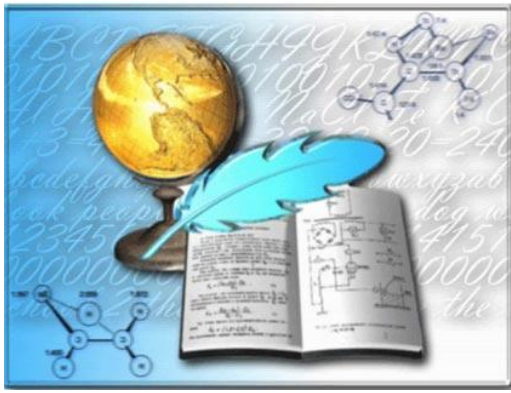
# Тема: Алгоритми пошуку та впорядкування



Задача пошуку елемента у послідовності – одна з найважливіших задач програмування як з теоретичної, так і з практичної точок зору.
Ось її формулювання:

**Нехай $A = \{a_1, a_2, ...\}$ – послідовність однотипних елементів і b - деякий елемент, що має властивість P. Необхідно знайти місце елемента b у послідовності A.**

Оскільки представлення послідовності у пам’яті може бути здійснено у вигляді масиву, то задачі можуть бути уточнені як задачі пошуку елемента у масиві А:

*1. Знайти максимальний елемент масиву,*
*2. Знайти заданий елемент масиву,*
*3. Знайти k-тий за величиною елемент масиву.*

Найбільш прості і оптимальні алгоритми засновано на послідовному перегляді масиву А з перевіркою властивості Р на кожному елементі. Цей метод називають **лінійним (послідовним) переглядом (пошуком).**

# Лінійний пошук

**nX** – номер потрібного елемента в масиві

```
nX = -1; // поки не знайшли ...
for ( i = 0; i < N; i ++)  // цикл по всіх елементах
   if ( A[i] == X )        // якщо знайшли, то ...
      nX = i;              // ... Запамятати номер
if (nX < 0) printf(«не знайшли...")
else        printf("A[%d]=%d", nX, X);
```

**Як можна покращити?**

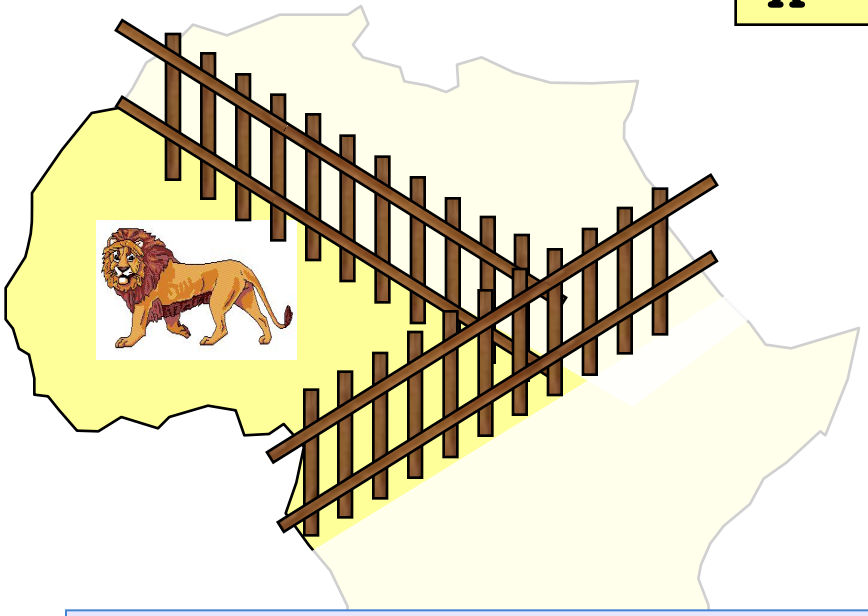
**Удосконалення:** після того, як знайшли **X**, виходимо з циклу.

```
nX = -1;
for ( i = 0; i < N; i ++)
 if ( A[i] == X ) {
    nX = i;
    break;   //вихід з циклу
    }
```

# *Бінарний пошук в упорядкованому масиві*

Стандартний метод пошуку в упорядкованому масиві – це метод поділу відрізка навпіл, причому відрізком є відрізок індексів 1..n. Дійсно, нехай m ($k < m < l$) – деякий індекс. Тоді якщо $A[m] > b$, далі елемент необхідно шукати на відрізку k..m-1, а якщо $A[m] < b$ – на відрізку m+1..l.

