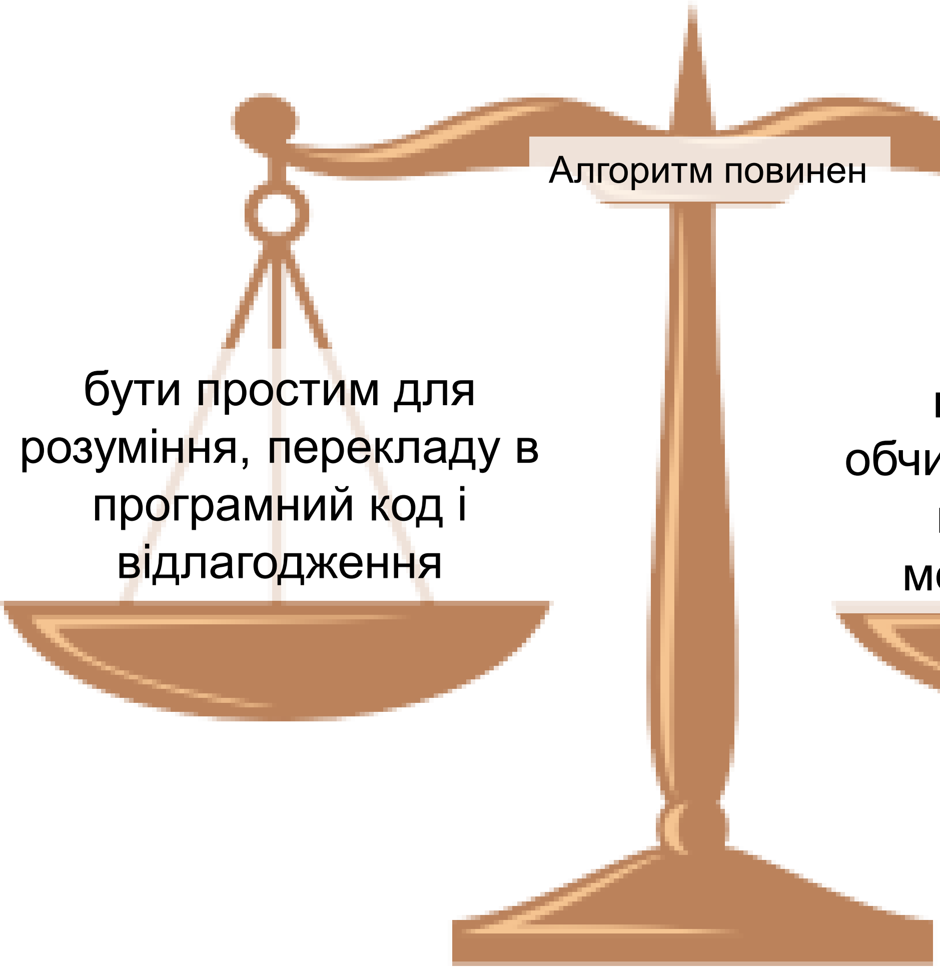
Аналіз складності та ефективності алгоритмів і структур даних

Алгоритм – це набір інструкцій, що описують порядок дій виконавця для розв'язання задачі за скінченну кількість дій; будь-яка коректно визначена обчислювальна процедура, на вхід якої подають деяку величину або набір величин і результатом виконання якої є вихідна величина або набір значень.

Основні властивості алгоритмів:

- зрозумілість
- результативність (скінченність)
- дискретність
- детермінованість
- масовість




Алгоритм повинен
бути простим для розуміння, перекладу в програмний код і відлагодження
ефективно використовувати обчислювальні ресурси і виконуватися по можливості швидко

