ЛЕКЦІЯ 5

ТЕМА:

Алгоритми сортування

§1. ПОСТАНОВА ЗАДАЧІ СОРТУВАННЯ

Сортування даних – це обробка інформації, в результаті якої її елементи розташовуються в заданій послідовності в залежності від значення деяких ознак елементів цієї інформації.

Задача сортування полягає в перестановці елементів послідовності у визначеному порядку. Впорядкування здійснюється в процесі багаторазового перегляду вхідного масиву.

КОЖНИЙ АЛГОРИТМ СОРТУВАННЯ МОЖНА РОЗБИТИ НА ТРИ ЧАСТИНИ:

- порівняння, що визначає впорядкованість пари елементів;
- перестановку, що міняє місцями пару елементів;
- алгоритм сортування, який виконує порівняння і перестановку до тих пір, доки всі елементи послідовності не будуть впорядковані.

Методи сортування :

- **внутрішнє** *сортування*, коли працюють з даними в оперативній пам’яті з довільним доступом;
- **зовнішнє** *сортування*, коли впорядковують інформацію, розташовану на зовнішніх носіях.

Ефективність алгоритму залежить від наступних факторів:

- скільки елементів необхідно відсортувати;
- в якій степені елементи вже відсортовані;
- який діапазон і розподілення значень елементів, що сортуються;
- записані елементи в файл або в масив, тощо.

ПАРАМЕТРИ, ЯКІ ВПЛИВАЮТЬ НА ОЦІНКУ АЛГОРИТМІВ СОРТУВАННЯ :

- **час сортування** – основний параметр, що характеризує швидкість роботи алгоритму;
- **пам'ять** – ряд алгоритмів вимагає виділення додаткової пам'яті під тимчасове зберігання даних;
- **стійкість** – сортування не змінює взаємного положення рівних елементів. Така властивість може бути корисною, коли всі елементи складаються з деяких полів, а сортування проводиться по одному з них;
- **природність поведінки** – ефективність метода при обробці вже відсортованих або частково відсортованих даних. Алгоритм веде себе природно, якщо враховує цю характеристику вхідної послідовності.

Приклад: Відсортувати масив, враховуючи стійкість сортування.

Заданий масив:

1	a		3	q		1	b		1	c		2	z

Відсортований масив:

1	a		1	b		1	c		2	z		3	q

Взаємне положення елементів з однаковим ключем 1 і додатковими полями a, b, c залишилися без змін.

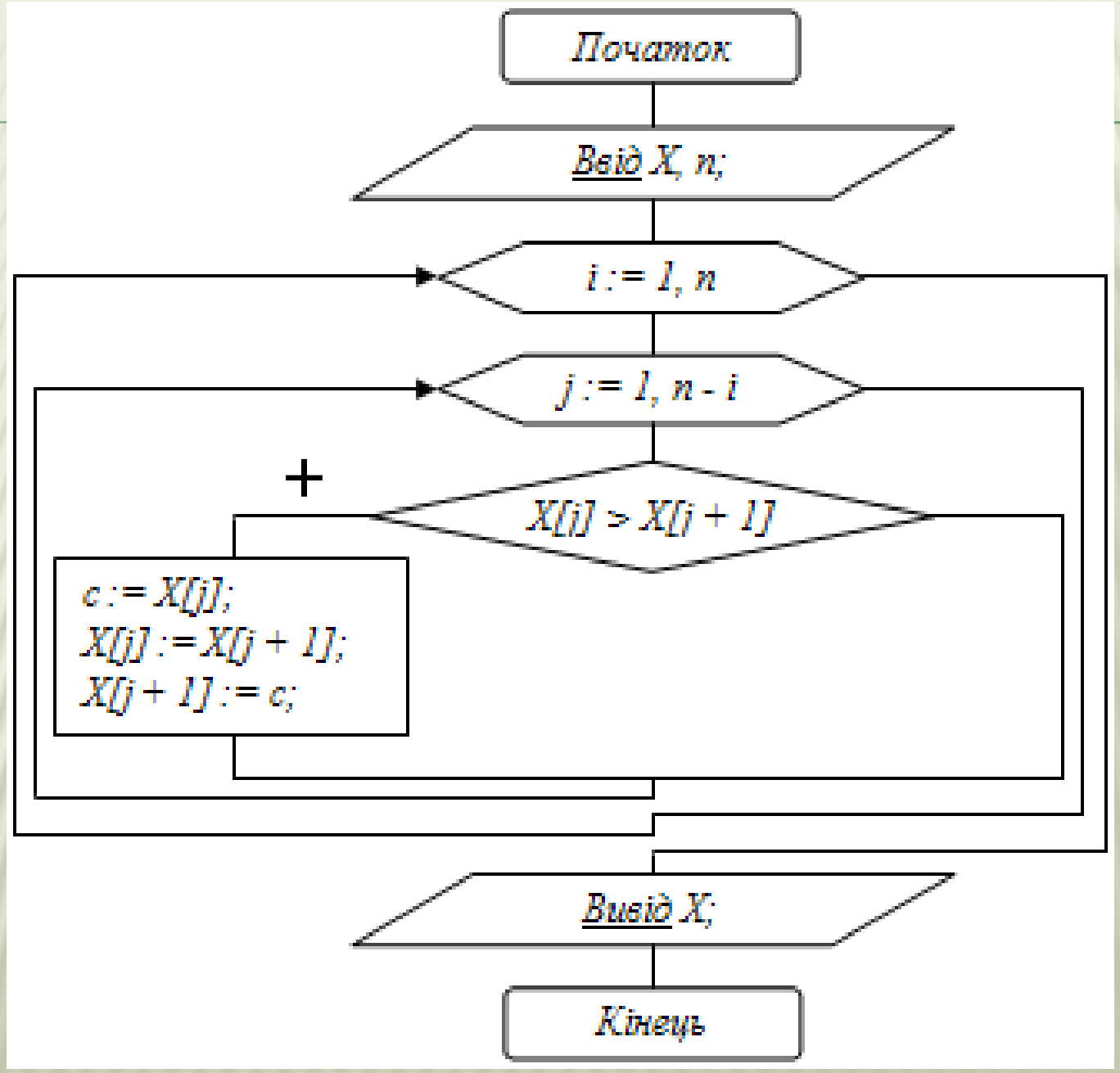
§2. ОСНОВНІ СХЕМИ ВНУТРІШНЬОГО СОРТУВАННЯ

2.1 Сортування обміном (бульбашкою)

Ідея методу: найлегші елементи масиву підіймаються вгору, а найважчі – тонуть.

Ми переглядаємо весь масив “знизу вверх ” і міняємо місцями поряд розташовані елементи, якщо нижній елемент менше за верхній. Таким чином ми виштовхуємо наверх самий легкий елемент масиву. Потім повторюємо цю операцію для n -1 елементів, що залишилися.

