Лабораторна робота №3

Спеціалізована структура даних *Heap* як основа ефективного нестійкого сортування

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

Зміст:

1. Теоретичні відомості про купи

Купа (піраміда, *Heap*) – це структура даних, що є об'єктом-масивом, який розглядається як майже повне бінарне дерево. Кожен вузол цього дерева відповідає граничному елементу масиву. На всіх рівнях, крім, можливо, останнього, дерево повністю заповнене (заповнений рівень – це такий, що містить максимально можливу кількість вузлів). Останній рівень заповнюється зліва направо доти, поки в масиві не закінчаться елементи. *Heap* спеціалізована структура даних типу дерево, яка задовольняє *властивості купи:* якщо B є вузлом-нащадком вузла A, то ключ(A) ≥ ключ(B). З цього випливає, що елемент із найбільшим ключем завжди є кореневим вузлом купи, тому іноді такі купи називають *max-купами*.

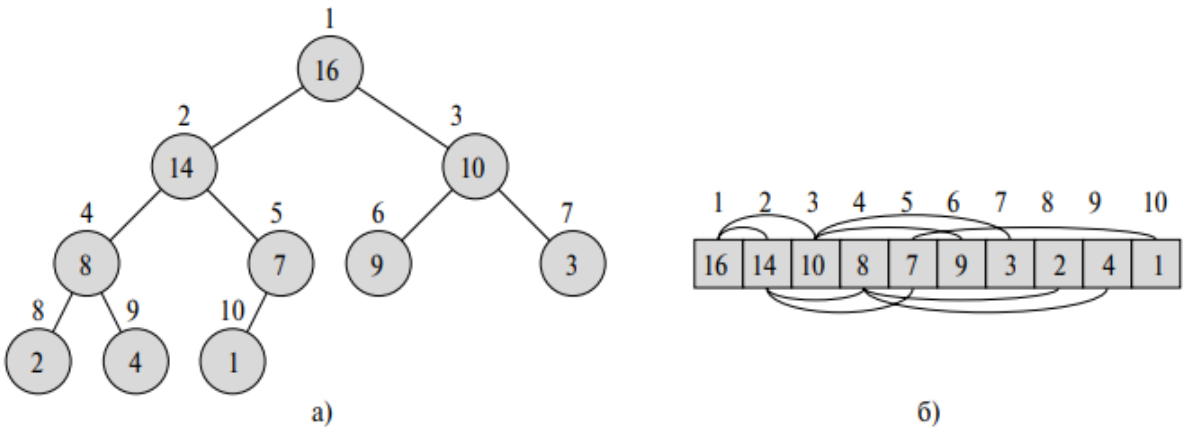

Рисунок 1 – Max-купа представлена у вигляді бінарного дерева та масиву

Представимо масив A у вигляді бінарного дерева, де кожній батьківській вершині A[i] відповідають дочірні вершини A[2i + 1] та A[2*i* + 2] (Рис.1). Якщо для кожного *i* ми маємо *A*[*i*] ≥ *A*[2*i* + 1] та *A*[*i*] ≥ *A*[2*i* + 2], тоді кожна батьківська вершина дерева більша або дорівнює своїм дочірнім вершинам. При цьому значення кореневої вершини дерева більше за всі інші елементи дерева тоді A[0] ≥ A[*i*] для всіх *i* < |A|. Саме таке бінарне дерево називають «максимальною купою».

В якості альтернативи, якщо представлення перевернути, то найменший елемент завжди буде кореневим вузлом, такі купи називають *min-купами*. Купа є максимально ефективною реалізацією абстрактного типу даних, яка називається *чергою із пріоритетом*. Купи мають вирішальне значення в деяких ефективних алгоритмах на графах, таких як алгоритм Дейкстри на d-купах і сортування методом піраміди.

2. Представлення масиву у вигляді максимальної купи, зв'язок з чергами з пріоритетом

На рисунку 2 показано представлення масиву *A* з 14-ти елементів у вигляді бінарного дерева

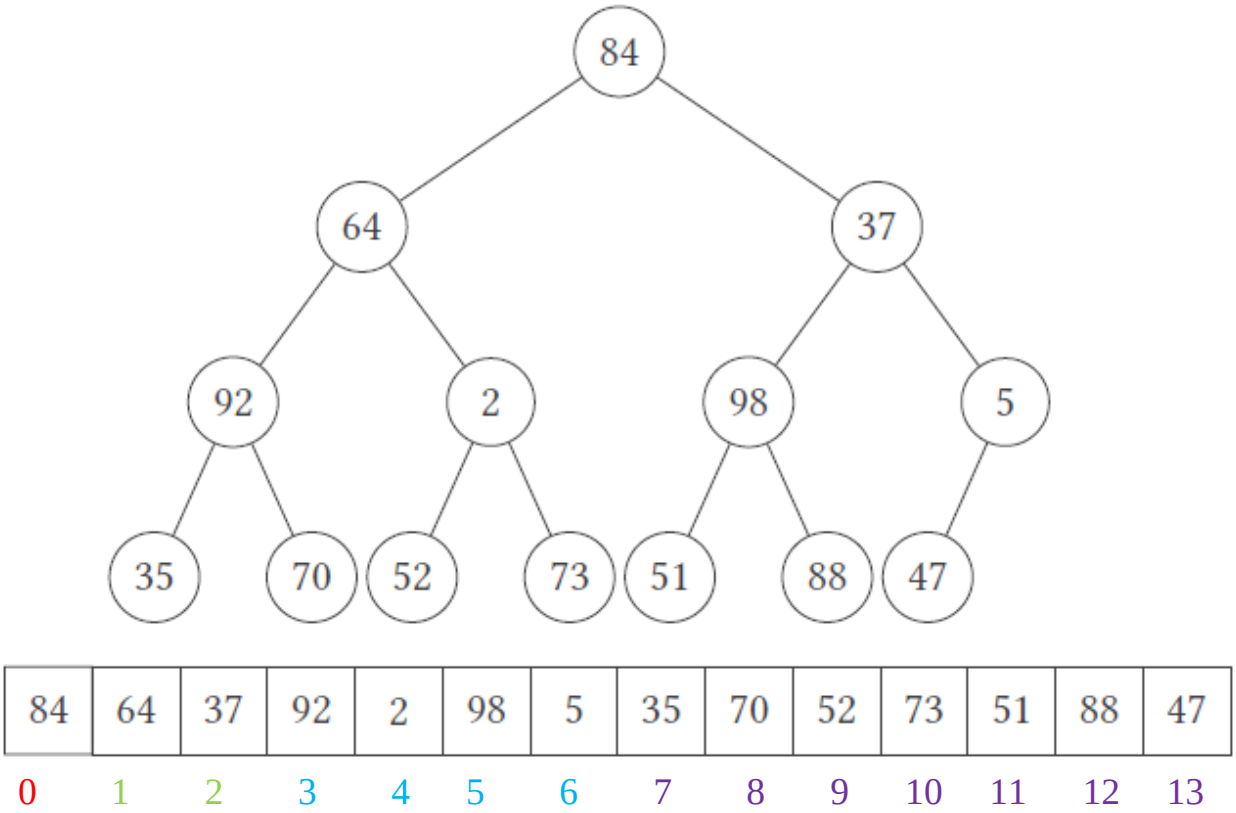

Рисунок 2 – Представлення елементів масиву у вигляді вершин та ребер дерева.