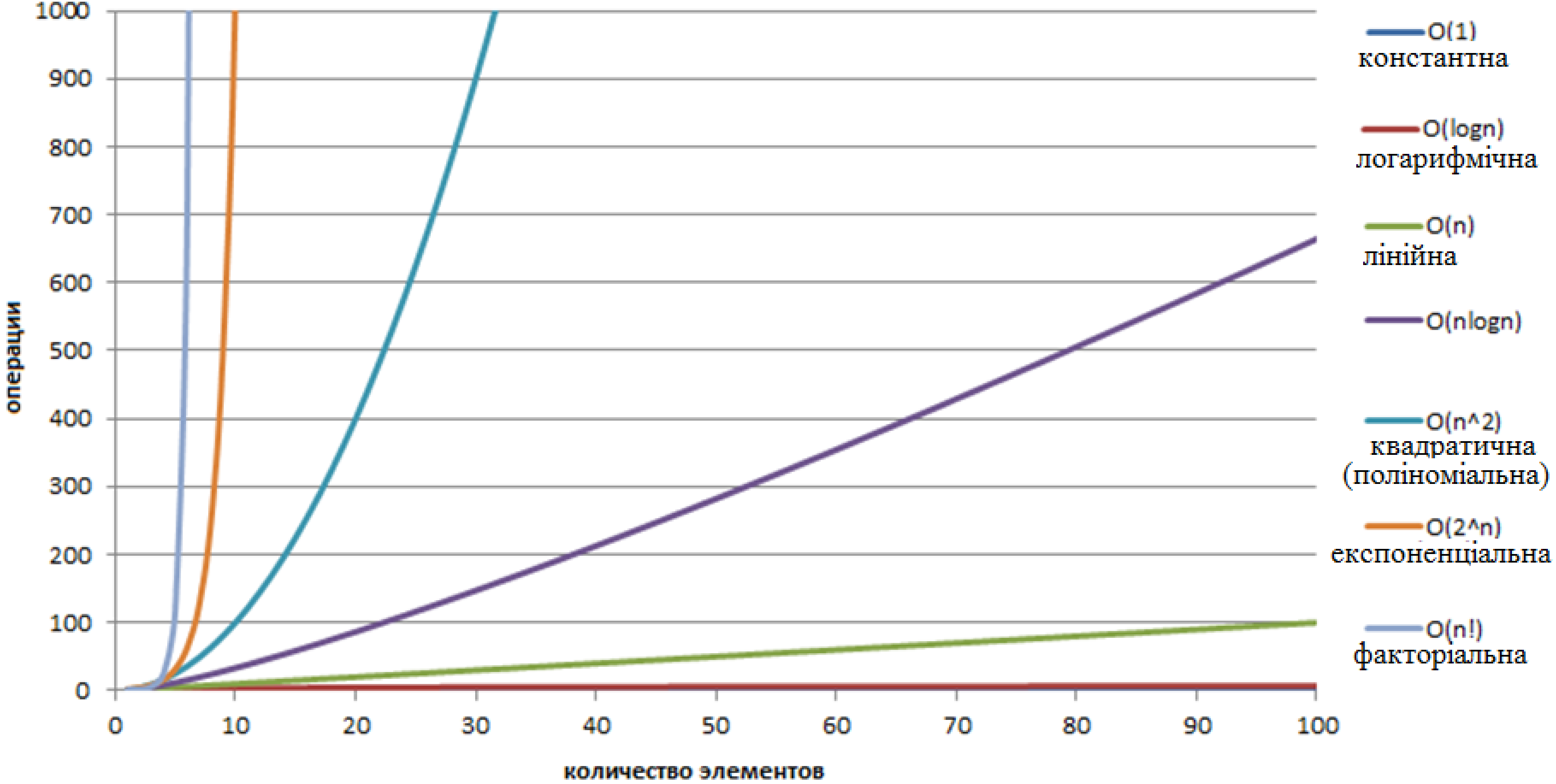
Складність алгоритму – кількісна характеристика, яка відображає порядок величини необхідного ресурсу (часу або додаткової пам’яті) залежно від розмірності задачі.

Обчислювальна складність – це оцінка кількості операцій, необхідних для досягнення результату, залежно від кількості вхідних елементів. Зазвичай говорять, що часова складність алгоритму має порядок $T(N)$ від вхідних даних розміру N.