Алгоритм дуже простий, але дуже повільний (середнє число порівнянь і обмінів n^2).


Початок
Ввід X, n;
i := 1, n
j := 1, n - i
+
X[j] > X[j + 1]
c := X[j];
X[j] := X[j + 1];
X[j + 1] := c;
Вивід X;
Кінець

Приклад: Відсортувати масив: 8,2,6,7,4,0.

8		0		0		0		0	
2		8		2		2		2	
6		2		8		4		4	
7		6		4		8		6	
4		7		6		6		8	
0		4		7		7		8	

Алгоритм дуже простий, але дуже повільний, його складність $O(n^2)$, не використовує додаткову пам’ять, має природну поведінку, стійкий.

2.2 Сортування вибором

Ідея методу полягає в тому, щоб створювати відсортовану послідовність шляхом приєднання до неї одного елемента за іншим в правильному порядку.

Будемо будувати послідовність, починаючи з лівого кінця масиву. Алгоритм складається з *n* послідовних кроків від 0 до *n*-1. На *i*-му кроці обираємо найменший з елементів *A*[*i*]....*A*[*n*] і міняємо його місцем з *A*[*i*].

Приклад: Відсортувати масив 9,6,2,1,4,5 вибором.

9 6 2 1 4 5

1 6 2 9 4 5

1 2 6 9 4 5

1 2 4 9 6 5

1 2 4 5 6 9

1 2 4 5 6 9

Таким чином на (*n*-1)-му кроці вся послідовність, крім *a*[*n*], буде відсортованою , а *a*[*n*] стоїть на останньому місці: всі менші елементи зліва від нього.

2.3 Сортування вставками

Ідея методу: на першому кроці впорядковуються два перші елементи масиву. Потім робиться вставка третього елемента у відповідне місце по відношенню до перших двох елементів і т.д.

На *i*-му кроці послідовність розділена на дві частини: впорядковану $A[0]...A[i]$ і невпорядковану $A[i+1]...A[n]$.

На наступному $(i+1)$-му кроці алгоритму беремо $A[i+1]$ і вставляємо в потрібне місце у впорядковану частину.

Пошук відповідного місця для наступного елемента здійснюється шляхом послідовних порівнянь з елементами, що розташовані перед ним. В залежності від результату порівняння елементи або залишаються на поточних місцях (вставку завершено), або міняються місцями і процес продовжується.

Приклад: Відсортувати масив 9,6,2,1,4,5 вставками.

9 6 2 1 4 5 → 6 9 2 1 4 5 → (6 2 9 1 4 5)

2 6 9 1 4 5 → 1 2 6 9 4 5

1 2 4 6 9 5 → 1 2 4 5 6 9

2.4 Метод Шелла

Ідея методу: спочатку прибирають «масовий безлад» в масиві, порівнюючи далеко розташовані один від одного елементи, поступово зменшуючи інтервал між ними до одиниці. Це означає, що на останніх стадіях алгоритм зводиться до перестановки сусідніх елементів.

Найчастіше використовують послідовність зміщень 8, 4, 2, 1 (для цього кількість елементів = 2^n), але застосовувати можна будь-яку послідовність (наприклад 7, 5, 3 ,1), головне щоб останнє зміщення було 1.

Приклад: Відсортувати масив 51, 31, 14, 19, 11, 26, 9, 41 методом Шелла.

h = 4 (крок, зміщення)

51 31 14 19 | 11 26 9 41

11 26 9 19 | 51 31 14 41

h=2

11 26 | 9 19 | 51 31 | 14 41

9 19 | 11 26 | 14 31 | 51 41

h=1

9 19 11 26 14 31 51 41

9 11 19 14 26 31 41 51

Якщо після останнього кроку методу Шелла масив повністю не відсортований, використовуємо алгоритм вставками.

2.5 Порівняння часу сортування

2.6 Швидке сортування (метод Хоара, Quick Sort)

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно. Алгоритм швидкого сортування може бути реалізований як у масиві, так і в двозв'язному списку.

$$\underbrace{A[0]\ldots A[s-1]}_{\text{Всі елементи} \le A[s]} \quad A[s] \quad \underbrace{A[s+1]\ldots A[n]}_{\text{Всі елементи} \ge A[s]}$$

Алгоритм:

Крок1. Обираємо ***опорний*** елемент *p* відносно якого буде виконуватися розбиття.

- За опорний елемент можна взяти перший елемент масиву, середній або останній.
- Бажано обрати опорний елемент, який має середнє значення.

Розглянемо найпростішу стратегію і в якості опорного оберемо перший елемент масиву: p = A[*1*].

Крок 2. Встановлюємо індекси *i* та *j* для проходу масиву.

i=2 – починаємо з другого елементу масиву, *j*=*n*.

Крок 3. Прохід масиву розпочинаємо зліва направо з позиції *i*. Елементи масиву порівнюємо з опорним елементом, пропускаємо елементи менші за опорний і зупиняємося зустрівши перший елемент, який більший за опорний. Це елемент A[*i*].

Крок 4. Розпочинаємо прохід масиву справа наліво з позиції *j*. Елементи масиву порівнюємо з опорним елементом, пропускаємо елементи більші за опорний і зупиняємося зустрівши перший елемент, який менший за опорний.

Це елемент A[*j*].

Крок 5. В залежності від того як розташовані індекси сканування, можливі три випадки:

1. Якщо індекси сканування *i* та *j* не перетнулися, тобто $i < j$, елементи A[*i*] та A[*j*] міняються місцями і продовжується сканування шляхом збільшення *i* та зменшення *j*.

		→ i		j←	
p	Всі елементи ≤ p	≥ p	...	≤ p	Всі елементи ≥ p

2. Якщо при скануванні вийшов перетин індексів, тобто $i > j$, то розбиття виконується міняючи місцями опорний елемент з A[*j*].

		j←		→ i	
p	Всі елементи ≤ p	≤ p	...	≥ p	Всі елементи ≥ p

3. Якщо при скануванні індекси зупинилися на одному елементі, тобто $i = j$, то значення цього елемента повинно бути p. Цей випадок можна об’єднати з випадком 2, міняючи місцями опорний елемент з A[j] при $i \geq j$.