На цьому прикладі розглянемо більш детально питання представлення масиву (дерева) *A* у вигляді максимальної купи

Якщо *останній* рівень дерева – це рівень h, ми починаємо з батьківських вершин рівня *h* – 1, перебираючи їх *справа наліво*. Ми порівнюємо кожну вершину-батька з її дочірніми вершинами. Якщо значення батьківської вершини менше, тоді міняємо її місцями із найбільшою з її дочірніх вершин.

Коли ми закінчуємо з рівнем h – 1, знаємо, що для всіх вузлів даного рівня у нас A[*i*] ≥ A[2*i* + 1] та A[*i*] ≥ A[2*i* + 2] (Рис.3). Сірим кольором відзначені вершини, які поміняли місцями. Дерево, яке ми отримали у результаті, зображено праворуч. Під кожним деревом знаходиться масив A, в якому відбуваються справжні перестановки, тому що насправді існує лише масив, а дерево – лише наочна візуалізація.

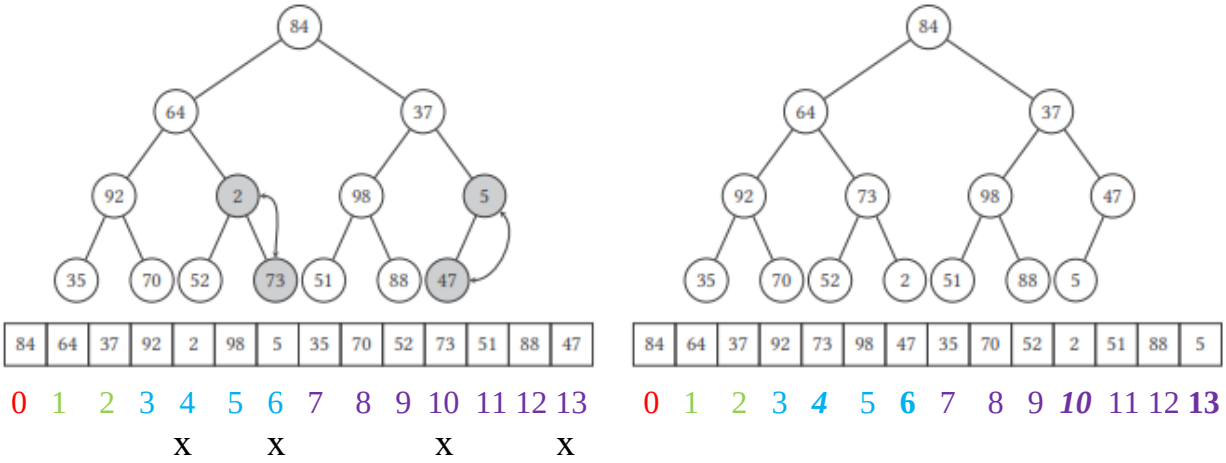

Рисунок 3 – Візуалізація створення максимальної купи на *h* – 1 рівні.

Далі повторюємо ту саму операцію з рівнем *h – 2*, проте цього разу, якщо ми здійснюємо заміну, потрібно перевірити, що відбувається з нижчим рівнем, тобто з *h – 1* рівнем (Рис. 4). Змінюємо місцями вершини 37 і 98. Якщо ми їх змінюємо, то дізнаємося, що 37 мають дочірні вершини, 51 і 88, які не відповідають нашій умові. Ми можемо це виправити, вчинивши, як минулого разу: знайдемо найбільшу з 51 та 88 і поміняємо її місцями з 37. Ті самі зміни відбуваються з вершиною 64, яка опускається до вершини 70.

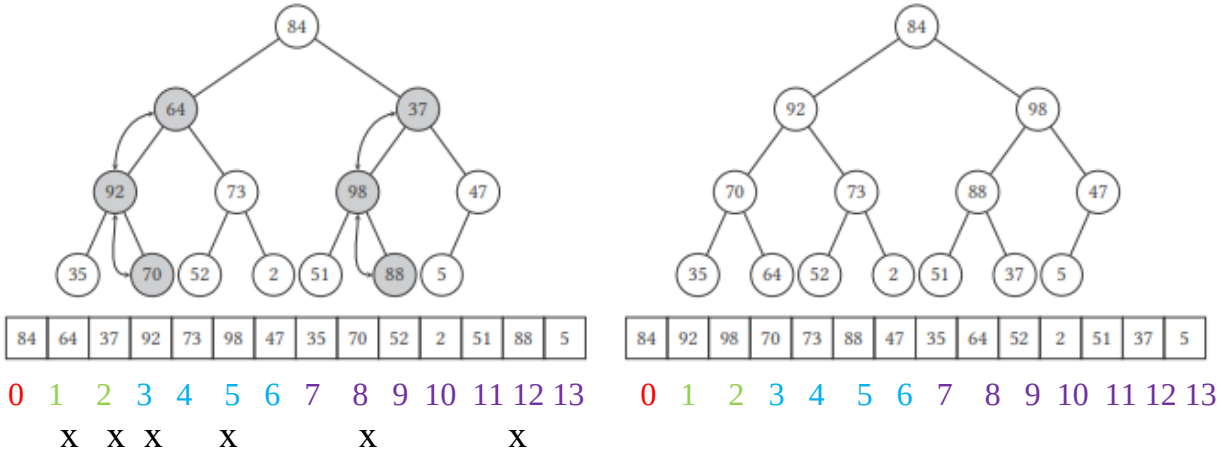

Рисунок 4 – Візуалізація створення максимальної купи на *h – 2* рівні

Нарешті ми дісталися кореневого рівня. Нам треба поміняти місцями 84 та 98. Після цього нам треба поміняти 84 та 88 (Рис. 5).

Простеження, способу переміщення елементів по конструкції максимальної купи, дозволяє назвати його *зануренням* (*просіюванням*) елементів зі свого колишнього місця на потрібний рівень. Вони опустилися на рівень нижче, тоді як вершини, чиє місце вони зайняли, піднялися нагору і зайняли належні

їм місця. Тобто псевдокод побудови максимальної купи на базі описаної процедури занурення показаний в таблиці 1

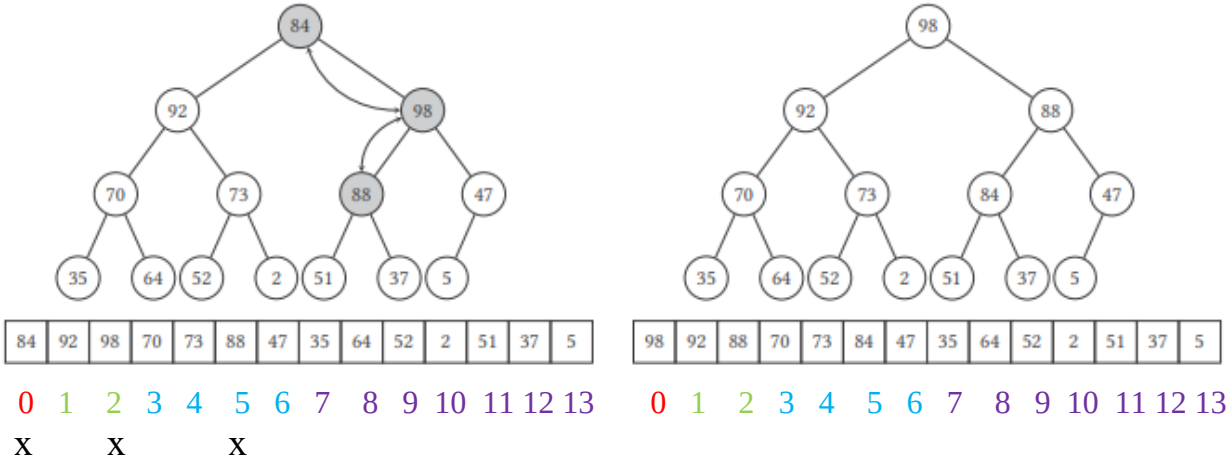

Рисунок 5 – Візуалізація створення максимальної купи на *h – 3* (кореневому) рівні

