Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Навчально-науковий інститут комп’ютерних систем

Кафедра інформаційних систем

Лабораторна робота № 3

з дисципліни: «Спеціалізована структура даних Heap як основа ефективного нестійкого сортування»

Варіант № 17

Виконав:

Студент групи АІ-243

Кучерявенко О.В.

Перевірили:

Смик С. Ю.

Арсірій О.О.

Одеса 2025

**Мета роботи:**  У роботі розглядаються спеціалізована структура даних купа

(піраміда, Heap), яка є основою для реалізації черги з пріоритетами, а також

алгоритму пірамідального сортування (Heapsort, «Сортування купою») як

прикладу нестійкого сортування

Завдання:

Виконати моделювання та показати графічно фазу створення максимальної

купи (рис.2-4), використовуючи процедуру Sink(A, i, n), підрахуйте кількість

операцій.

Виконати моделювання і показати графічно фазу сортування купою (рис.5,6),

а також фази просіювання (рис.7), використовуючи процедуру Sink(A, 0, n),

підрахуйте кількість операцій.

**Результати виконання завдання:**

Лістинг 1.1 — Псевдокод алгоритму сортування купою

|  |
| --- |
| Алгоритм HeapSort(A):  n = довжина(A)  // Фаза 1: Побудова максимальної купи  for i = floor(n/2)-1 до 0:  Sink(A, i, n)  // Фаза 2: Сортування  for i = n-1 до 1:  обміняти A[0] та A[i]  Sink(A, 0, i)  Процедура Sink(A, i, n):  k = i  while True:  j = 2\*k + 1  if j >= n: break  if j+1 < n і A[j+1] > A[j]:  j = j+1  if A[k] >= A[j]: break  обміняти A[k] та A[j]  k = j |

Лістинг 1.2 — Реалізація на Python з підрахунком операцій

|  |
| --- |
| def swap(arr, i, j):  arr[i], arr[j] = arr[j], arr[i]  def sink(arr, i, n, counts):  k = i  while True:  j = 2 \* k + 1  if j >= n:  break  counts['comparisons'] += 1  if j + 1 < n and arr[j + 1] > arr[j]:  j += 1  counts['comparisons'] += 1  if arr[k] >= arr[j]:  break  swap(arr, k, j)  counts['assignments'] += 3  k = j  def heapsort(arr):  n = len(arr)  counts = {'comparisons': 0, 'assignments': 0}  # Фаза 1: Побудова купи  for i in range(n // 2 - 1, -1, -1):  sink(arr, i, n, counts)  # Фаза 2: Сортування  for i in range(n - 1, 0, -1):  swap(arr, 0, i)  counts['assignments'] += 3  sink(arr, 0, i, counts)  return arr, counts  A = [7, 89, 4, 68, 70, 49, 10, 62, 51]  sorted\_A, counts = heapsort(A.copy())  print("Відсортований масив (Heapsort):", sorted\_A)  print("Порівняння:", counts['comparisons'])  print("Присвоєння:", counts['assignments'])  print("Загальна кількість операцій:", counts['comparisons'] + counts['assignments']) |

Відсортований масив (Heapsort): [4, 7, 10, 49, 51, 62, 68, 70, 89]

