МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ

ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП’ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ЕЛЕКТРОНІКИ

ЗВІТ  
З ПРАКТИЧНОЇ РОБОТИ №3

Виконала:  
студентка групи КН-24-1  
Лабущак А.В.

Перевірив:  
доцент кафедри КІЕ  
Сидоренко В. М.

Кременчук 2025

### Тема. Алгоритми сортування та їх складність. Порівняння алгоритмів сортування

**Мета:**  
Опанувати основні алгоритми сортування та навчитись методам аналізу їх асимптотичної складності.

### Теоретичні відомості

Алгоритми сортування — це методи впорядкування елементів за певним критерієм.  
Найвідоміші: сортування бульбашкою, злиттям, швидке сортування, вставками, вибором.  
Вони відрізняються асимптотичною складністю, стабільністю, ефективністю.

## Індивідуальне завдання (варіант 11)

## 1. Реалізувати бульбашкове сортування. Оцінити складність.

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

for j in range(0, n - i - 1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

· Найгірший випадок (зворотній порядок): **O(n²)**

· Найкращий випадок (відсортовано): **O(n)**

## Оцінити складність сортування злиттям (merge sort) через рекурсію.

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr)//2

L = arr[:mid]

R = arr[mid:]

merge\_sort(L)

merge\_sort(R)

i = j = k = 0

while i < len(L) and j < len(R):

if L[i] < R[j]:

arr[k] = L[i]

i += 1

else:

arr[k] = R[j]

j += 1

k += 1

while i < len(L):

arr[k] = L[i]

i += 1

k += 1

while j < len(R):

arr[k] = R[j]

j += 1

k += 1

· У всіх випадках (кращий, середній, гірший): **O(n log n)**

· Основна теорема рекурсії: T(n) = 2T(n/2) + O(n) → O(n log n)

### Реалізувати швидке сортування. Оцінити складність.

def quick\_sort(arr):

if len(arr) <= 1:

return arr

pivot = arr[0]

left = [x for x in arr[1:] if x < pivot]

right = [x for x in arr[1:] if x >= pivot]

return quick\_sort(left) + [pivot] + quick\_sort(right)

· Середній випадок: **O(n log n)**

· Найгірший випадок (поганий вибір pivot): **O(n²)**

## Контрольні запитання

1. **Що таке асимптотична складність алгоритму сортування і чому вона важлива?**  
   Це оцінка ефективності при великих вхідних даних. Важлива для вибору оптимального алгоритму.
2. **Які алгоритми мають квадратичну складність у найгіршому випадку?**  
   Сортування вставками, вибором, бульбашкою.
3. **У чому перевага сортування злиттям?**  
   Стабільність і гарантована логарифмічна складність у будь-якому випадку.
4. **Які алгоритми використовуються в бібліотеках Python, Java, C++?**  
   Python — Timsort, Java — Dual-Pivot QuickSort, C++ — IntroSort.
5. **У чому різниця між злиттям і швидким сортуванням?**  
   Merge sort — стабільний, однакова складність; Quick sort — нестабільний, але ефективний у середньому.
6. **Як обрати алгоритм сортування для задачі?**  
   Залежно від обсягу даних, стабільності, часу виконання, ресурсоємності.

## Висновки

У результаті практичної роботи були реалізовані та проаналізовані три ключові алгоритми сортування:  
**бульбашкове, злиттям і швидке сортування.**  
Порівняно їхню ефективність за складністю в різних випадках. Отримано практичні навички програмної реалізації та аналітичного оцінювання.