# Бінарний пошук



**X = 7**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | **8** | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

X < 8

X > 4

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | **4** | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

X > 6

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | **6** | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

1. Вибрати середній елемент `A[c]` і порівняти з X.
2. Якщо `X = A[c]`, знайшли (вихід).
3. Якщо `X < A[c]`, шукати далі в першій частині.
4. Якщо `X > A[c]`, шукати далі в другій половині.

# Бінарний пошук


```
nX = -1;
L = 0; R = N-1; // межі: шукаєм від A[0] до A[N-1]
while ( R >= L ){
  c = (R + L) / 2;
  if (X == A[c]) {
    nX = c;
    break;
    }
  if (X < A[c]) R = c - 1;
  if (X > A[c]) L = c + 1;
  }
if (nX < 0) printf(«не знайшли...");
else        printf("A[%d]=%d", nX, X);
```

Номер середнього елементу

якщо знайшли

Вийти з циклу

Зсуваємо межі

# *Порівняння методів пошуку*

| | лінійний | двійковий |
|---|---|---|
| **підготовка** | немає | **відсортувати** |
| | **Кількість кроків** | |
| **N = 2** | **2** | **2** |
| **N = 16** | **16** | **5** |
| **N = 1024** | **1024** | **11** |
| **N= 1048576** | **1048576** | **21** |
| **N** | **$\leq$ N** | **$\leq \log_2 N+1$** |

*Усе, що розум усвідомлює й у що вірить, можна здійснити*

*Стів Джобс*

# ***Методи впорядкування даних***

**метод впорядкування вибором;**
**метод попарної перестановки;**
**метод сортування “бульбашки”.**

# ***Метод впорядкування вибором***

**У послідовності $k_1, k_2, k_3 \ldots k_n$ знаходиться максимальний елемент. Максимальний елемент і крайній правий елемент міняються місцями. Після цього правий крайній елемент з подальшого розгляду виключається. На наступному кроці аналізується послідовність $k_1, k_2, k_3 \ldots k_{n-1}$, в якій також знаходиться максимальний елемент. Цей елемент і елемент міняються місцями. Далі аналогічно до попереднього виконуються дії в послідовності $k_1, k_2, k_3 \ldots k_{n-2}$, потім в послідовності $k_1, k_2, k_3 \ldots k_{n-3}$ і т.д. Перевага методу впорядкування вибором полягає в його простоті. Проте цей метод найповільніший, оскільки він не враховує наявності в заданій послідовності можливої впорядкованості деяких елементів**

# ***Метод попарної перестановки***

**Послідовність** $k_1, k_2, k_3, ..., k_n$ **потрібно розмістити в порядку зростання її елементів (послідовність розглядається зліва направо). При цьому порівнюються елементи** $k_1$ і $k_2$, $k_2$ *і* $k_3, ..., k_3$ і $k_4$ **і т. д. Кожний раз, коли попередній елемент більший за наступний, вони міняються місцями. Після першого перегляду всієї послідовності на крайній правій позиції виявиться максимальний елемент. Після цього переглядається послідовність** $k_1, k_2, k_3, ..., k_{n-1}$ **і над її елементами виконуються аналогічні дії, внаслідок чого на позиції *n-1* виявиться другий за величиною елемент і т. д.**

# ***Метод сортування “бульбашки”***

**Його сутність: перший елемент послідовності поступово порівнюється з елементами, що знаходяться справа від нього. Якщо елемент справа менший за перший, то вони міняються місцями. Потім порівнюється перший елемент з тими, що залишилися правіше розглянутого. Якщо знайдеться менший елемент, то вони також міняються місцями. Такий процес продовжується до досягнення правого крайнього елемента. Внаслідок цього на першому місці буде розташовано найменший елемент. Потім аналогічне порівняння здійснюється з 2-го елемента, далі – 3-го і т. д.**