Порівняння: 40

Присвоєння: 99

Загальна кількість операцій: 139

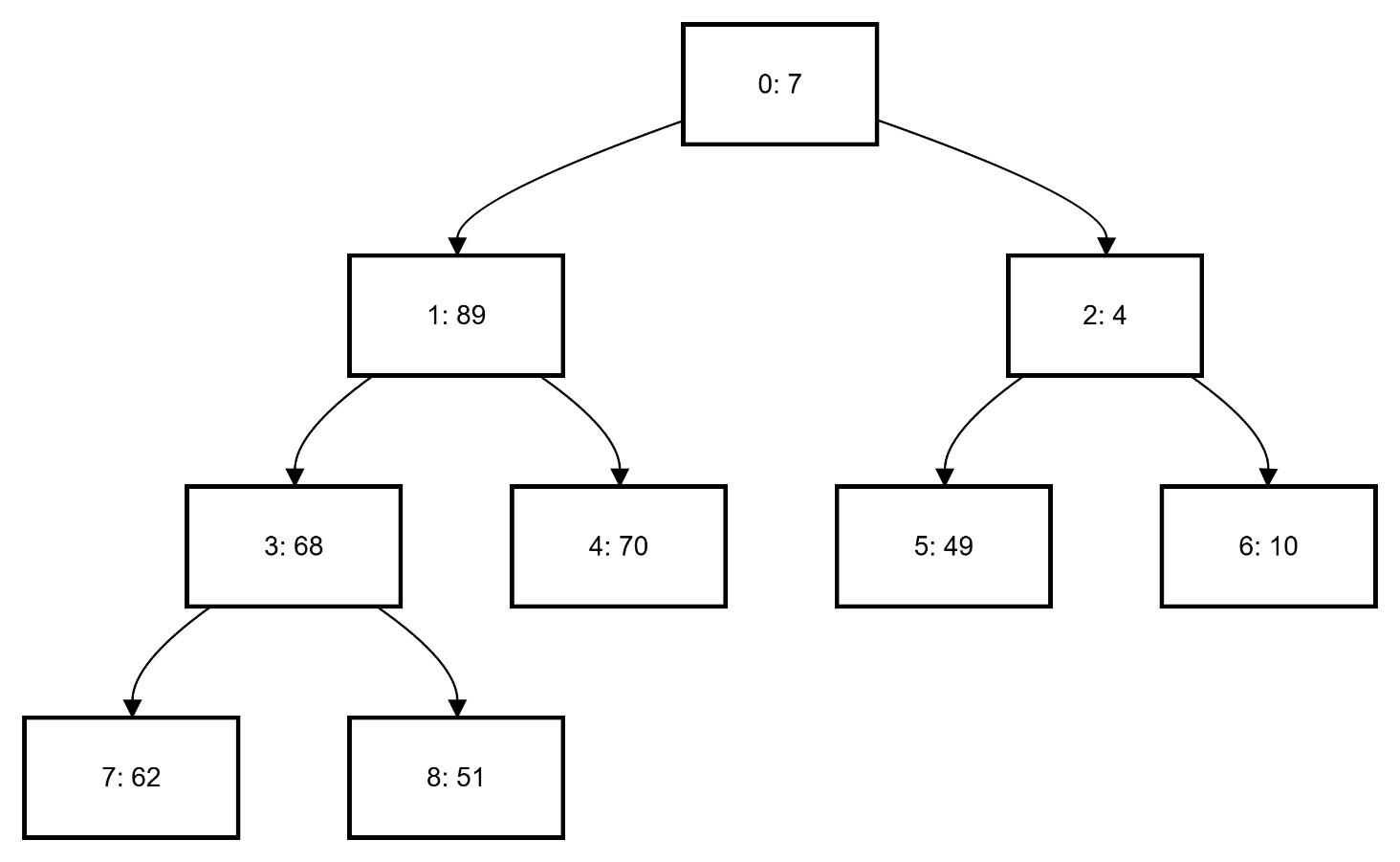
Таблиця 1.1 — Трасування роботи алгоритму Heapsort