Таблиця 1 Приклад псевдокоду алгоритму створення максимальної купи на базі процедури занурення

```
Heap_Max (A)
// Вхідні дані: не відсортований масив A
// Вихідні дані: масив A, який є максимальною купою
    n ← |A|
    // Починаємо з першого елемента, що має дочірні вузли та рухаємося до
    // початку масиву.
    for i ← floor((n / 2) − 1) down to 0 do
        Sink(A, i, n)
```

```
Sink(A, i, n)
// Вхідні дані: масив A, індекс кореня піддерева i,
// n – розмір купи
// Вихідні дані: масив A, де елемент A[i] занурився на потрібне місце

    k ← i
    while True do
        j ← 2 * k + 1  // Індекс лівого дочірнього елемента

        // Перевіряємо, чи існує лівий дочірній елемент
        if j >= n then
            break

        // Знаходимо індекс найбільшого дочірнього елемента
        if j + 1 < n and A[j + 1] > A[j] then
            j ← j + 1

        // Порівнюємо поточний елемент з найбільшим дочірнім
        if A[k] >= A[j] then
            break // Елемент на своєму місці, вихід з циклу
        else
            Swap(A[k], A[j])
            k ← j
```

Далі надаємо деякі пояснення до псевдокоду.
Цикл for в псевдокоді алгоритму Heap_Max починається із floor((n / 2) − 1), де «floor» означає округлення числа до найближчого цілого вниз. Це потрібно для розрахування індексу першого елемента в масиві, який має дочірні елементи у представленні купи. В алгоритмі Heap_Max ми починаємо будувати купу з *останнього батьківського вузла* і рухаємося до першого. Оскільки дочірні вузли елемента з індексом і знаходяться за формулою 2*i+1 та 2*i+2, батьківський вузол для елемента з індексом j можна знайти за формулою floor((j-1)/2). Наприклад, для масиву з 7 елементів (n=7), останній елемент знаходиться за індексом 6. Його батьківський вузол знаходиться за індексом floor((6-1)/2) = 2. Таким чином, цикл починається з індексу 2, і це є правильним першим кроком для побудови купи.
Умова циклу while в псевдокоді процедури занурення Sink з явним break вказує на продовження циклу, доки не буде досягнуто кінця гілки дерева або елемент не стане більшим за своїх дочірніх. Тобто коли умова A[k] >= A[j] виконується, це означає, що елемент уже на своєму місці, і ми можемо вийти з циклу.
Функція Swap є допоміжною та міняє місцями два елементи в масиві.

Наприкінці зробимо декілька зауважень що до властивостей купи. Розглядаючи купу як дерево, визначимо висоту її вузла як число ребер у найдовшому простому низхідному шляху від цього вузла до якогось листя дерева. *Висота купи визначається як висота її кореня*. Оскільки n-елементна купа будується за принципом повного бінарного дерева, то її висота дорівнює Θ(lg n). При цьому час виконання основних операцій у купі приблизно пропорційний висоті дерева, тому ці операції вимагають для роботи час O (lg n). Зважаючи на сказане процедура занурення Sink виконується протягом часу O (lg n) і служить підтримки властивості максимальної купи. Час виконання процедури Heap_Max збільшується зі збільшенням кількості елементів лінійно. Ця процедура призначена для створення максимальної купи з невпорядкованого вхідного масиву.
Наприкінці необхідно зауважити, що важливим практичним застосуванням структура даних купа є реалізація черг з пріоритетами.

Черга з пріоритетами (priority queue) – це структура даних, призначена для обслуговування множини S, з кожним елементом якої пов'язано певне значення, що називається ключем (key). У черзі з пріоритетами, що не зростає (максимальна купа), підтримуються наступні операції.
– Операція INSERT(S, x) вставляє елемент x у множину S. Цю операцію
– можна записати як S ← S ∪ {x}.
– Операція MAXIMUM(S) повертає елемент множини S з найбільшим
– ключем.
– Операція EXTRACT_MAX(S) повертає елемент із найбільшим ключем,
– видаляючи його з множини S.
– Операція INCREASE_KEY(S, x, k) збільшує значення ключа, відповідного елементу x, замінивши його ключем зі значенням k. Передбачається, що величина k не менша за поточний ключ елемента x.

3. Сортування масиву за допомогою купи

3.1 Проектування та реалізація мовою Python алгоритму сортування купою

Псевдокод сортування купою показаний в таблиці 2

Таблиця 2 Приклад псевдокоду алгоритму сортування купою на базі процедури занурення Sink(A, i, n)

```
HeapSort(A)
// Вхідні дані: не відсортований масив A
// Вихідні дані: відсортований масив A
   n ← |A|
   // Фаза 1: Побудова максимальної купи
   // Починаємо з першого елемента, що має дочірні вузли,
   // і рухаємося до початку масиву.
   for i ← floor((n / 2) − 1) down to 0 do
      Sink(A, i, n)

   // Фаза 2: Сортування
   // Повторно вилучаємо максимальний елемент і переміщуємо його в кінець масиву.
   for i ← n − 1 down to 1 do
      Swap(A[0], A[i])
      // Відновлюємо властивості купи для зменшеного масиву.
      Sink(A, 0, i)
```

Псевдокод процедури занурення показаний в таблиці 1

Необхідно зауважити, що алгоритм створення максимальної купи і лежить в основі сортувального алгоритму купою *Heapsort*, як прикладу *нестійкого сортування*. На рисунках 3-5 вже було показано перетворення даних масиву A на купу. На рисунку 3 показано, що відбувається з Sink(A, i, |A|), для i = 6, 5, 4, 3; на рисунку 4 показано, що відбувається із Sink(A, i, |A|) для i = 2, 1; і нарешті на рисунку 5 показана ситуація для Sink(A, 0, |A|).

В випадку побудованої мах-купи знаємо, що максимальний з усіх елементів масиву є коренем дерева; він повинен знаходитись у самому кінці, коли всі елементи будуть відсортовані по порядку. Тому дістаємо його з кореня купи і міняємо місцями з останнім елементом купи A[n – 1]. При цьому необхідно пам'ятати, що дерево насправді – це замаскований масив. Тоді перші n – 1 елементів масиву вже не максимальна купа, оскільки відбулася заміна кореневого елемента. Тому необхідно запустити алгоритм занурення для перших n – 1 елементів, щоб занурити щойно змінений місцями A[0] у належне йому місце: Sink(A, 0, i), де i= n − 1. По суті, ми працюємо з купою, яка зменшилася на один елемент. Коли алгоритм занурення завершить роботу, у нас знову буде максимальна купа, нагорі якої буде другий за величиною елемент. Ми тоді можемо поміняти місцями A[0] та останній елемент нинішньої купи, де i= n − 2. Елементи A[n – 2] і A[n – 1]

тоді будуть знаходитися на відповідних їм позиціях, дозволяючи нам продовжити процес, щоб створити з n – 2 елементів, що залишилися, максимальну купу з Sink(A, 0, i) . У кінці всі елементи будуть відсортовані, починаючи з кінця масиву A і до його початку.