		$\rightarrow$ i = j$\leftarrow$	
p	Всі елементи ≤ p	= p	Всі елементи ≥ p

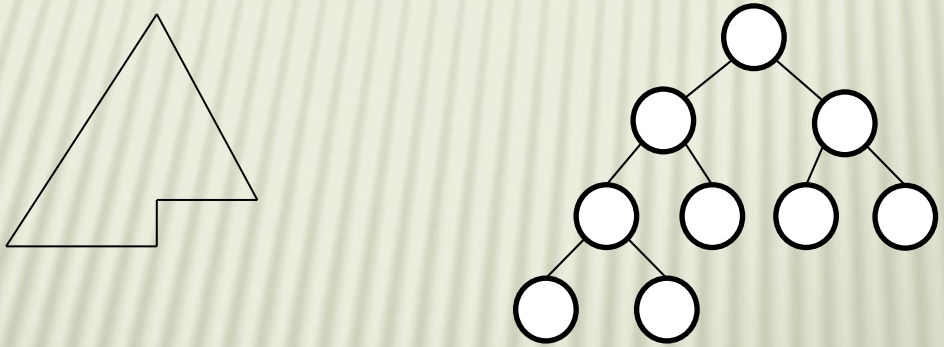
	i								j
50	15	30	60	55	65	40	35	45	70
			i					j	
50	15	30	60	55	65	40	35	45	70
			i					j	
50	15	30	45	55	65	40	35	60	70
				i			j		
50	15	30	45	55	65	40	35	60	70
				i			j		
50	15	30	45	35	65	40	55	60	70
					i	j			
50	15	30	45	35	65	40	55	60	70
					i	j			
50	15	30	45	35	40	65	55	60	70
					j	i			
50	15	30	45	35	40	65	55	60	70
40	15	30	45	35	50	65	55	60	70

	i			j			i		j
40	15	30	45	35		65	55	60	70
			i	j				i = j	
40	15	30	45	35		65	55	60	70
			i	j		60	55	65	70
40	15	30	35	45			i = j		
			j	i		60	55		
40	15	30	35	45		55	60		
35	15	30	40	45		55			70
	i	j							
35	15	30							
		i = j							
35	15	30							
30	15	35							
	i = j								
30	15								
15	30								
15				45					

2.7 Пірамідальне сортування

Піраміда (бінарна куча, сортуюче дерево, binary heap) – це бінарне дерево висотою *k*, в якому:

1) всі вузли мають глибину *k* або *k*-1, тобто дерево збалансоване. При цьому рівень *k*-1 цілком заповнений зліва направо, тобто форма піраміди має приблизно такий вид:



2) Виконується умова: кожний нащадок менше або дорівнює батьківському вузлу.

Для зберігання піраміди помістимо її в масив A[1..N].

Відповідність між геометричною структурою піраміди як дерева і масивом встановлюється за наступною схемою:

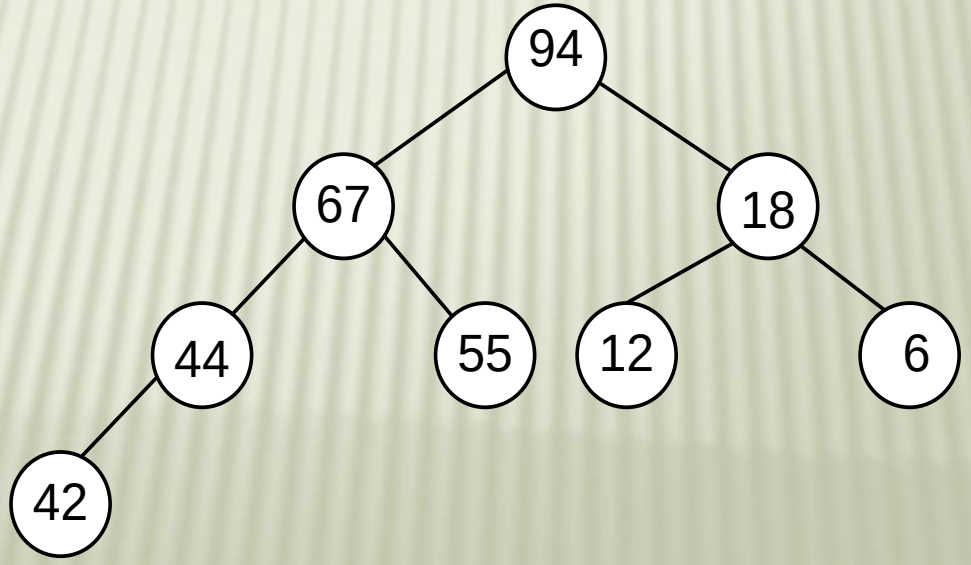
- в *A*[1] зберігається корінь дерева – максимальний елемент;
- Parent(i)=i/2 – індекс батьківського вузла i;
- Left(i)=2i – індекс лівого нащадка;
- Right(i)=2i+1– індекс правого нащадка.

$$A[\text{Parent}(i)] \geq A[i]$$

Переваги такого зберігання:

- ніяких додаткових змінних, треба лише розуміти схему;
- вузли зберігаються вниз, рівень за рівнем;
- вузли одного рівня зберігаються в масиві зліва направо.

Приклад: Задано масив 94, 67, 18, 44, 55, 12, 6, 42. Відтворити у вигляді дерева і довести, що отримане дерево піраміда.



АЛГОРИТМ ПІРАМІДАЛЬНОГО СОРТУВАННЯ:

1 фаза сортування: Побудова піраміди.

Почати побудову піраміди можна з A[k]… A[n], k=[n /2] (кількість елементів масиву).

1)Розглянемо нащадки зліва A[2i] і справа A[2i+1], обираємо найбільший з них:

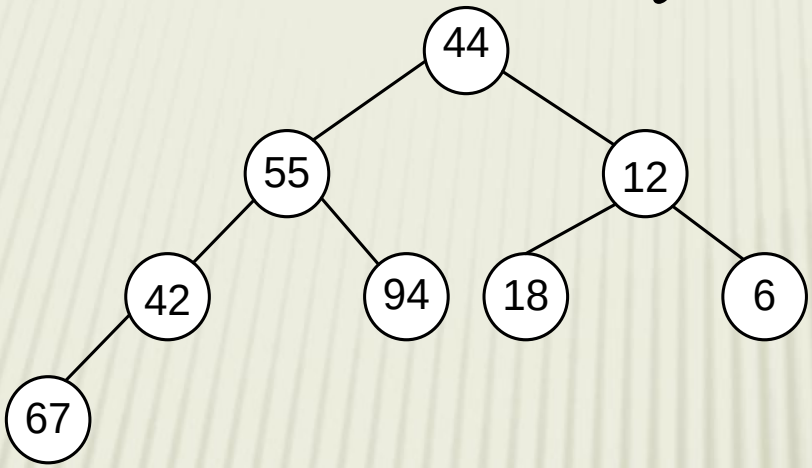
X=max(A[2i], A[2i+1]).

2) Якщо X>A[i] – міняємо X з A[i] місцями. Переходимо до кроку 1, враховуючи нове положення A[i] в масиві.

3) A[1] – max елемент масиву. Виконується «просіювання» елементів від A[2] до A[n] для виконання властивості піраміди.

Приклад: Заданий масив 44, 55, 12, 42, 94, 18, 6, 67 представити у вигляді піраміди.

