2.

Bubble sort - сортування бульбашкою (сортування простими обмінами). Один з найпростіших для розуміння методів сортування масивів, але ефективний він лише для невеликих масивів

3. На слайді можете побачити псевдокод бульбашкового сортування

Алгоритм працює таким чином – у масиві порівнюються два сусідні елементи. Якщо один з елементів, не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями. Прохід по списку продовжується до тих пір, доки дані не будуть відсортованими.

Алгоритм отримав свою назву від того, що процес сортування за ним нагадує поведінку бульбашок повітря у резервуарі з водою. Оскільки для роботи з елементами масиву він використовує лише порівняння, це сортування на основі порівнянь.

4.

**Продуктивність**

Складність алгоритму у найгіршому у середньостатистичному випадку рівна [*О*](https://www.wiki.uk-ua.nina.az/%D0%9D%D0%BE%D1%82%D0%B0%D1%86%D1%96%D1%8F_%D0%9B%D0%B0%D0%BD%D0%B4%D0%B0%D1%83.html)(*n*²), де *n* — кількість елементів для сортування. Існує чимало значно ефективніших алгоритмів, наприклад, з найгіршою ефективністю рівною *O*(*n*log*n*) (це, наприклад, плавне сортування (Smoothsort), пірамідальне сортування (Heapsort,), швидке сортування(Quick Sort), сортування злиттям(merge sort)).

Тому даний алгоритм має низьку ефективність у випадках, коли N є досить великим, за винятком рідкісних конкретних випадків, коли заздалегідь відомо, що масив з самого початку буде добре відсортований.

Якщо список уже відсортовано, немає заміни, і алгоритм буде виконуватися лише n разів. Тож у кращому випадку: O(n)

Якщо список відсортовано в порядку спадання, тобто він відсортований у зворотному порядку, це найгірший випадок і складність за часом: O(n²).

**Переваги**:

         ¤ простота реалізації алгоритму;

         ¤ моделює природну поведінку людини, яка здійснює сортування вручну.

**Недоліки**: не ефективний, якщо необхідно відсортувати масив великого розміру.

### 5.

### Кролики і черепахи

Позиції елементів, що підлягають сортуванню відіграють велику роль у питанні продуктивності даного алгоритму. Великі елементи на початку списку не являють проблему, оскільки вони досить швидко зміщуються на свої місця. Однак, малі елементи у кінці списку переміщуються на його початок дуже повільно. Це призвело до того, що обидва типи елементів було названо [кролями](https://www.wiki.uk-ua.nina.az/%D0%9A%D1%80%D1%96%D0%BB%D1%8C.html) (великі) і [черепахами](https://www.wiki.uk-ua.nina.az/%D0%A7%D0%B5%D1%80%D0%B5%D0%BF%D0%B0%D1%85%D0%B0.html)(малі), відповідно.

З метою підвищення швидкодії алгоритму, у свій час було здійснено чимало зусиль для зменшення кількості «черепах». [Сортування перемішуванням](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D1%96%D1%88%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%D0%BC.html) є порівняно непоганим, однак, усе ще у своєму найгіршому випадку має складність *O(n2)*. [Сортування гребінцем](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B3%D1%80%D0%B5%D0%B1%D1%96%D0%BD%D1%86%D0%B5%D0%BC.html) спершу порівнює великі елементи один з одним, а вже тоді поступово переходить до все менших і менших. Його середньостатистична швидкість приблизно рівна такій в алгоритмі [Швидке сортування](https://www.wiki.uk-ua.nina.az/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F.html).