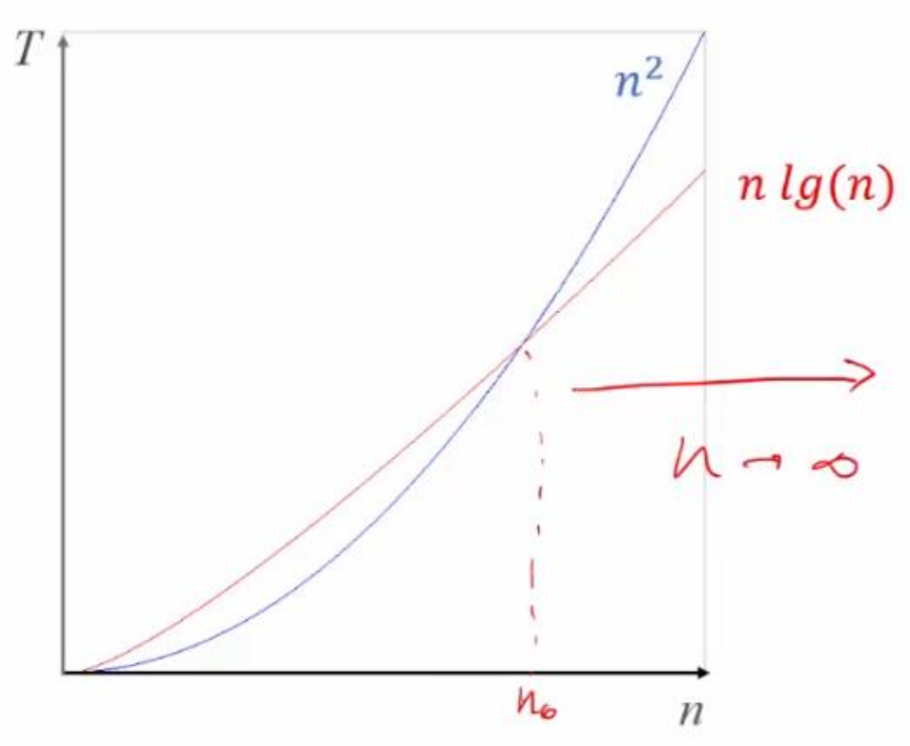
Чому важливо оцінювати швидкість роботи алгоритмів?

	n	$n \log n$	n^2	2^n
$n = 20$	1 сек	1 сек	1 сек	1 сек
$n = 50$	1 сек	1 сек	1 сек	13 дней
$n = 10^2$	1 сек	1 сек	1 сек	$4 \cdot 10^{13}$ лет
$n = 10^6$	1 сек	1 сек	17 мин	
$n = 10^9$	1 сек	30 сек	30 лет	
макс n для 1 сек	10^9	10^7	$10^{4,5}$	30

https://stepik.org/lesson/13230/step/6?unit=3416


1000
900
800
700
600
500
400
300
200
100
0
операции
0
10
20
30
40
50
60
70
80
90
100
количество элементов
O(1)
константна
O(logn)
логарифмічна
O(n)
лінійна
O(nlogn)
O(n^2)
квадратична
(поліноміальна)
O(2^n)
експоненціальна
O(n!)
факторіальна

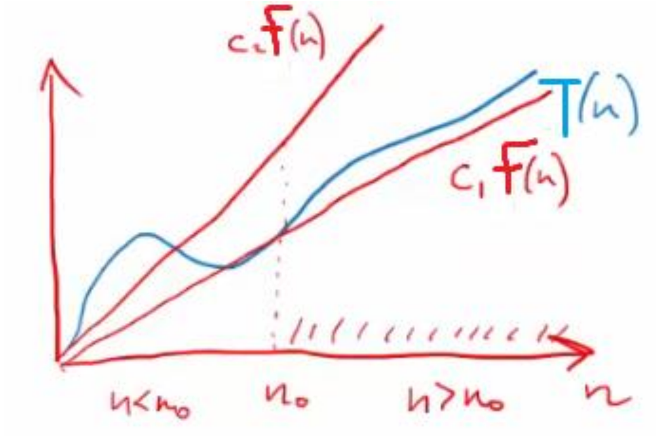
Точно визначити величину $T(n)$ на практиці досить важко. Тому розглядають **асимптотичну складність алгоритму** й оцінюють, як час роботи алгоритму зростає зі збільшенням розміру вхідних даних, коли він збільшується до нескінченності.