1. Зобразимо заданий масив у вигляді дерева.

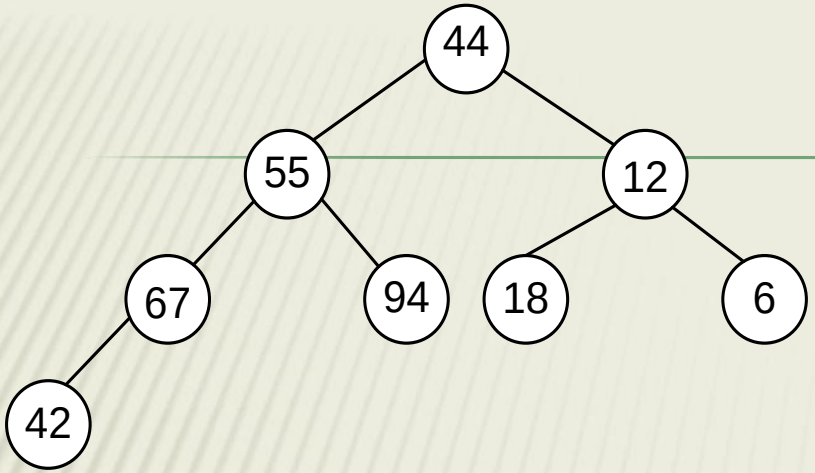


2. Почнемо побудову піраміди з вузла A[n/2].

A[i]=A[4]=42, A[2i]= A[8]=67,

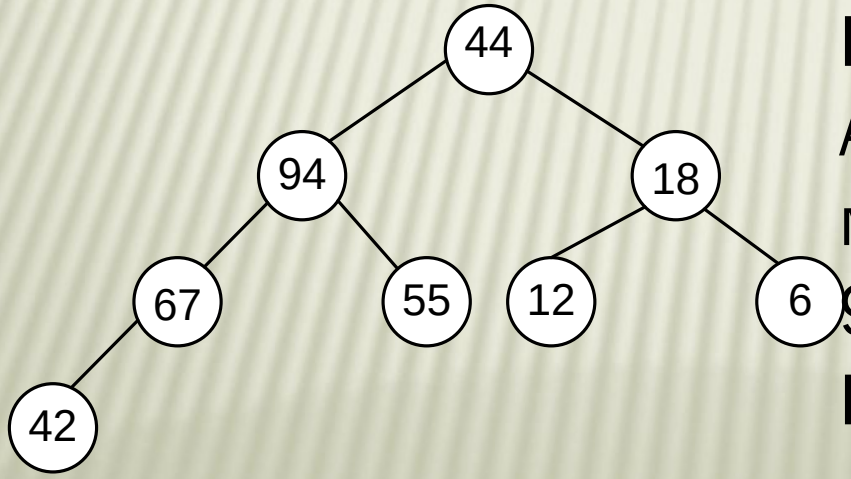
A[2i+1]= A[9] - відсутній.

Порівнюємо A[i] та A[2i], A[4]=42 та A[8]=67, так як A[8] > A[4] міняємо місцями 42 і 67.

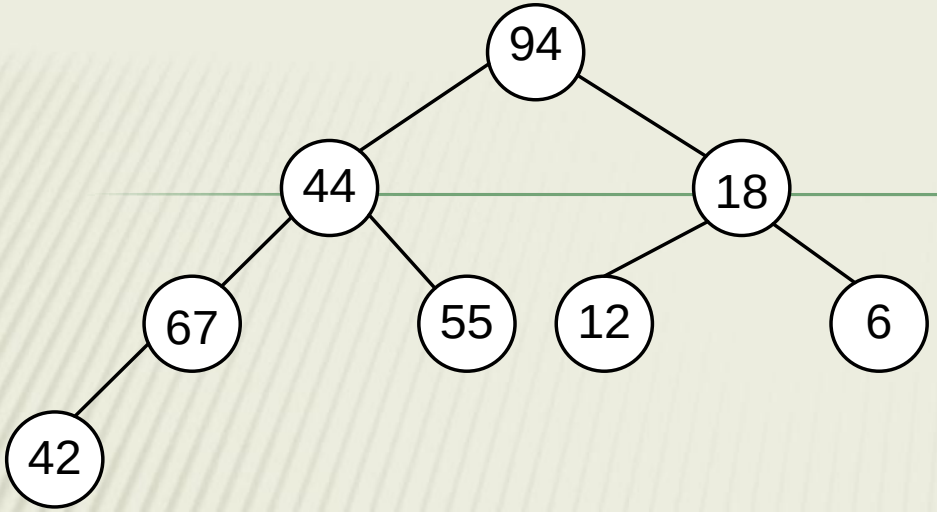


Порівнюємо нащадки вузла A[2]:
max(A[4], A[5])= max (67,94)= 94, max(94,55)=94.
Поміняли місцями 94 і 55.

Порівнюємо нащадки вузла A[3]:
max(A[6], A[7])= max (18,6)= 18, max(18,12)=18.
Поміняли місцями 18 і 12.

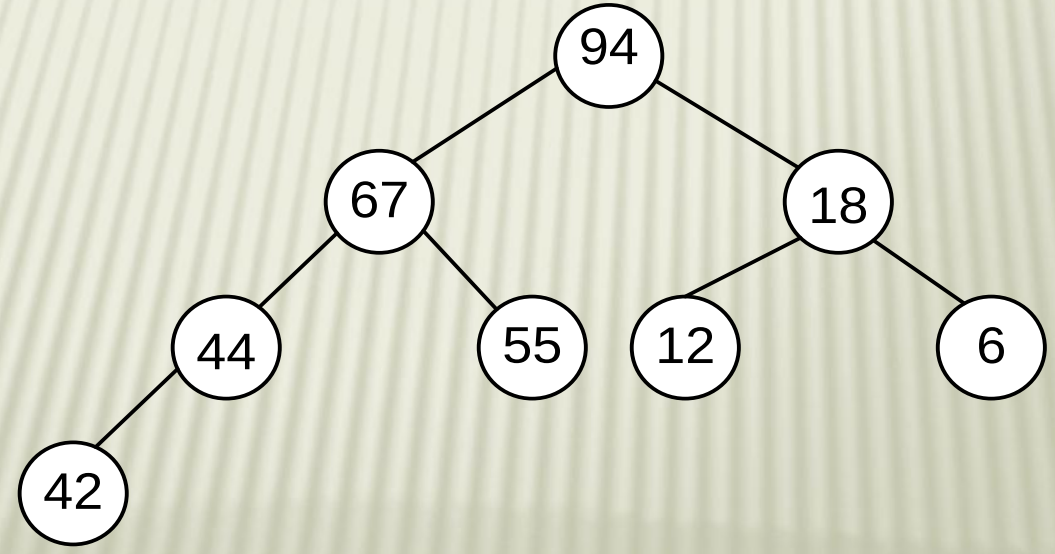


Порівнюємо нащадки вузла A[1]:
max(A[2], A[3])= max (94,18)= 94, max(94,44)=94.
Поміняли місцями 94 і 44.