6.Python

**Масив випадкових чисел – це середня складність. Тому продуктивність алгоритму O(n²), що видно по графіку.**

**7.**

**(діаграма + табл)**

**8.** Python

**Масиви, розбиті на декілька впорядкованих підмасивів**

**9.**

**(діаграма + табл)**

**10.** Python

**Упорядковані масиви зі «свопами»**

**11.**

**(діаграма + табл)**

**12.** Python

**Масиви, вже впорядковані у потрібному порядку**

**13.** Python

**(діаграма + табл)**

**14.** Python

**Масиви, впорядковані у зворотньому порядку**

**15.**

**(діаграма + табл)**

**16.** Python

**(всі графіки)**

**17.** Python

**(загальна діаграма)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**18. C++**

**Масив випадкових чисел – це середня складність. Тому продуктивність алгоритму O(n²), що видно по графіку.**

**19. C++**

**(діаграма + табл)**

**20. C++**

**Масиви, розбиті на декілька впорядкованих підмасивів**

**21. C++**

**(діаграма + табл)**

**22. C++**

**Упорядковані масиви зі «свопами»**

**23. C++**

**(діаграма + табл)**

**24. C++**

**Масиви, вже впорядковані у потрібному порядку**

**25. C++**

**(діаграма + табл)**

**26. C++**

**Масиви, впорядковані у зворотньому порядку**

**27. C++**

**(діаграма + табл)**

**28. C++**

**(всі графіки)**

**29. C++**

**(загальна діаграма)**

**30.**

**(графік с++ пайтон)**

**31.**

**(діаграма с++ пайтон)**

**32.**

**(відео з бабл сорт)**

33

**Висновок**

Через свою простоту, алгоритм часто використовується для пояснення студентам концепції алгоритмів, та алгоритмів сортування, зокрема. Однак, деякі дослідники, як то Оуен Астрахан (Owen Astrachan) ретельно дослідивши даний алгоритм, стверджують, що він настільки поганий і неефективний, що вони навіть не використовуватимуть його як приклад у своїй викладацькій діяльності.

Дональд Кнут у своїй знаменитій праці The Art of Computer Programming прийшов до висновку, що «немає жодних підстав рекомендувати використовувати даний алгоритм, окрім, хіба що через примітну назву і через те, що він є лідером у кількості цікавих теоретичних проблем», частину з яких він обговорює у своїй праці.

Сортування бульбашкою під час своєї роботи є [асимптоматичним](https://www.wiki.uk-ua.nina.az/%D0%9D%D0%BE%D1%82%D0%B0%D1%86%D1%96%D1%8F_%D0%9B%D0%B0%D0%BD%D0%B4%D0%B0%D1%83.html) еквівалентом алгоритму [сортування включенням](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC.html), у своєму найгіршому випадку, однак, обидва алгоритми дуже сильно відрізняються кількістю необхідних операцій переміщення. Результати низки експериментів, наприклад, проведених [Астраханом](https://www.wiki.uk-ua.nina.az/w/index.php?title=%D0%9E%D1%83%D0%B2%D0%B5%D0%BD_%D0%90%D1%81%D1%82%D1%80%D0%B0%D1%85%D0%B0%D0%BD&action=edit&redlink=1) також підтверджують той факт, що продуктивність алгоритму [сортування включенням](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC.html) є значно вищою. Тому багато сучасних посібників з алгоритмів навіть не згадують про алгоритм сортування бульбашками, і віддають перевагу [сортуванню включеннями](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC.html).

Сортування бульбашкою також погано використовує можливості сучасних [мікропроцесорів](https://www.wiki.uk-ua.nina.az/%D0%9C%D1%96%D0%BA%D1%80%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80.html). Він вимагає щонайменше удвічі більше операцій, ніж [сортування включенням](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC.html), удвічі більше [кеш](https://www.wiki.uk-ua.nina.az/%D0%9A%D0%B5%D1%88.html) пам'яті, і [асимптоматично](https://www.wiki.uk-ua.nina.az/%D0%9D%D0%BE%D1%82%D0%B0%D1%86%D1%96%D1%8F_%D0%9B%D0%B0%D0%BD%D0%B4%D0%B0%D1%83.html) більше [передбачень переходів](https://www.wiki.uk-ua.nina.az/w/index.php?title=%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B1%D0%B0%D1%87%D1%83%D0%B2%D0%B0%D1%87_%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4%D1%96%D0%B2&action=edit&redlink=1). Результати експериментів проведених [Астраханом](https://www.wiki.uk-ua.nina.az/w/index.php?title=%D0%9E%D1%83%D0%B2%D0%B5%D0%BD_%D0%90%D1%81%D1%82%D1%80%D0%B0%D1%85%D0%B0%D0%BD&action=edit&redlink=1) показали, що сортування рядка за алгоритмом сортування бульбашками у п'ять разів повільніше за сортування того ж рядка за алгоритмом [сортування включенням](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC.html) і на 40% повільніше за сортування за алгоритмом [сортування вибором](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B8%D0%B1%D0%BE%D1%80%D0%BE%D0%BC.html).

34

***Selection sort*** - сортування вибором

Також один з найпростіших методів сортування масивів.

35.

На слайді можете побачити псевдокод сортування вибором

Алгоритм працює таким чином:

Знаходить у списку найменше значення

Міняє його місцями із першим значеннями у списку

Повторює два попередніх кроки, доки список не завершиться (починаючи з наступної позиції)

Фактично, таким чином ми поділили список на дві частини: перша (ліва) — повністю відсортована, а друга (права) — ні.

Цей метод називається сортуванням вибором оскільки він працює циклічно вибираючи найменший з елементів, що залишилися.

Сортування вставками також може працювати зі списками, які підтримують операції додавання і видалення, як то [зв'язаний список](https://www.wiki.uk-ua.nina.az/%D0%97%D0%B2%27%D1%8F%D0%B7%D0%B0%D0%BD%D0%B8%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA.html). У такому разі, більш зручно видаляти зі списку найменший елемент, і вставляти його в кінець відсортованої частини масиву.