https://courses.prometheus.org.ua/courses/KPI/Algorithms101/2015_Spring/about

Означення 1. $T(n)=\Theta(f(n))$. Час роботи алгоритму $T(n)$ має **порядок зростання** $f(n)$, якщо існують натуральне число n_0 і додатні константи c_1 і c_2 ($0 < c_1 \le c_2$) такі, що для будь-якого натурального n, починаючи з n_0, виконується нерівність

$$c_1 \cdot f(n) \le T(n) \le c_2 \cdot f(n).$$

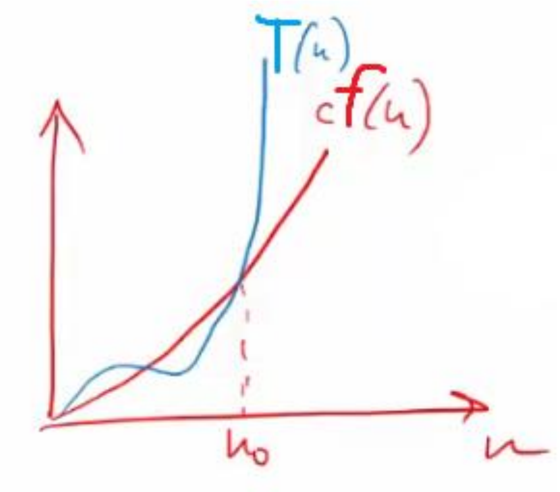


$$T(n)=\Theta(f(n))$$

У такому разі говорять, що функція $f(n)$ є асимптотично точна оцінка функції $T(n)$, оскільки за визначенням вона не відрізняється від функції $f(n)$ із точністю до сталого множника.

Означення 2. $T(n)=\Omega(f(n))$. Час роботи алгоритму $T(n)$ має **нижню оцінку** $f(n)$, якщо існують натуральне число n_0 і додатна константа c такі, що для будь-якого натурального n, починаючи з n_0, виконується нерівність

$$T(n) \geq c \cdot f(n).$$

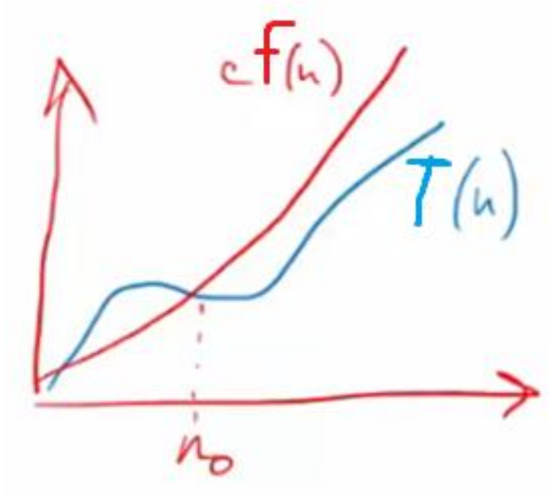


$$T(n)=\Omega(f(n))$$

У такому випадку говорять, що час виконання алгоритму $T(n)$ зростає не повільніше, ніж функція $f(n)$.

Означення 3. $T(n)=O(f(n))$. Час роботи алгоритму $T(n)$ має **верхню оцінку** $f(n)$, якщо існують натуральне число n_0 і додатна константа c такі, що для будь-якого натурального n, починаючи з n_0, виконується нерівність

$$T(n)\le c\cdot f(n).$$



$$T(n)=O(f(n))$$

У такому разі говорять, що час виконання алгоритму $T(n)$ зростає не швидше, ніж функція $f(n)$.