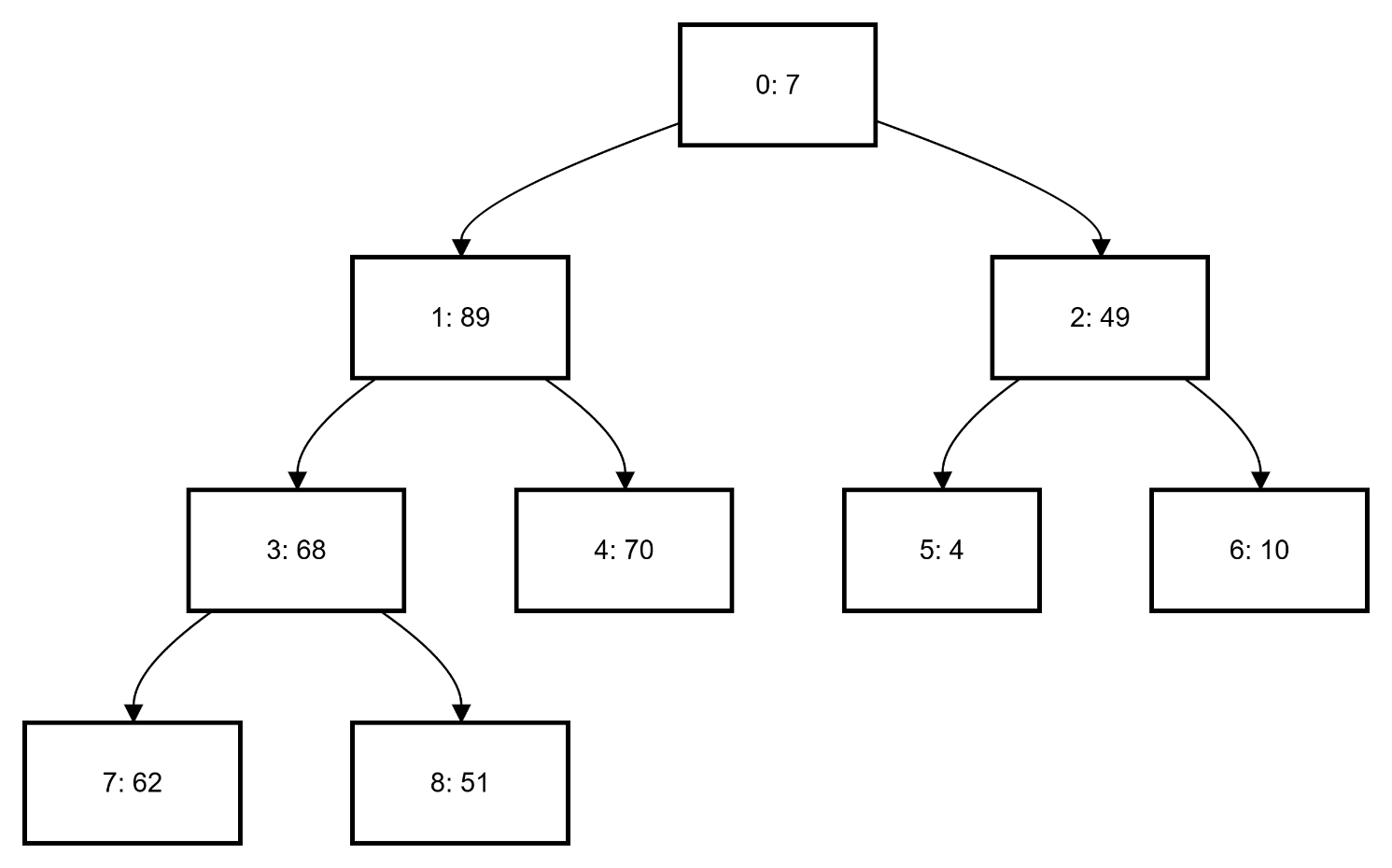
|  |
| --- |
| Початковий масив: [7, 89, 4, 68, 70, 49, 10, 62, 51]  Фаза 1: Побудова максимальної купи  Крок 1: i = 3 (елемент 68)  Піддерево: 68 → 62, 51  68 ≥ 62, 68 ≥ 51 → змін немає  Крок 2: i = 2 (елемент 4)  Піддерево: 4 → 49, 10  4 < 49 → обмін 4 ↔ 49  Масив: [7, 89, 49, 68, 70, 4, 10, 62, 51]  Крок 3: i = 1 (елемент 89)  Піддерево: 89 → 68, 70  89 ≥ 68, 89 ≥ 70 → змін немає  Крок 4: i = 0 (елемент 7)  Піддерево: 7 → 89, 49  7 < 89 → обмін 7 ↔ 89  Масив: [89, 7, 49, 68, 70, 4, 10, 62, 51]  Продовжуємо: 7 → 68, 70  7 < 70 → обмін 7 ↔ 70  Масив: [89, 70, 49, 68, 7, 4, 10, 62, 51]  Проміжний масив після фази 1: [89, 70, 49, 68, 7, 4, 10, 62, 51]  Фаза 2: Сортування  Крок 1: Обмін A[0] (89) ↔ A[8] (51)  Масив: [51, 70, 49, 68, 7, 4, 10, 62, 89]  Sink(A, 0, 8):  51 → 70, 49 → обмін 51 ↔ 70  Масив: [70, 51, 49, 68, 7, 4, 10, 62, 89]  51 → 68, 7 → обмін 51 ↔ 68  Масив: [70, 68, 49, 51, 7, 4, 10, 62, 89]  Крок 2: Обмін A[0] (70) ↔ A[7] (62)  Масив: [62, 68, 49, 51, 7, 4, 10, 70, 89]  Sink(A, 0, 7)  62 → 68, 49 → обмін 62 ↔ 68  Масив: [68, 62, 49, 51, 7, 4, 10, 70, 89]  62 → 51, 7 → змін немає  Крок 3: Обмін A[0] (68) ↔ A[6] (10)  Масив: [10, 62, 49, 51, 7, 4, 68, 70, 89]  Sink(A, 0, 6):  10 → 62, 49 → обмін 10 ↔ 62  Масив: [62, 10, 49, 51, 7, 4, 68, 70, 89]  10 → 51, 7 → обмін 10 ↔ 51  Масив: [62, 51, 49, 10, 7, 4, 68, 70, 89]  Крок 4: Обмін A[0] (62) ↔ A[5] (4)  Масив: [4, 51, 49, 10, 7, 62, 68, 70, 89]  Sink(A, 0, 5):  4 → 51, 49 → обмін 4 ↔ 51  Масив: [51, 4, 49, 10, 7, 62, 68, 70, 89]  4 → 10, 7 → обмін 4 ↔ 10  Масив: [51, 10, 49, 4, 7, 62, 68, 70, 89]  Крок 5: Обмін A[0] (51) ↔ A[4] (7)  Масив: [7, 10, 49, 4, 51, 62, 68, 70, 89]  Sink(A, 0, 4):  7 → 10, 49 → обмін 7 ↔ 49  Масив: [49, 10, 7, 4, 51, 62, 68, 70, 89]  Крок 6: Обмін A[0] (49) ↔ A[3] (4)  Масив: [4, 10, 7, 49, 51, 62, 68, 70, 89]  Sink(A, 0, 3):  4 → 10, 7 → обмін 4 ↔ 10  Масив: [10, 4, 7, 49, 51, 62, 68, 70, 89]  Крок 7: Обмін A[0] (10) ↔ A[2] (7)  Масив: [7, 4, 10, 49, 51, 62, 68, 70, 89]  Sink(A, 0, 2):  7 → 4 → змін немає  Крок 8: Обмін A[0] (7) ↔ A[1] (4)  Масив: [4, 7, 10, 49, 51, 62, 68, 70, 89]  Фінальний відсортований масив: [4, 7, 10, 49, 51, 62, 68, 70, 89] |