Розглянемо принцип роботи HeapSort (A). Фаза побудови купи у сортуванні купою докладно показана на рисунках 3-5, а сортування купою на рисунках 6-8.

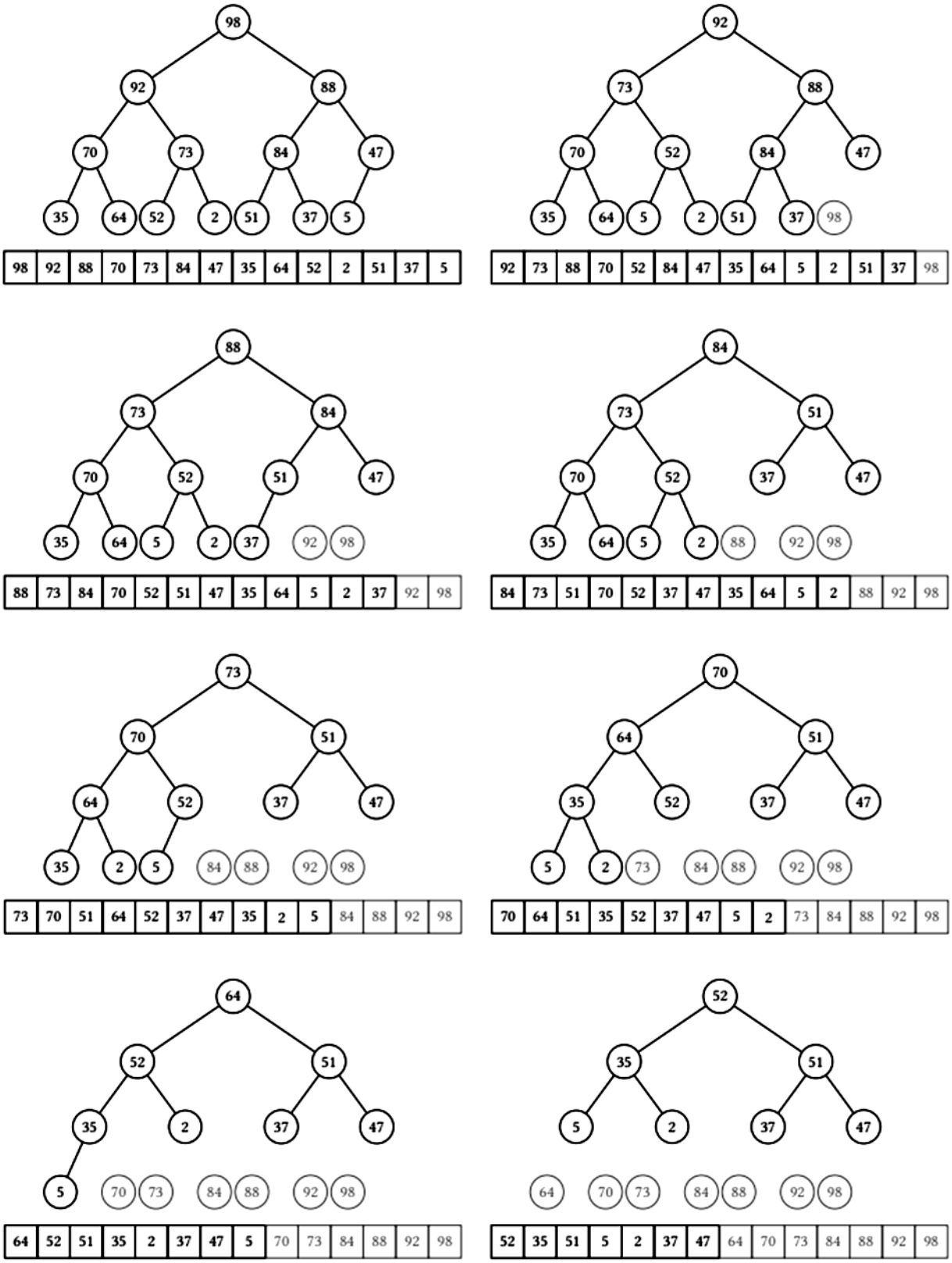

Рисунок 6 – Візуалізація другої фази сортування купою (занурення).