Просіювання елементів зверху вниз.

Отримаємо піраміду:

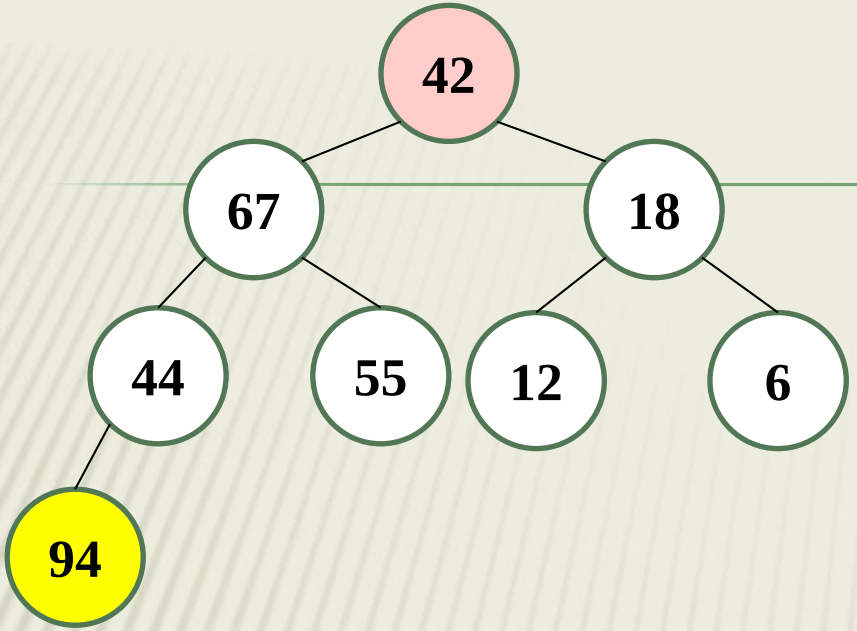


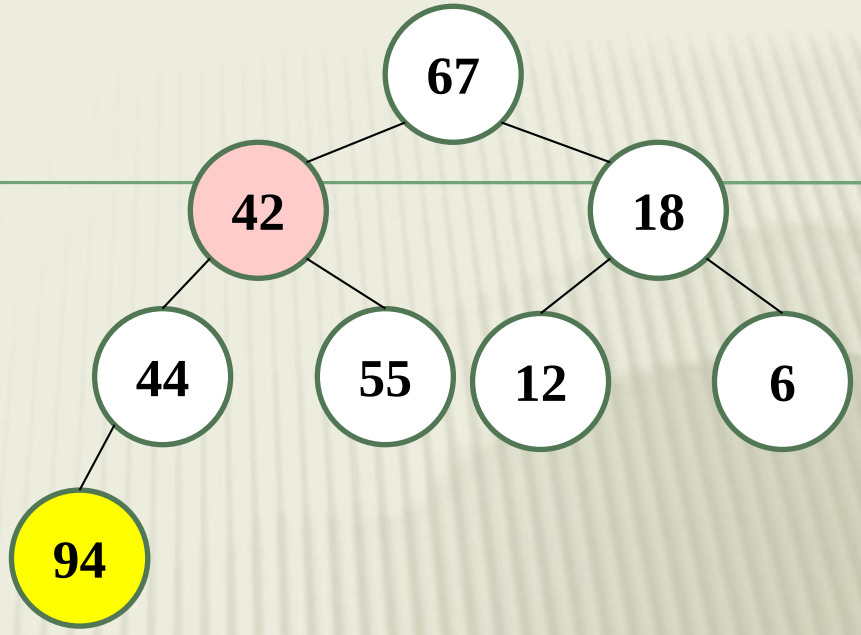
Фаза 2: Сортування.

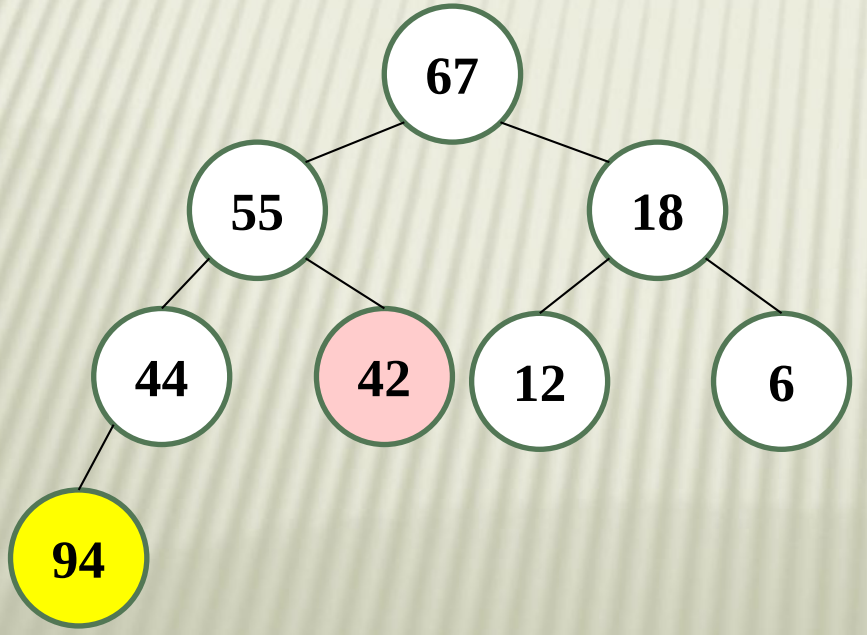
В корні піраміди після її побудови завжди знаходиться максимальний елемент.