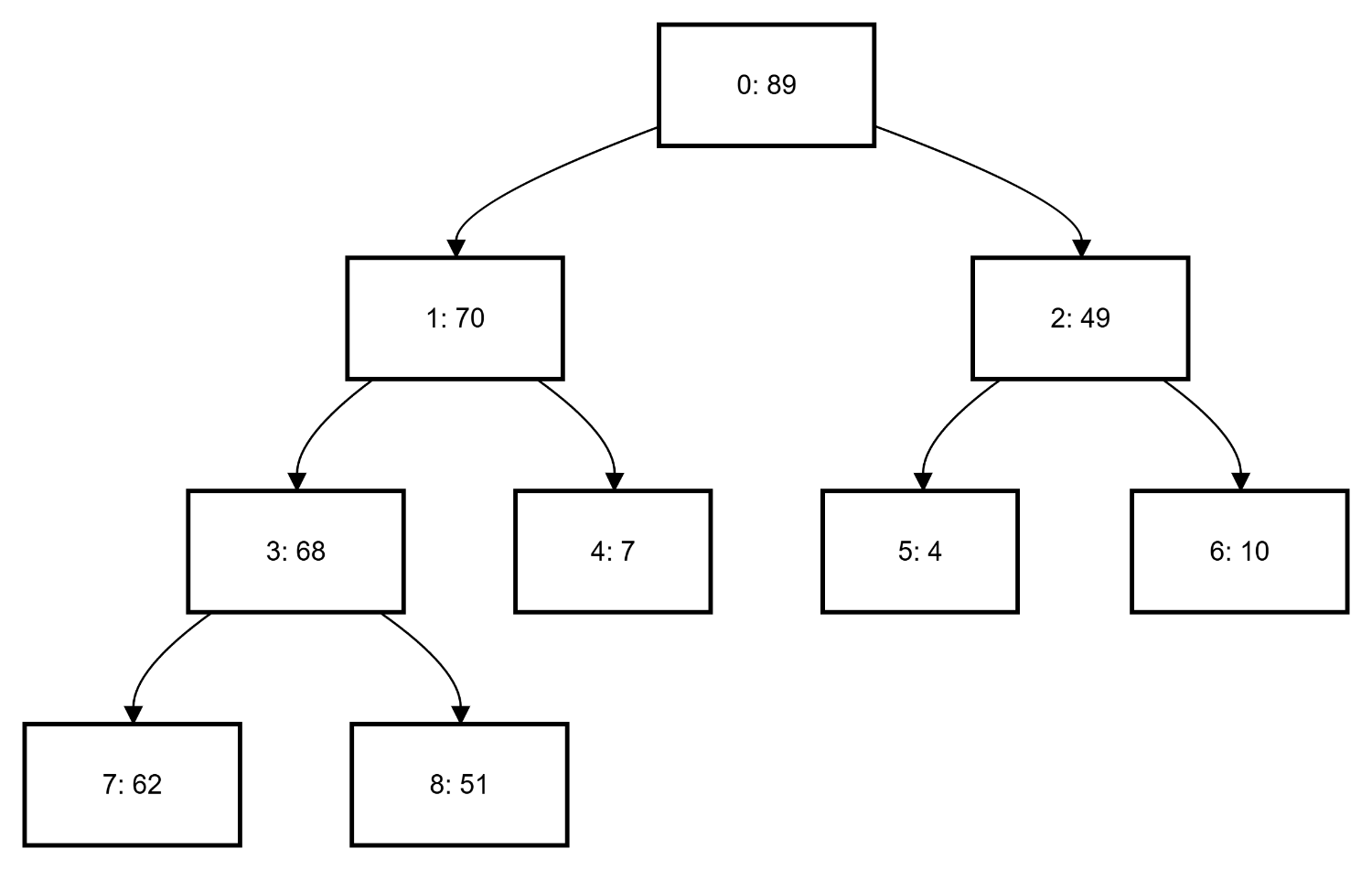
Таблиця 1.2 — Порівняльна таблиця алгоритмів сортування (n=9)