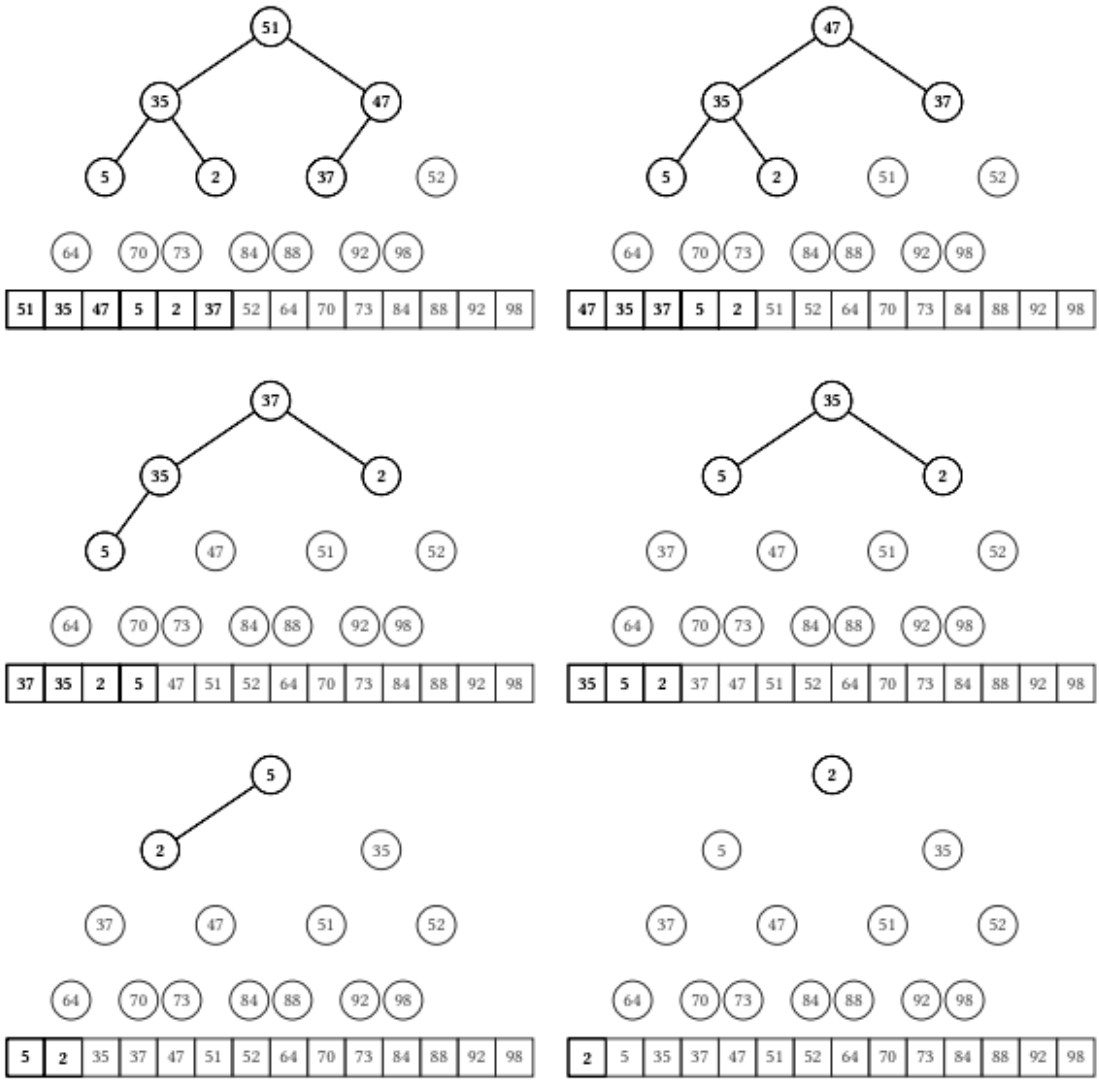

Рисунок 7 – Продовження візуалізації другої фази сортування купою.

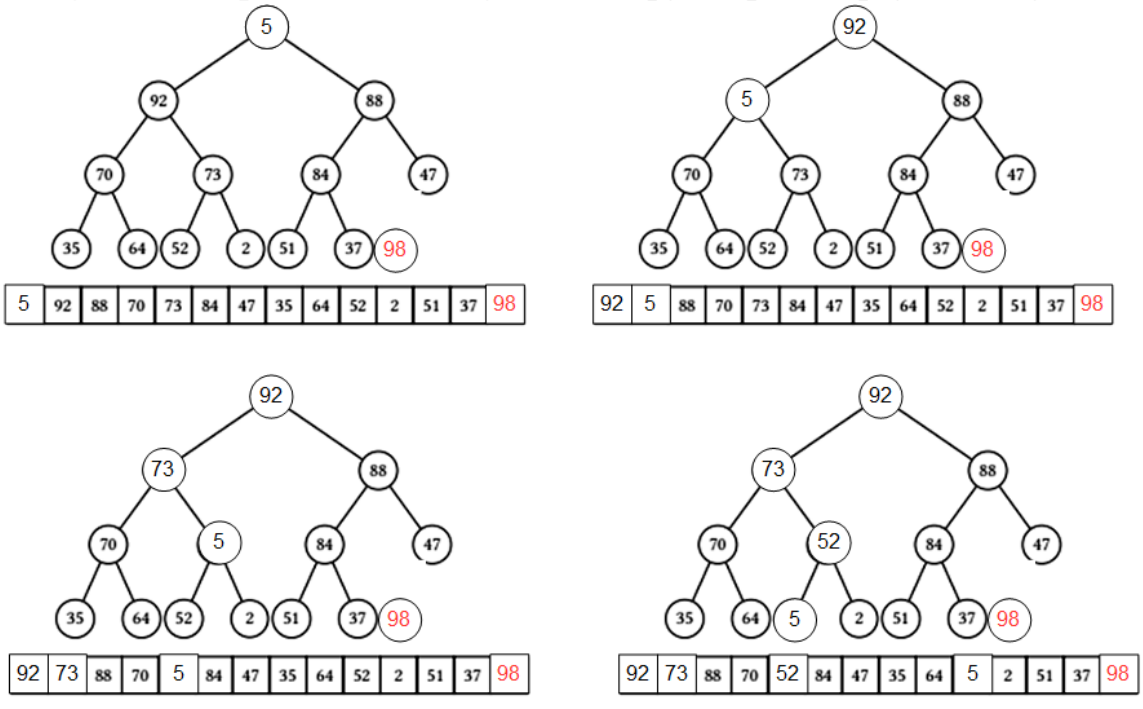

Рисунок 8 – Побудова максимальної купи шляхом занурення для елемента 5.

Надамо пояснення що побудови максимальної купи. Спочатку заносимо в змінну n загальний розмір A і приступаємо до *першої фази* сортування купою, в якій (цикл for) ми перетворюємо масив на максимальну купу за допомогою виклику Sink(A, i, n) n = |A| для всіх i, яким відповідають *внутрішні вершини дерева* i = (n − 1)/2=6 . Наприклад, на рис. 5 це буде вершина зі значенням 47 Останньою вершиною дерева буде дочірня вершина 5, що знаходиться в позиції n – 1. Сама батьківська вершина перебуває в позиції [(n –1)/2]. Значення i, яким відповідають позиції внутрішніх вузлів, відповідно, дорівнюють [(n –1)/2], [(n –1)/2] – 1, …, 0. При цьому у циклах for межі to не включені, тому, щоб отримати 0, область циклу має бути to –1. Після створення купи переходимо до *другої фази* сортування купою, в якій ми запускаємо цикл while. Ми повторюємо цикл кількість разів, що дорівнює кількості елементів у масиві A. Беремо перший елемент, A[0], який є найбільшим з A[0], A[1],…, A[n – 1], міняємо його місцями з A[n – 1], зменшуємо n на один і перебудовуємо купу з перших n – 1 елементів масиву A. У нашому прикладі з A друга фаза сортування купою починається так, як показано на рисунку 6. Кожен елемент береться з вершини купи, тобто з першої позиції A, і поміщається в позиції n – 1, n – 2, …, 1. Щоразу, коли так відбувається, *замінений елемент умовно кладеться нагору купи, а потім занурюється на відповідне йому у дереві місце*. Після цього найбільший з невідсортованих елементів, що залишилися, знову знаходиться в першій позиції A, і ми можемо повторити всю попередню процедуру зі зменшеною купою. На рисунках 6 і 7 ми виділяємо сірим кольором елементи, які вже зайняли свої кінцеві позиції, і прибираємо їх з дерева, щоб підкреслити зменшення купи з кожним циклом. На рисунку 8 показана процедура «занурення», коли замінений елемент (елемент n – 1 зі значенням 5) поміщається у вершину дерева, а далі занурюється вниз на основі порівняння з дочірніми елементами.

Реалізацію алгоритму сортування купою HeapSort(A) мовою Python показано за допомогою Лістингу 1.