У випадку порівняння верхніх оцінок асимптотичної складності алгоритмів правдиві такі правила:

1) мультиплікативні константи можна опускати: $C \cdot N = O(N)$;

2) N^a зростає швидше N^b, якщо $a > b$. Наприклад, $N^2 = O(N^{2.5})$, $3N^2 + 10N + 35 = O(N^2)$;

3) будь-яка експонента (з основою > 1) зростає швидше будь-якого полінома. Наприклад, рівність $N^5 = O(2^N)$ досить очевидна, але також правдиво $N^{1000} = O(1.01^N)$ за досить великих значень N.

4) будь-який поліном зростає швидше, ніж логарифм. Так, $\log(N)^{20} = O(N^{0.5})$.

$$O(\log N)$$

$$O(\log_? N)$$

$$\log_a N = \frac{\log_b N}{\log_b a}$$

const

Для функції $f(n) = 5n + 2$ вкажіть, яке твердження щодо її приналежності до класів складності є неправильним:

a) $f(n) \in O(n)$
b) $f(n) \in \Theta(n)$
c) $f(n) \in \Omega(n)$
d) $f(n) \in \Theta(n^2)$

Для функції $f(n)=8n^2–20n+5$ вкажіть, яке твердження щодо її приналежності до класів складності є неправильним:

a) $f(n) \in O(n^2)$
b) $f(n) \in O(n^3)$
c) $f(n) \in \Omega(n^3)$
d) $f(n) \in \Omega (n^2)$

Для функції $f(n) = 50 \cdot 2^n \cdot n^2 + 5n - \log(n)$ вкажіть, яке твердження щодо її приналежності до класів складності є правильним:

a) $f(n) \in O(2^n)$
b) $f(n) \in O(2.1^n)$
c) $f(n) \in O(n^2)$
d) усі відповіді є правильними

Приклад
Сортування вставками
(insertion sort)

Задача сортування

На вхід алгоритму подається послідовність n чисел

$$a_1, a_2, \ldots, a_n.$$

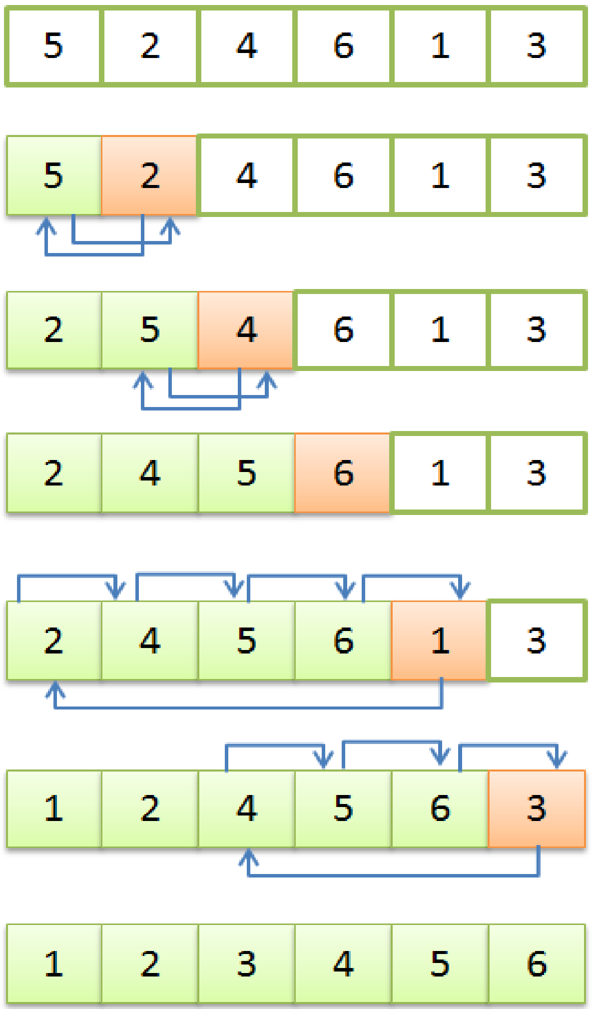
На виході алгоритм повинен повернути перестановку початкової послідовності