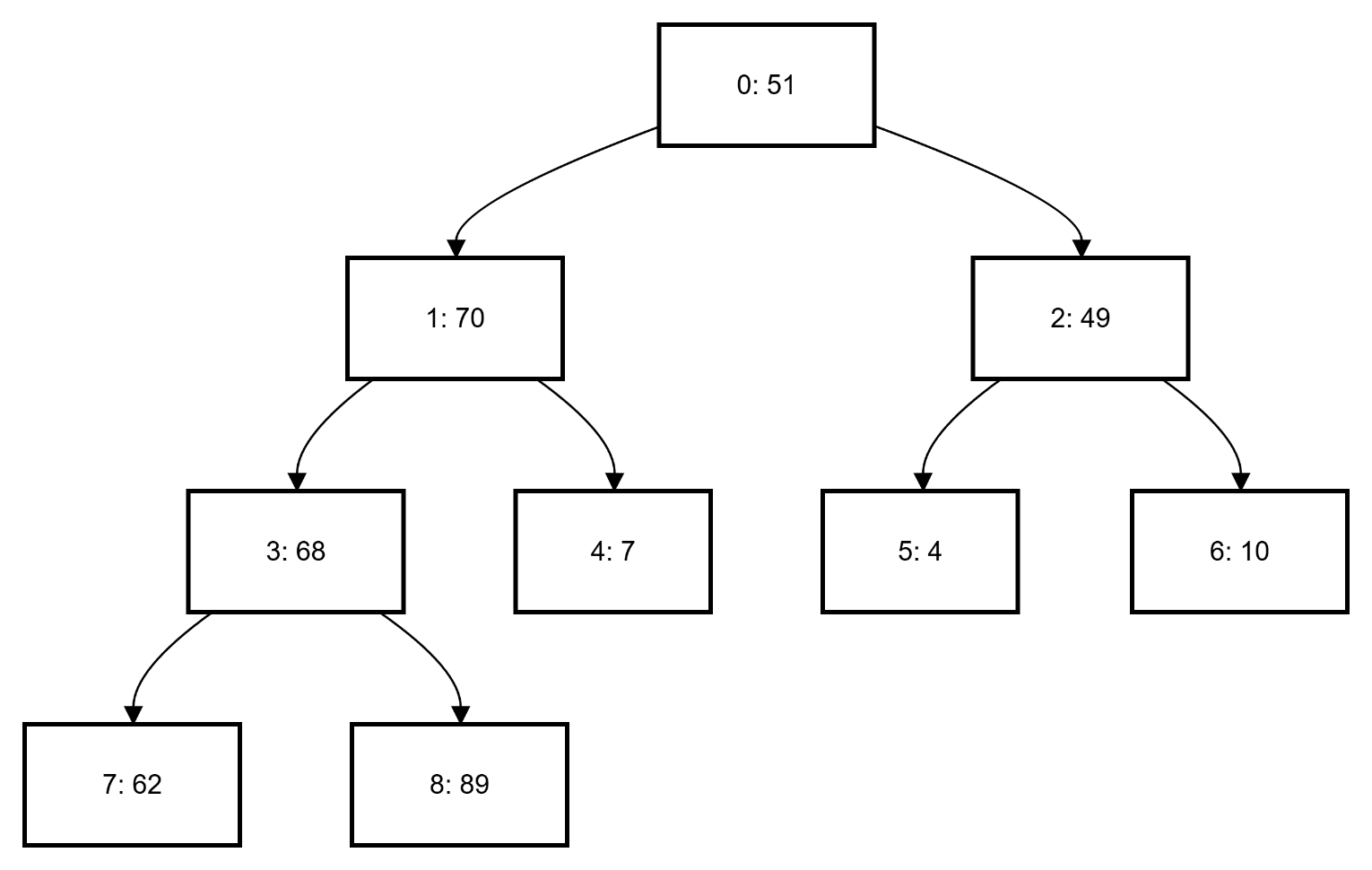
|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Порівняння | Присвоєння | Загалом операцій |
| Selection Sort | 36 | 42 | 78 |
| Insertion Sort | 24 | 59 | 83 |
| Heapsort | 40 | 99 | 139 |