Лістинг 1 – Python-код реалізації алгоритму сортування купою.

```
def swap(arr, i, j):
    """Міняє місцями два елементи в масиві."""
    arr[i], arr[j] = arr[j], arr[i]

def sink(arr, i, n):
    """
    Процедура 'занурення' елемента вниз по купі.
    A: масив
    i: індекс поточного елемента
    n: розмір купи
    """
    k = i
    while True:
        j = 2 * k + 1  # Індекс лівого дочірнього елемента
        if j >= n:
            break
```

```
            # Знаходимо індекс найбільшого дочірнього елемента
            if j + 1 < n and arr[j + 1] > arr[j]:
                j += 1

            # Якщо поточний елемент більший або дорівнює найбільшому дочірньому
            if arr[k] >= arr[j]:
                break

            # Міняємо місцями та продовжуємо занурення
            swap(arr, k, j)
            k = j

    def heapsort(arr):
        """
        Алгоритм пірамідального сортування.
        """
        n = len(arr)
        print(f"Початковий масив: {arr}\n")

        # Фаза 1: Побудова максимальної купи
        # Починаємо з першого елемента, що має дочірні елементи.
        print("--- Фаза 1: Побудова максимальної купи ---")
        for i in range(n // 2 - 1, -1, -1):
            print(f"Занурюємо елемент з індексу {i}: {arr[i]}")
            sink(arr, i, n)
        print(f"\nМасив після побудови купи: {arr}\n")

        # Фаза 2: Сортування
        print("--- Фаза 2: Сортування ---")
        for i in range(n - 1, 0, -1):
            # Переносимо найбільший елемент (корінь) в кінець
            print(f"Міняємо місцями корінь ({arr[0]}) та останній елемент ({arr[i]})")
            swap(arr, 0, i)

            # Зменшуємо розмір купи та відновлюємо її властивості
            n -= 1
            print(f"Розмір купи зменшився до {n}. Відновлюємо властивості купи.")
            sink(arr, 0, n)
            print(f"Масив на поточному кроці: {arr}\n")

        return arr

    # Моделювання
    A = [84, 64, 37, 92, 2, 98, 5, 35, 70, 52, 73, 51, 88, 47]
sorted_A = heapsort(A)
print(f"Відсортований масив: {sorted_A}")
```

Результати виконання Python-коду, який наведено в Лістінг1 та Лістінг2 показано в таблиці 3

Таблиця 3 Результати реалізації алгоритму сортування купою

Початковий масив: [84, 64, 37, 92, 2, 98, 5, 35, 70, 52, 73, 51, 88, 47]

--- Фаза 1: Побудова максимальної купи ---
Занурюємо елемент з індексу 6: 5
Занурюємо елемент з індексу 5: 98
Занурюємо елемент з індексу 4: 2
Занурюємо елемент з індексу 3: 92
Занурюємо елемент з індексу 2: 37
Занурюємо елемент з індексу 1: 64
Занурюємо елемент з індексу 0: 84

Масив після побудови купи: [98, 92, 88, 70, 73, 84, 47, 35, 64, 52, 2, 51, 37, 5]

--- Фаза 2: Сортування ---
Міняємо місцями корінь (98) та останній елемент (5)
Розмір купи зменшився до 13. Відновлюємо властивості купи.
Масив на поточному кроці: [92, 73, 88, 70, 52, 84, 47, 35, 64, 5, 2, 51, 37, 98]

Міняємо місцями корінь (92) та останній елемент (37)
Розмір купи зменшився до 12. Відновлюємо властивості купи.
Масив на поточному кроці: [88, 73, 84, 70, 52, 51, 47, 35, 64, 5, 2, 37, 92, 98]

Міняємо місцями корінь (88) та останній елемент (37)
Розмір купи зменшився до 11. Відновлюємо властивості купи.
Масив на поточному кроці: [84, 73, 51, 70, 52, 37, 47, 35, 64, 5, 2, 88, 92, 98]

Міняємо місцями корінь (84) та останній елемент (2)
Розмір купи зменшився до 10. Відновлюємо властивості купи.
Масив на поточному кроці: [73, 70, 51, 64, 52, 37, 47, 35, 2, 5, 84, 88, 92, 98]

Міняємо місцями корінь (73) та останній елемент (5)
Розмір купи зменшився до 9. Відновлюємо властивості купи.
Масив на поточному кроці: [70, 64, 51, 35, 52, 37, 47, 5, 2, 73, 84, 88, 92, 98]

Міняємо місцями корінь (70) та останній елемент (2)
Розмір купи зменшився до 8. Відновлюємо властивості купи.
Масив на поточному кроці: [64, 52, 51, 35, 2, 37, 47, 5, 70, 73, 84, 88, 92, 98]

Міняємо місцями корінь (64) та останній елемент (5)
Розмір купи зменшився до 7. Відновлюємо властивості купи.
Масив на поточному кроці: [52, 35, 51, 5, 2, 37, 47, 64, 70, 73, 84, 88, 92, 98]

Міняємо місцями корінь (52) та останній елемент (47)
Розмір купи зменшився до 6. Відновлюємо властивості купи.
Масив на поточному кроці: [51, 35, 47, 5, 2, 37, 52, 64, 70, 73, 84, 88, 92, 98]

```
Міняємо місцями корінь (51) та останній елемент (37)
Розмір купи зменшився до 5. Відновлюємо властивості купи.
Масив на поточному кроці: [47, 35, 37, 5, 2, 51, 52, 64, 70, 73, 84, 88, 92, 98]

Міняємо місцями корінь (47) та останній елемент (2)
Розмір купи зменшився до 4. Відновлюємо властивості купи.
Масив на поточному кроці: [37, 35, 2, 5, 47, 51, 52, 64, 70, 73, 84, 88, 92, 98]

Міняємо місцями корінь (37) та останній елемент (5)
Розмір купи зменшився до 3. Відновлюємо властивості купи.
Масив на поточному кроці: [35, 5, 2, 37, 47, 51, 52, 64, 70, 73, 84, 88, 92, 98]

Міняємо місцями корінь (35) та останній елемент (2)
Розмір купи зменшився до 2. Відновлюємо властивості купи.
Масив на поточному кроці: [5, 2, 35, 37, 47, 51, 52, 64, 70, 73, 84, 88, 92, 98]

Міняємо місцями корінь (5) та останній елемент (2)
Розмір купи зменшився до 1. Відновлюємо властивості купи.
Масив на поточному кроці: [2, 5, 35, 37, 47, 51, 52, 64, 70, 73, 84, 88, 92, 98]

Відсортований масив: [2, 5, 35, 37, 47, 51, 52, 64, 70, 73, 84, 88, 92, 98]
```

3.2 Трасування алгоритму сортування купою

Для виконання трасування Python-коду (Лістинг 1) необхідно самостійно додати команди print у відповідних місцях коду. В таблиці 4 показано приклад трасування коду за кроками для наступного прикладу $A=[89_0, 45_1, 68_2, 90_3, 29_4, 34_5, 17_6]$ n=7. В таблиці 4 показано результати трасування алгоритму сортування купою, а на рисунках 9 та 10 візуалізацію у вигляді графу результатів трасування двох фаз Python-коду алгоритму сортування купою.