36

**Продуктивність**

Складність алгоритму у найгіршому у середньостатистичному випадку рівна [*О*](https://www.wiki.uk-ua.nina.az/%D0%9D%D0%BE%D1%82%D0%B0%D1%86%D1%96%D1%8F_%D0%9B%D0%B0%D0%BD%D0%B4%D0%B0%D1%83.html)(*n*²), де *n* — кількість елементів для сортування. Це робить його неефективним при сортування великих масивів, і в цілому, менш ефективним за подібний алгоритм [сортування включенням](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC.html). Сортування вибором вирізняється більшою простотою, ніж [сортування включенням](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC.html), і в деяких випадках, вищою продуктивністю.

Якщо список уже відсортовано, немає заміни, і алгоритм буде виконуватися лише n разів. Тож у кращому випадку: O(*n*²),

Якщо список відсортовано в порядку спадання, тобто він відсортований у зворотному порядку, це найгірший випадок і складність за часом: O(n²).

**Переваги**:

* простота у реалізації
* ефективний (зазвичай) на маленьких масивах
* ефективний при сортуванні масивів, дані в яких вже непогано відсортовані: продуктивність рівна O(*n* + *d*), де *d* — кількість інверсій
* на практиці ефективніший за більшість інших квадратичних алгоритмів (O(*n*2)), як то [сортування вибором](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B8%D0%B1%D0%BE%D1%80%D0%BE%D0%BC) та [сортування бульбашкою](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D1%83%D0%BB%D1%8C%D0%B1%D0%B0%D1%88%D0%BA%D0%BE%D1%8E): його швидкодія рівна *n*2/4, і в найкращому випадку є лінійною
* є [стабільним алгоритмом](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

**Недоліки**: для знаходження мінімального елемента на кожному проході доводиться переглядати всі елементи масиву.

37.Python

**Масив випадкових чисел – це середня складність. Тому продуктивність алгоритму O(n²), що видно по графіку.**

**38.**

**(діаграма + табл)**

**39.** Python

**Масиви, розбиті на декілька впорядкованих підмасивів**

**40.**

**(діаграма + табл)**

**41.** Python

**Упорядковані масиви зі «свопами»**

**42.**

**(діаграма + табл)**

**43.** Python

**Масиви, вже впорядковані у потрібному порядку**

**44.** Python

**(діаграма + табл)**

**45.** Python

**Масиви, впорядковані у зворотньому порядку**

**46.**

**(діаграма + табл)**

**47.** Python

**(всі графіки)**

**48.** Python

**(загальна діаграма)**

**49. C++**

**Масив випадкових чисел – це середня складність. Тому продуктивність алгоритму O(n²), що видно по графіку.**

**50. C++**

**(діаграма + табл)**

**51. C++**

**Масиви, розбиті на декілька впорядкованих підмасивів**

**52. C++**

**(діаграма + табл)**

**53. C++**

**Упорядковані масиви зі «свопами»**

**54. C++**

**(діаграма + табл)**

**55. C++**

**Масиви, вже впорядковані у потрібному порядку**

**56. C++**

**(діаграма + табл)**

**57. C++**

**Масиви, впорядковані у зворотньому порядку**

**58. C++**

**(діаграма + табл)**

**59. C++**

**(всі графіки)**

**60. C++**

**(загальна діаграма)**

**61.**

**(графік с++ пайтон)**

**62.**

**(діаграма с++ пайтон)**

**63.**

**(відео з бабл сорт)**

64.

**Висновок**

У сортування вибором незалежно від того, як упорядковані дані, завжди будуть проводитися порівняння та заміни, тому складність часу для найкращого, середнього та гіршого випадку: O(n2)

Сортування вибором — поєднання бульбашкового сортування та сортування вставками (Сортування вставками — елементи вхідної послідовності проглядаються по одному, і кожен новий елемент, що надійшов, розміщується в придатне місце серед раніше упорядкованих елементів. Застосовується для майже цілком відсортованих даних та даних невеликого розміру. Часова складність — O(n2). Як і у бульбашкового сортування, цей алгоритм проходить масивом раз за разом, переміщаючи одне значення на правильну позицію. Однак, на відміну від бульбашкового сортування, вибирає найменше невідсортоване значення замість найбільшого. Як і при сортуванні вставками, упорядкована частина масиву розташована на початку, тоді як у бульбашкового сортування — наприкінці. Часова складність — O(n2).

65.

Порівняння бабл і селекшн на різних мовах для випадкових чисел графік

66.

Порівняння бабл і селекшн на різних мовах для випадкових чисел діаграма

67.

Відео

Bubble-sort with Hungarian ("Csángó") folk dance (угорський, мадярський)

68.

Відео

Select-sort with Gypsy folk dance (циганський)