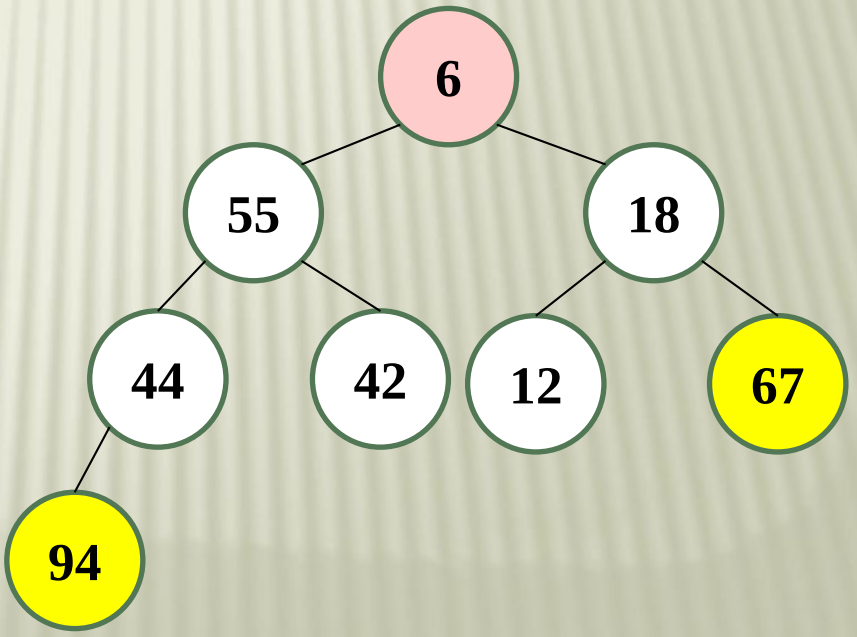
1) Беремо елемент піраміди A[1] і міняємо місцями з останнім A[*n*]. Далі розглядаємо масив A[1]...A[*k*], k=n-1..2. Для перетворення його в піраміду достатньо «просіяти» лише новий перший елемент.

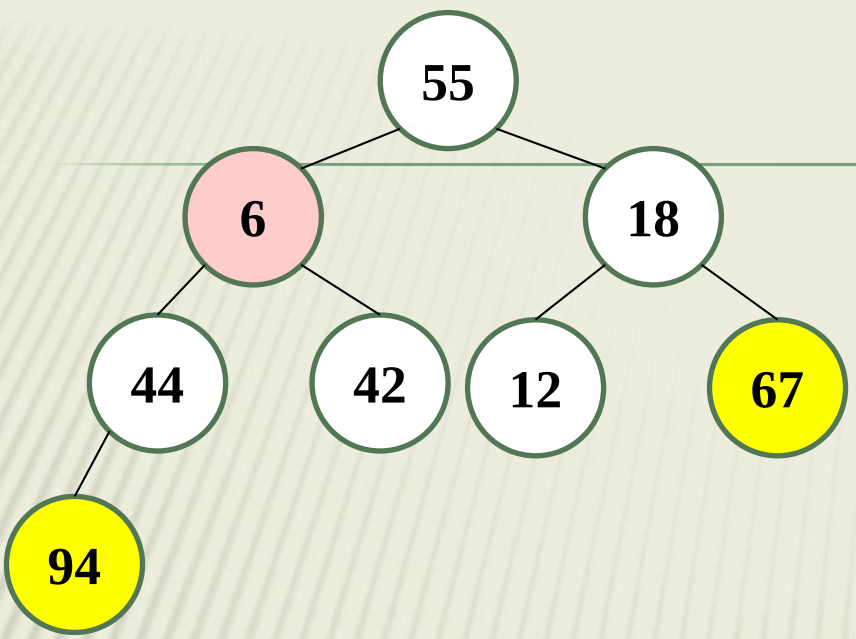
2) Повторюємо крок 1, поки частина масиву, що обробляється не зменшиться до одного елементу.