Таблиця 4 Результати трасування алгоритму сортування купою

```
Початковий масив: [89, 45, 68, 90, 29, 34, 17]

--- Фаза 1: Побудова максимальної купи ---
Починаємо 'занурювати' елемент: 68 з індексу 2
  Обираємо лівий дочірній елемент: 34 на індексі 5
  68 (батько) >= 34 (найбільший дочірній). Елемент на своєму місці.
--------------------------------
Починаємо 'занурювати' елемент: 45 з індексу 1
  Обираємо лівий дочірній елемент: 90 на індексі 3
  Міняємо місцями 45 (батько) та 90 (дочірній)
  Масив після обміну: [89, 90, 68, 45, 29, 34, 17]
Елемент 45 на індексі 3 досяг кінця купи. Завершуємо.
--------------------------------
```

```
Починаємо 'занурювати' елемент: 89 з індексу 0
  Обираємо лівий дочірній елемент: 90 на індексі 1
  Міняємо місцями 89 (батько) та 90 (дочірній)
  Масив після обміну: [90, 89, 68, 45, 29, 34, 17]
  Обираємо лівий дочірній елемент: 45 на індексі 3
  89 (батько) >= 45 (найбільший дочірній). Елемент на своєму місці.
--------------------------------

Масив після побудови купи: [90, 89, 68, 45, 29, 34, 17]

--- Фаза 2: Сортування ---
Міняємо місцями корінь (90) та останній елемент (17)
Розмір купи зменшився до 6. Відновлюємо властивості купи.
Починаємо 'занурювати' елемент: 17 з індексу 0
  Обираємо лівий дочірній елемент: 89 на індексі 1
  Міняємо місцями 17 (батько) та 89 (дочірній)
  Масив після обміну: [89, 17, 68, 45, 29, 34, 90]
  Обираємо лівий дочірній елемент: 45 на індексі 3
  Міняємо місцями 17 (батько) та 45 (дочірній)
  Масив після обміну: [89, 45, 68, 17, 29, 34, 90]
Елемент 17 на індексі 3 досяг кінця купи. Завершуємо.
Масив на поточному кроці: [89, 45, 68, 17, 29, 34, 90]

Міняємо місцями корінь (89) та останній елемент (34)
Розмір купи зменшився до 5. Відновлюємо властивості купи.
Починаємо 'занурювати' елемент: 34 з індексу 0
  Порівнюємо 45 (лівий) та 68 (правий). Обираємо 68
  Міняємо місцями 34 (батько) та 68 (дочірній)
  Масив після обміну: [68, 45, 34, 17, 29, 89, 90]
Елемент 34 на індексі 2 досяг кінця купи. Завершуємо.
Масив на поточному кроці: [68, 45, 34, 17, 29, 89, 90]

Міняємо місцями корінь (68) та останній елемент (29)
Розмір купи зменшився до 4. Відновлюємо властивості купи.
Починаємо 'занурювати' елемент: 29 з індексу 0
  Обираємо лівий дочірній елемент: 45 на індексі 1
  Міняємо місцями 29 (батько) та 45 (дочірній)
  Масив після обміну: [45, 29, 34, 17, 68, 89, 90]
  Обираємо лівий дочірній елемент: 17 на індексі 3
  29 (батько) >= 17 (найбільший дочірній). Елемент на своєму місці.
Масив на поточному кроці: [45, 29, 34, 17, 68, 89, 90]

Міняємо місцями корінь (45) та останній елемент (17)
Розмір купи зменшився до 3. Відновлюємо властивості купи.
Починаємо 'занурювати' елемент: 17 з індексу 0
  Порівнюємо 29 (лівий) та 34 (правий). Обираємо 34
```

```
  Міняємо місцями 17 (батько) та 34 (дочірній)
  Масив після обміну: [34, 29, 17, 45, 68, 89, 90]
Елемент 17 на індексі 2 досяг кінця купи. Завершуємо.
Масив на поточному кроці: [34, 29, 17, 45, 68, 89, 90]

Міняємо місцями корінь (34) та останній елемент (17)
Розмір купи зменшився до 2. Відновлюємо властивості купи.
Починаємо 'занурювати' елемент: 17 з індексу 0
  Обираємо лівий дочірній елемент: 29 на індексі 1
  Міняємо місцями 17 (батько) та 29 (дочірній)
  Масив після обміну: [29, 17, 34, 45, 68, 89, 90]
Елемент 17 на індексі 1 досяг кінця купи. Завершуємо.
Масив на поточному кроці: [29, 17, 34, 45, 68, 89, 90]

Міняємо місцями корінь (29) та останній елемент (17)
Розмір купи зменшився до 1. Відновлюємо властивості купи.
Починаємо 'занурювати' елемент: 17 з індексу 0
Елемент 17 на індексі 0 досяг кінця купи. Завершуємо.
Масив на поточному кроці: [17, 29, 34, 45, 68, 89, 90]

Відсортований масив: [17, 29, 34, 45, 68, 89, 90]
```

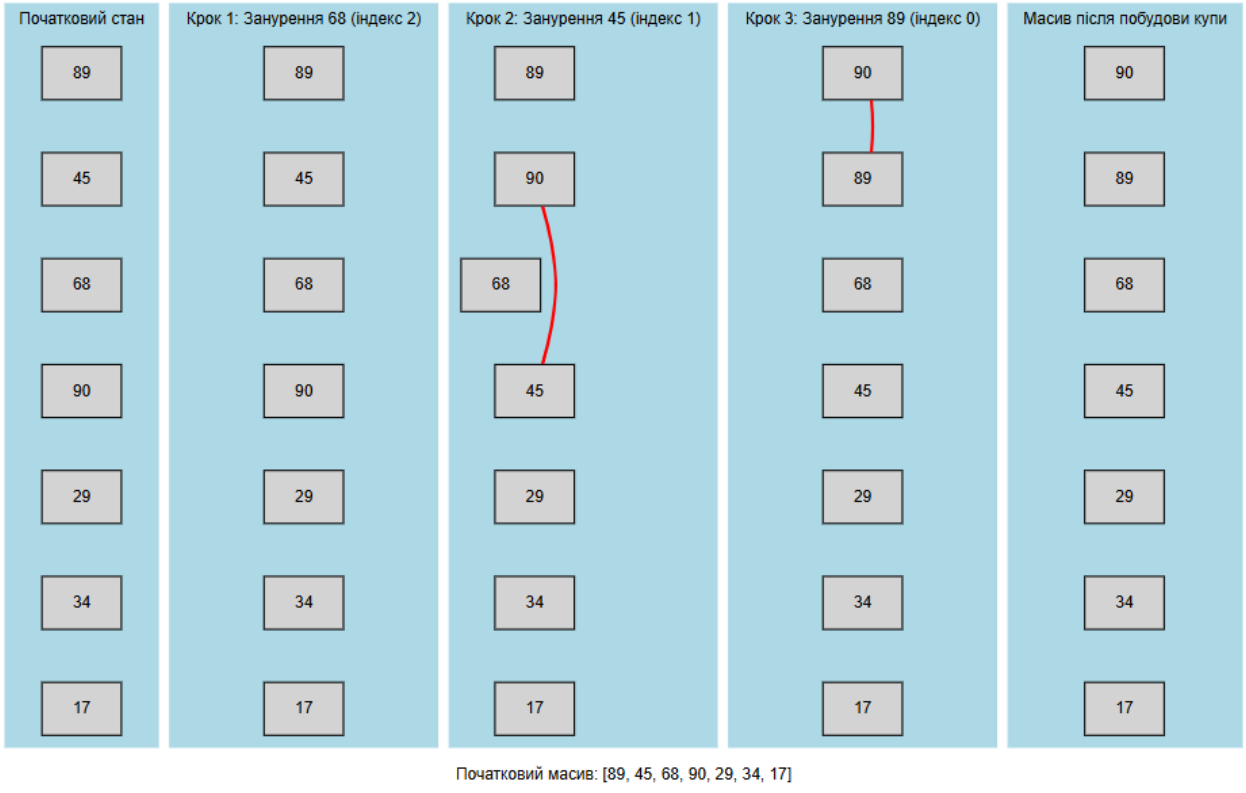

Рисунок 9 – Візуалізація кроків фази побудови максимальної купи алгоритму `heapsort`