$$a'_1, a'_2, \ldots, a'_n,$$

Таким чином, щоб виконувалося наступне співвідношення

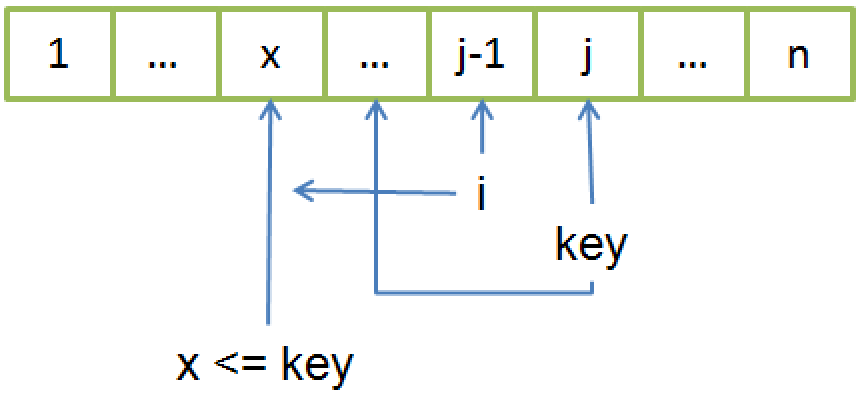
$$a'_1 \leq a'_2 \leq \ldots \leq a'_n.$$

Приклад



Алгоритм

```
for j = 2 to n
    key = A[j]
    i = j – 1
    while i > 0 and A[i] > key
        A[i+1] = A[i]
        i = i – 1
    A[i+1] = key
```



Кількість операцій

for j = 2 to n	c_1 n
key = A[j]	c_2 n-1
i = j – 1	c_3 n-1
while i > 0 and A[i] > key	$c_4 \sum_{2}^{n} t_j$
A[i+1] = A[i]	$c_5 \sum_{2}^{n} (t_j - 1)$
i = i – 1	$c_6 \sum_{2}^{n} (t_j - 1)$
A[i+1] = key	c_7 n-1

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

На практиці час виконання алгоритму часто залежить не тільки від кількості вхідних даних, але і від їх значень, тому розрізняють:

- **максимальну складність** або складність найбільш несприятливого випадку, коли алгоритм працює найдовше;
- **середню складність** – складність алгоритму в середньому;
- **мінімальну складність** – складність у найбільш сприятливому випадку, коли алгоритм виконується найшвидше.

Найкращий випадок для цього алгоритму – коли з самого початку масив вже відсортований:

$$t_j = 1, \quad \sum_{2}^{n} t_j = n-1, \sum_{2}^{n} (t_j - 1) = 0.$$

$$T_{\min}(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) =$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) = A_1 n + B_1$$

$$T_{\min}(n) = O(n)$$

Найгірший випадок – масив відсортований у зворотному порядку: $t_j = j$,

$$\sum_{2}^{n} t_j = \sum_{2}^{n} j = \frac{(2+n)(n-1)}{2} = \frac{n^2+n-2}{2},$$

$$\sum_{2}^{n} (t_j - 1) = \sum_{2}^{n} (j-1) = \frac{(1+n-1)(n-1)}{2} = \frac{n^2-n}{2}.$$

$$T_{\max}(n) = c_1 n + c_2(n-1) + c_3 \cdot (n-1) + c_4 \frac{n^2+n-2}{2} + c_5 \frac{n^2-n}{2} + c_6 \frac{n^2-n}{2} +$$