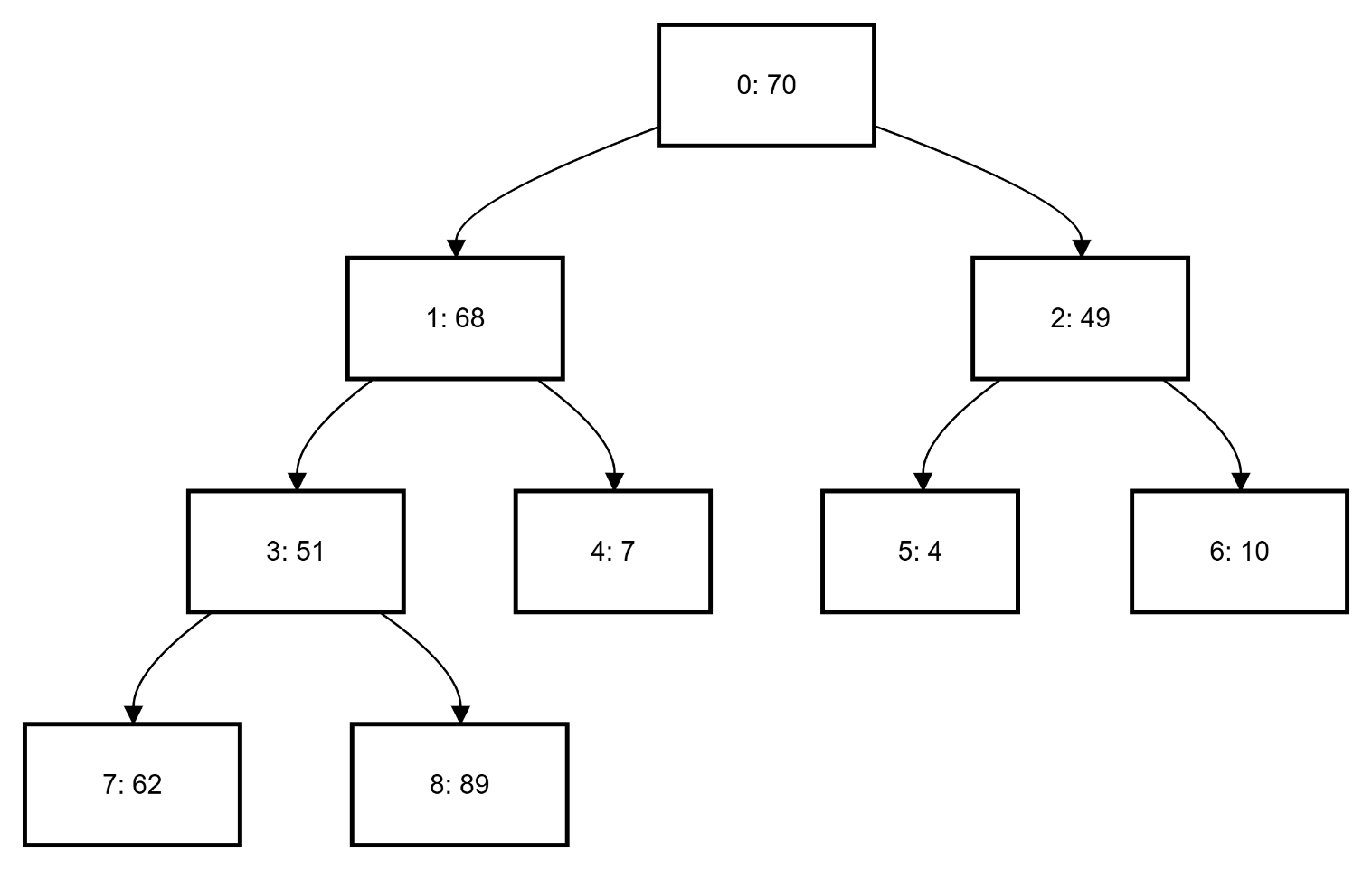
**Візуалізація побудови максимальної купи**