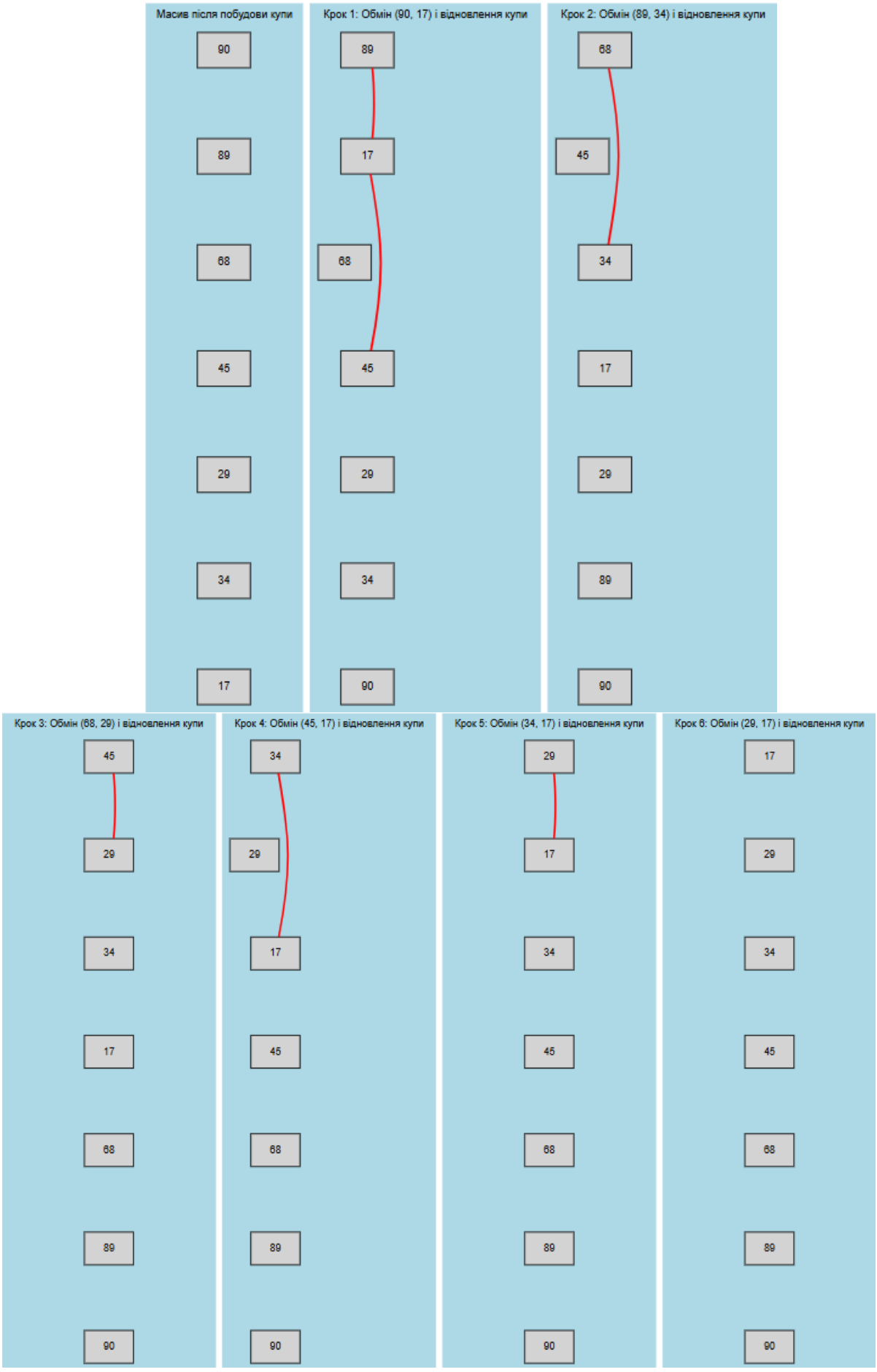

Рисунок 10 – Візуалізація кроків фази сортування алгоритму сортування `heapsort`

4. Порівняльний аналіз нестійких алгоритмів сортування

Завдання:

1. Відсортувати послідовність за варіантами завдань за допомогою сортування купою.

Варіанти завдань:

Варіант	Послідовність
1	38, 15, 50, 99, 41, 52, 47, 65, 95
2	79, 97, 82, 18, 20, 2, 88, 61, 17
3	68, 97, 12, 15, 31, 40, 22, 50, 53
4	35, 95, 16, 33, 28, 76, 27, 10, 5
5	46, 11, 49, 78, 77, 4, 62, 8, 69
6	58, 5, 50, 99, 61, 32, 27, 45, 75
7	41, 68, 67, 10, 7, 69, 95, 43, 98
8	69, 52, 97, 27, 10, 88,29, 1, 24
9	12, 23, 67, 65, 50, 70, 80, 61, 92
10	87, 79, 97, 82,98, 40, 42, 88, 61
11	47, 50, 61, 41,53, 12, 68, 63, 3
12	53, 100, 44, 74, 53, 38, 82, 65, 28
13	90, 10, 15, 80, 100, 6, 57, 5, 29
14	10, 90, 95, 30, 45, 60, 57, 28, 5
15	50, 80, 19, 86, 35, 7, 60, 48, 51
16	54, 65, 7, 33, 86, 29, 11, 91, 12
17	7, 89, 4, 68, 70, 49, 10, 62, 51
18	21, 44, 22, 50, 63, 68, 97, 12, 15
19	80, 27, 37, 36, 91, 53, 86, 66, 98
20	86, 36, 14, 50, 64, 21, 2, 83, 82
21	50, 57, 78, 34, 41, 68, 47, 61, 38
22	19, 75, 43, 31, 5, 66, 62, 34, 76
23	77, 89, 74, 68, 70, 49, 5, 62, 51
24	53, 5, 44, 47, 35, 83, 82, 85, 28
25	11, 42, 67, 55, 65, 78, 25, 50, 69
26	41, 52, 47, 65, 95, 38, 15, 50, 99
27	18, 20, 2, 88, 61, 17, 79, 97, 82
28	31, 40, 22, 50, 53, 68, 97, 12, 15
29	28, 76, 27, 10, 5, 35, 95, 16, 33
30	78, 77, 4, 62, 8, 69, 46, 11, 49
31	61, 32, 27, 45, 75, 58, 5, 50, 99
32	95, 43, 98, 41, 68, 67, 10, 7, 69
33	29, 1, 24, 69, 52, 97, 27, 10, 88
34	70, 80, 61, 92, 12, 23, 67, 65, 50

35	98, 40, 42, 88, 61, 87, 79, 97, 82
36	53, 12, 68, 63, 3, 47, 50, 61, 41

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.

5.Вимоги до представлення звіту

Звіт повинен містити:

1. Титульну частину (лист) з вказівкою на назву лабораторної роботи, ПІБ студента, групу, номер варіанту та послідовність, яку необхідно отсортувати.
2. Псевдокод сортування купою, результат його чисельного моделювання (див. Таблицю 1) за варіантами завдань
3. Результати порівнянь сортування купою (за порівняннями та присвоюваннями) із сортуванням за алгоритмами ЛР1 та ЛР2
4. Висновки