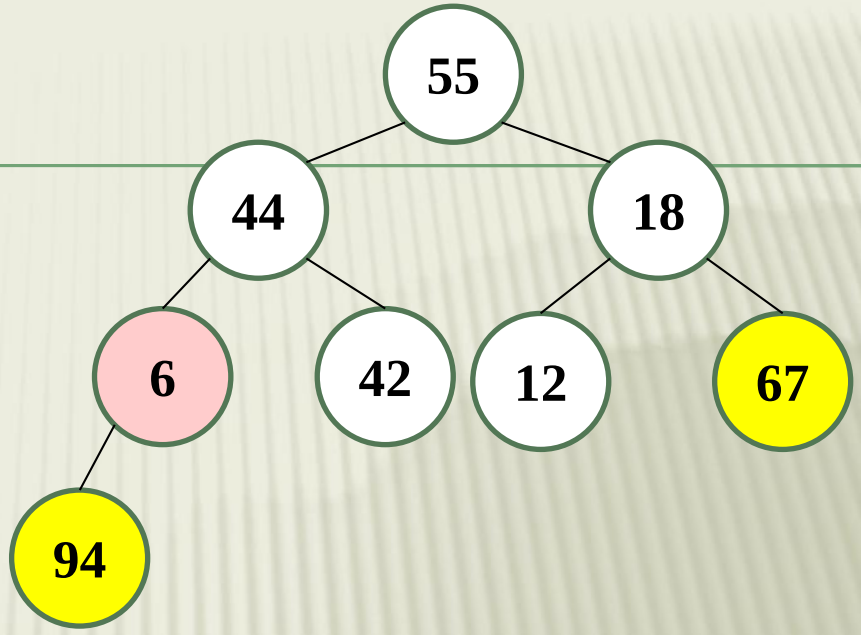

42
67
18
44
55
12
6
94

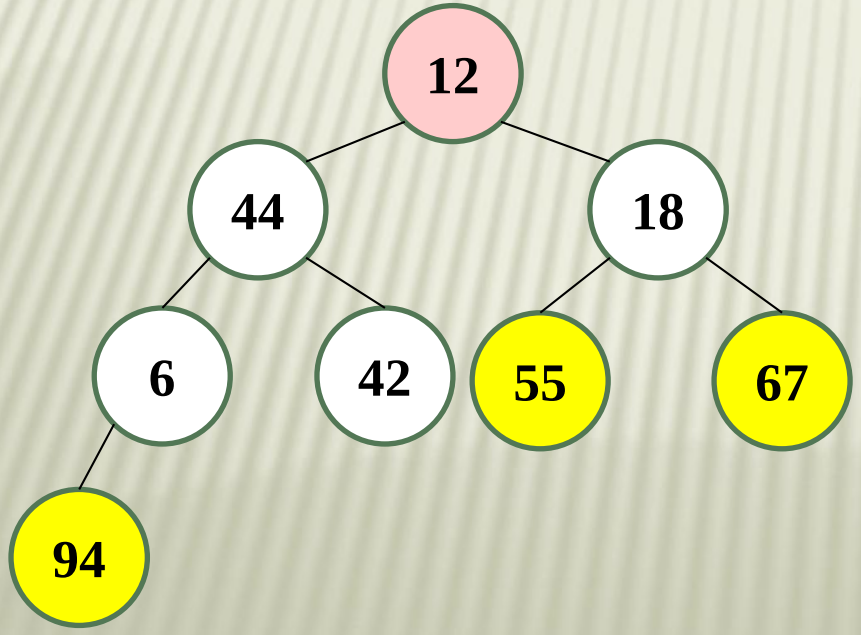

67
42
18
44
55
12
6
94

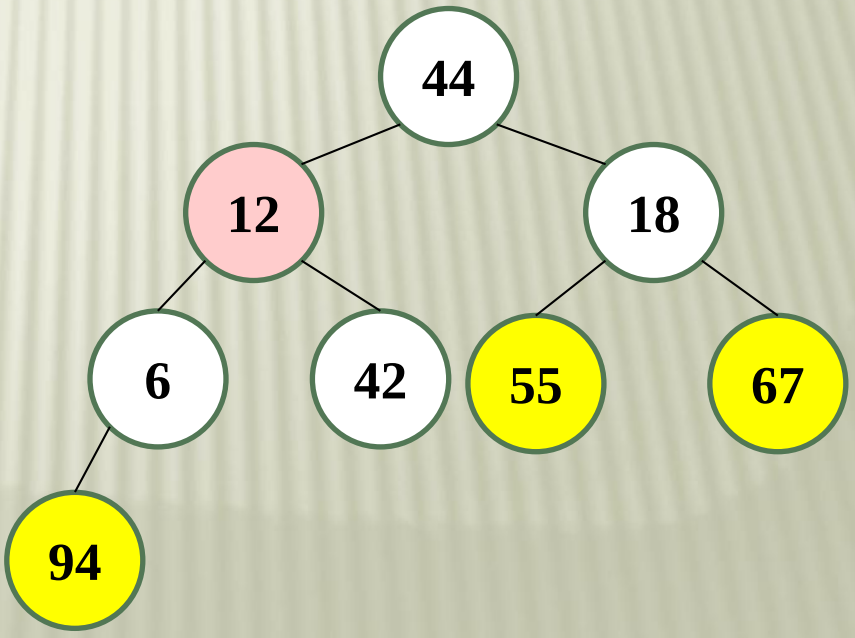

67
55
18
44
42
12
6
94

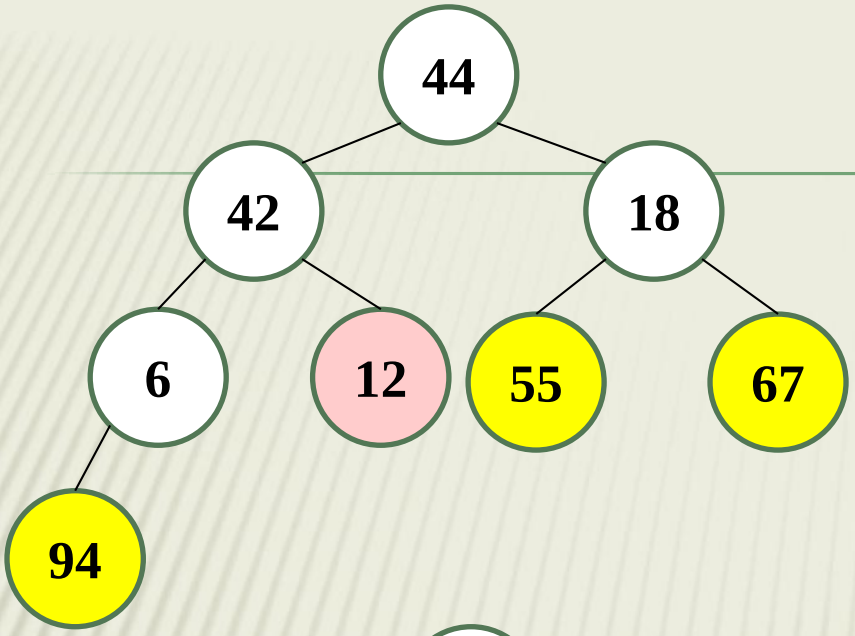

6
55
18
44
42
12
67
94

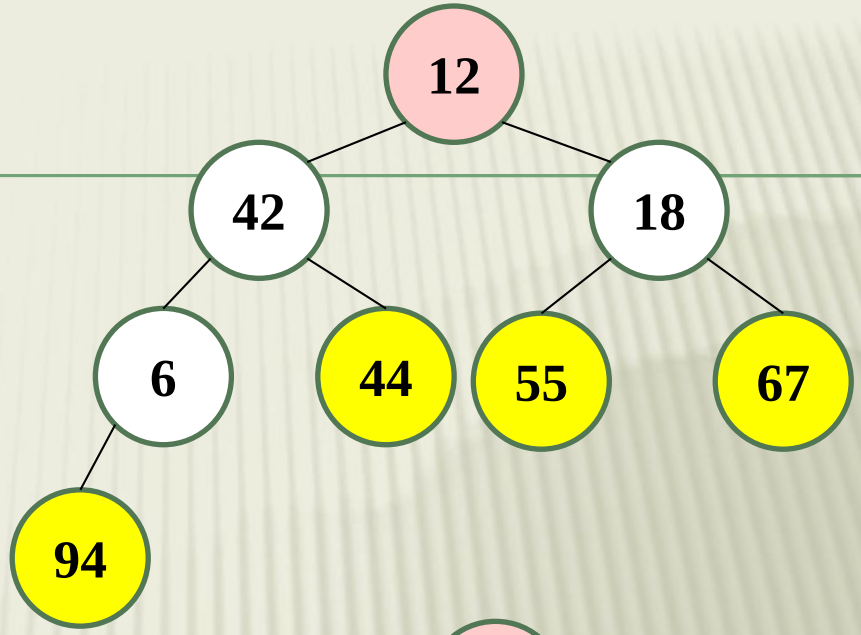

55
6
18
44
42
12
67
94

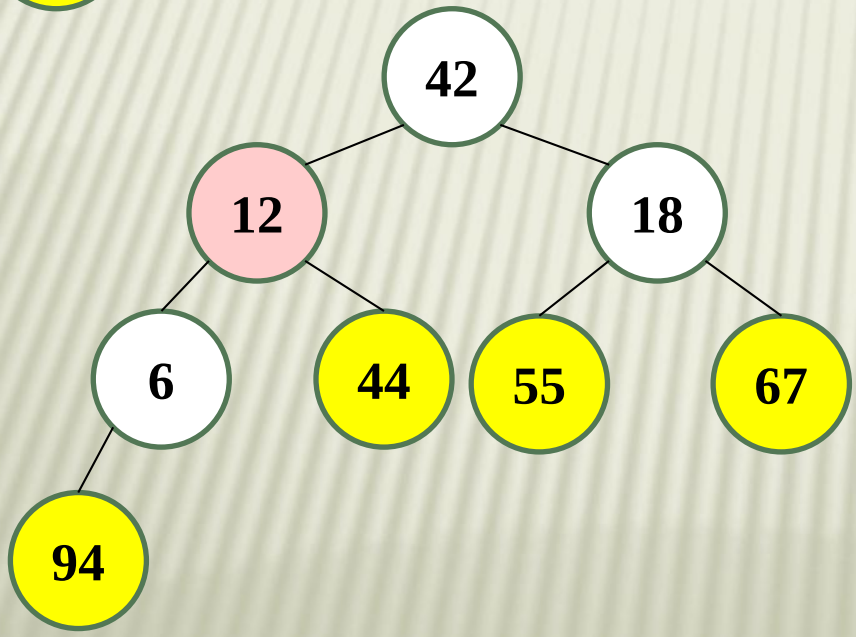

55
44
18
6
42
12
67
94

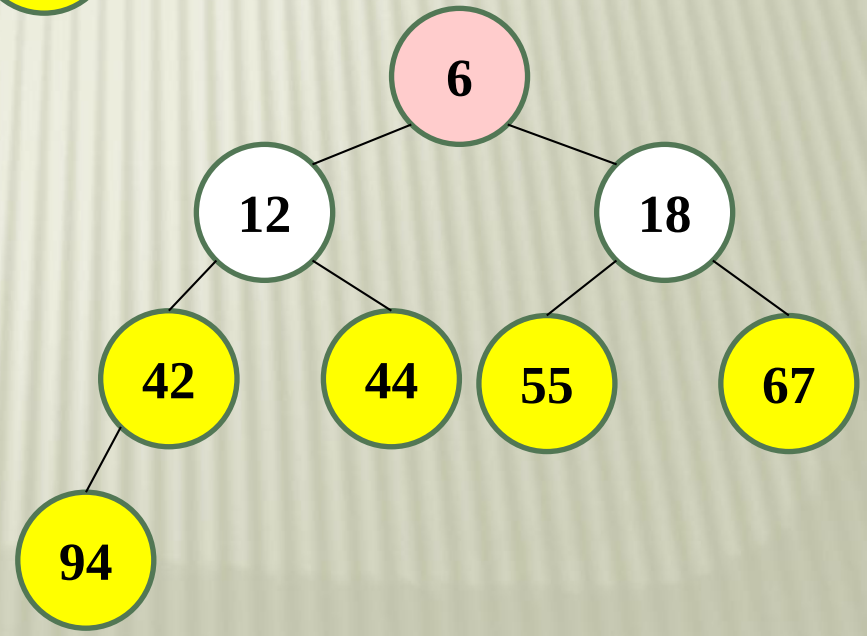

12
44
18
6
42
55
67
94

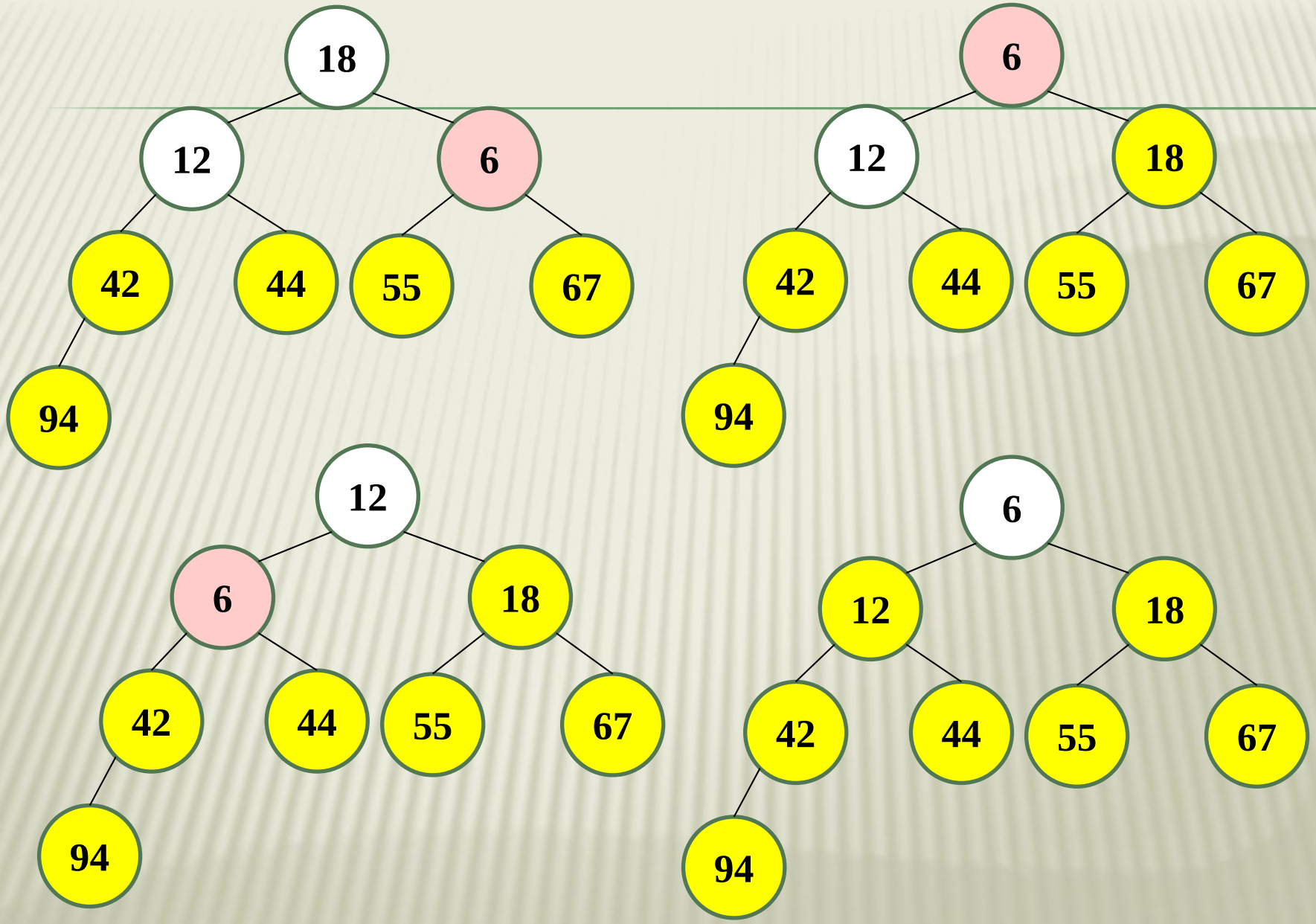

44
12
18
6
42
55
67
94


44
42
18
6
12
55
67
94


12
42
18
6
44
55
67
94


42
12
18
6
44
55
67
94


6
12
18
42
44
55
67
94


18
12
6
42
44
55
67
94
6
12
18
42
44
55
67
94
12
6
18
42
44
55
67
94
6
12
18
42
44
55
67
94

Відсортований масив:

6, 12, 18, 42, 44, 55, 67, 94

Пірамідальне сортування не використовує додаткову пам’ять.

Побудова піраміди займає O (n logn) операцій. Друга фаза займає O (*n* log *n*) часу.

Метод не є стійким: по ходу роботи початковий порядок елементів масиву повністю змінюється.

Поведінка неприродна: часткова впорядкованість масиву ніяк не враховується.