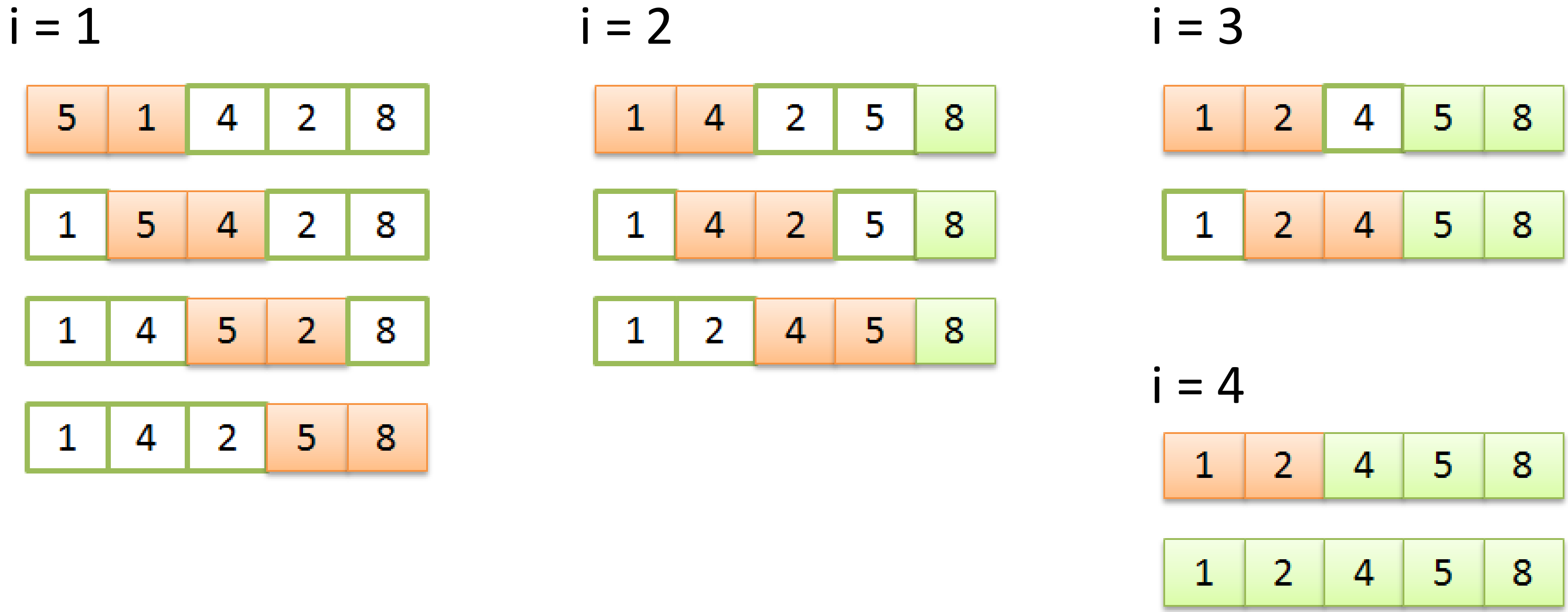
$$+c_7(n-1) = (c_4+c_5+c_6)n^2 + (c_1+c_2+c_3+\frac{c_4}{2}-\frac{c_5}{2}-\frac{c_6}{2}+c_7)n - (c_2+c_3+c_4+c_7) =$$

$$= A_2 n^2 + B_2 n + C_2$$

$$T_{\max}(n) = O(n^2)$$

Сортування бульбашкою
(bubble sort)

Приклад


i = 1
5 1 4 2 8
1 5 4 2 8
1 4 5 2 8
1 4 2 5 8
i = 2
1 4 2 5 8
1 4 2 5 8
1 2 4 5 8
i = 3
1 2 4 5 8
1 2 4 5 8
i = 4
1 2 4 5 8
1 2 4 5 8

Складність

Кількість операцій

```
for i = 1 to n-1
    for j = 1 to n-i
        if A[j] > A[j+1] then
            temp = A[j]
            A[j] = A[j+1]
            A[j+1] = temp
```

n

$$\sum_{1}^{n-1}(n-i+1)=\frac{(n+2)(n-1)}{2}=\frac{n^2+n-2}{2}$$

$$\sum_{1}^{n-1}(n-i)=\frac{(n-1+1)(n-1)}{2}=\frac{n^2-n}{2}$$

=> $T(n)=O(n^2).$

Завдання для самостійної роботи

Опрацювати алгоритми сортування та оцінку їх складності за лекціями курсу “**Розробка та аналіз алгоритмів»** на https://courses.prometheus.org.ua/courses/KPI/Algorithms101/2015_Spring/about

- Сортування включенням
- Сортування злиттям
- Швидке сортування
- Рандомізоване швидке сортування
- Лінійне сортування