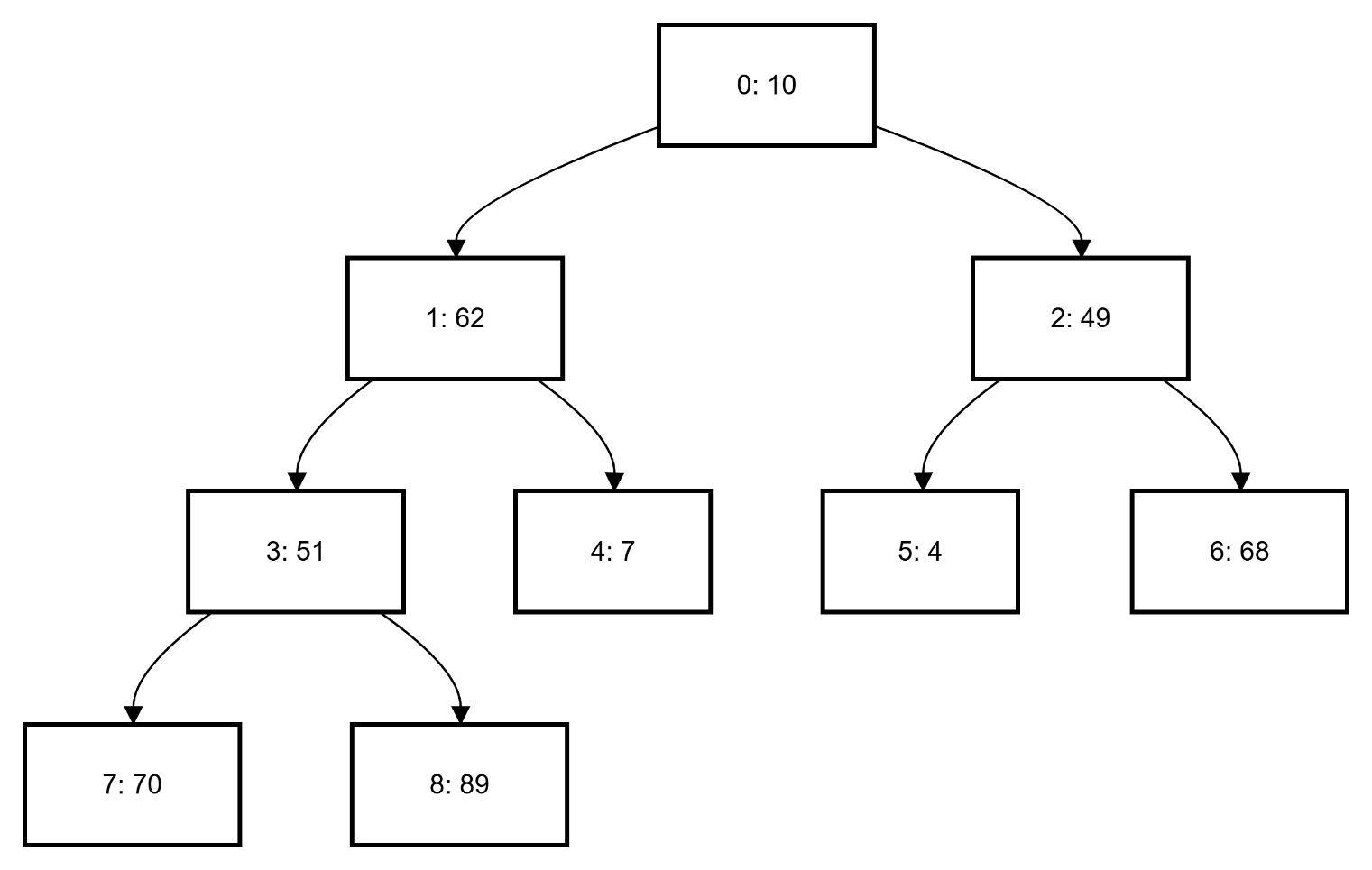
****

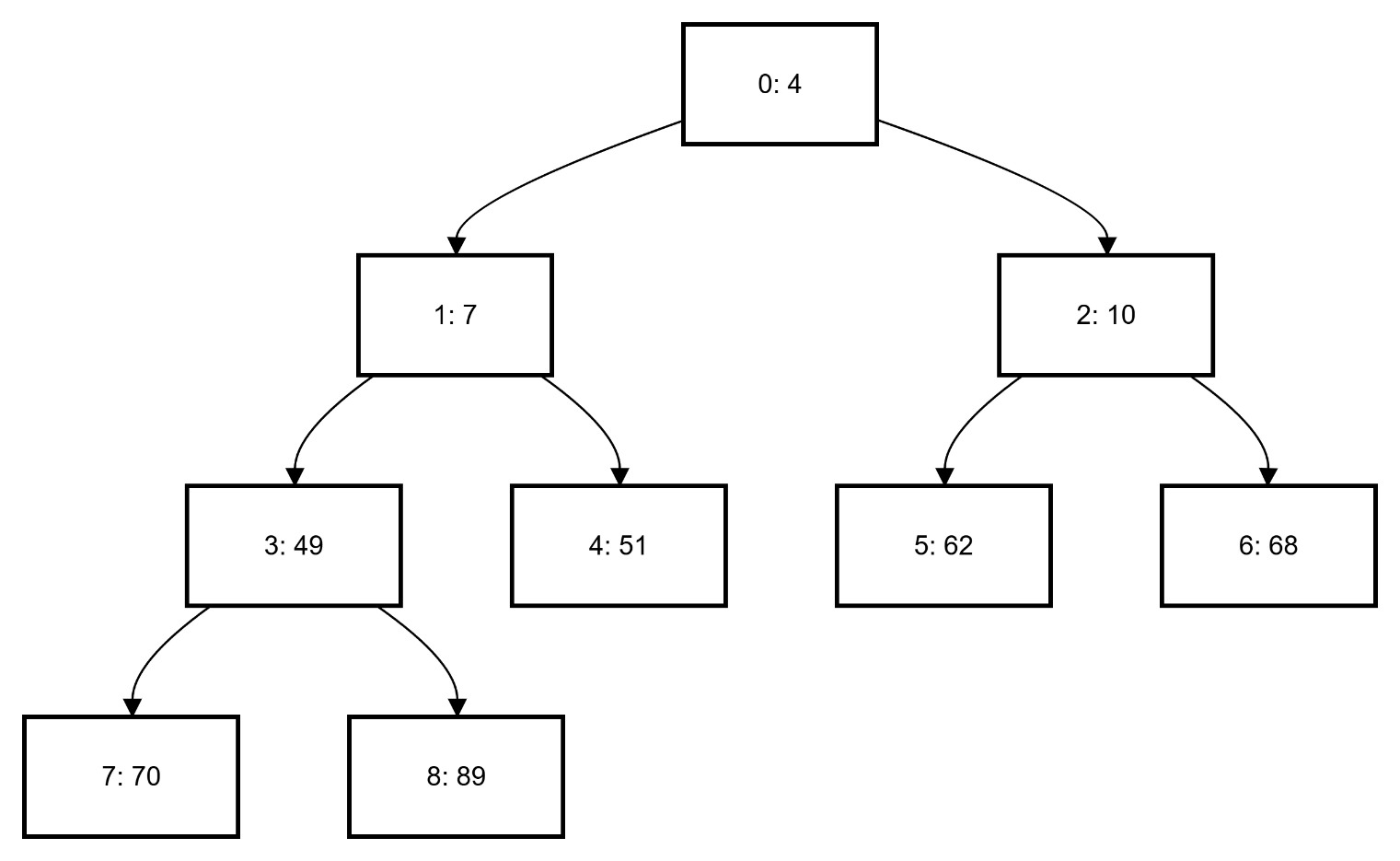












**Висновки:** Heapsort гарантує час виконання O(n log n) навіть у найгіршому випадку, що робить його стійким до поганих вхідних даних.

Проте, для невеликих масивів (n=9) Heapsort виконує більше операцій порівняння та присвоювання, ніж Selection Sort та Insertion Sort.

Insertion Sort виявився найефективнішим за кількістю порівнянь для цих даних, але має квадратичну складність у гіршому випадку.

Selection Sort має меншу загальну кількість операцій, ніж Heapsort, для малих n, але його складність O(n²) робить його неефективним для великих масивів.

Heapsort є оптимальним вибором для великих обсягів даних, коли потрібна гарантована швидкодія без додаткової пам’яті.

**Посилання на GitHub: https://github.com/gutigomd/Algoritms/tree/main/lr3**