ЛЕКЦІЯ 6. МЕТОД ДЕКОМПОЗИЦІЇ

6.1. Метод декомпозиції

В методі сортування включенням використовується так званий *інкрементний* підхід: маючи відсортований підмасив $A[1\ldots j-1]$, ми розміщуємо черговий елемент $A[j]$ туди, де він повинен заходитись, в результаті чого отримуємо відсортований підмасив $A[1\ldots j]$. В цій темі буде розглянутий рекурсивний підхід, коли початкова задача розбивається на підзадачі, розв'язки яких після цього певним чином опрацьовуються для отримання загального рішення. Зрозуміло, що підзадачі початкової задачі потім так само розбиваються на власні підзадачі, і так триває до досягнення певного мінімального розміру задачі, яка сама розв'язується вже тривіальним чином.

Ця рекурсивна парадигма отримала назву „*розділяй та володарюй*" (англ. *divide and conquer*). Одним з перших алгоритмів, який працював в рамках цієї парадигми, був метод Карацуби множення двох цілих чисел, який розглядався в першій темі. Сама парадигма „розділяй та володарюй" складається з трьох етапів:

- *Розділення* задачі на декілька підзадач.
- *Рекурсивний розв'язок* цих підзадач. Коли об'єм підзадач достатньо малий, вони розв'язуються безпосередньо.
- *Комбінування* розв'язку початкової задачі з розв'язків допоміжних підзадач.

Розгляд цієї парадигми ми почнемо з алгоритму сортування *методом злиття* (англ. *merge sort*). В рамках підходу „розділяй та володарюй" цей метод можна описати так:

- *Розділення*: послідовність для сортування, яка складається з n елементів, розбивається на дві менші послідовності, кожна з яких містить $n/2$ елементів.
- *Рекурсивний розв'язок*: сортування обох створених послідовностей методом злиття у випадку, якщо їх розмір перевищує один.
- *Комбінування*: злиття двох відсортованих послідовностей для отримання кінцевого результату.

Рекурсія досягає своєї нижньої межі, коли довжина послідовності для

сортування дорівнює 1. В цьому випадку вся робота вже зроблена, оскільки будь-яку таку послідовність можна вважати впорядкованою.

Основна операція, яка виконується в процесі сортування за методом злиття, – це об'єднання двох відсортованих послідовностей під час комбінування (останній етап). Це робиться за допомогою додаткової процедури `Merge(A, p, q, r)`, де A – це масив, а p, q та r – індекси, які нумерують елементи масиву, такі що $p \le q < r$. В цій процедурі припускається, що елементи підмасивів $A[p \ldots q]$ та $A[q+1 \ldots r]$ впорядковані. Вона зливає ці два підмасиви в один відсортований, елементи якого замінюють поточні елементи підмасиву $A[p \ldots r]$.

Для виконання процедури `Merge` потрібний час $\Theta(n)$, де $n = r - p + 1$ – кількість елементів, які потрібно об'єднати. Для демонстрації роботи процедури можна знову повернутись до прикладу гральних карт. Нехай на столі лежать дві стопки карт, кожна з яких є вже відсортованою. Для того, щоб об'єднати їх в одну відсортовану стопку, можна діяти наступним чином: на кожному кроці ми порівнюємо по одній карті зі стопок, які лежать згори, та обираємо в нову стопку ту з них, яка є найменшою. Цей крок повторюється до тих пір, допоки одна зі стопок не спорожніє. Тоді залишиться лише додати карти зі стопки, що лишилась, у нову стопку не порушуючи порядок карт. З обчислювальної точки зору виконання кожного основного кроку займає однакові проміжки часу, тоді як все зводиться до порівняння двох верхніх карт. Оскільки необхідно виконати принаймні n основних кроків, час роботи процедури `Merge` дорівнює $\Theta(n)$.

У наведеному нижче псевдокоді представлена процедура `Merge`. Проте для її спрощення використовуються додаткові міркування. Щоб на кожному кроці не перевіряти, чи не спорожнів якийсь з двох підмасивів, до кожного з них додається так званий сигнальний елемент, який має значення нескінченності ∞. Таким чином, не існує елементів масивів, які були б більшими за ці сигнальні елементи. Робота процедури `Merge` продовжується до тих пір, поки поточні елементи в обох підмасивах не виявляться сигнальними. Як тільки це станеться, це буде означати, що всі несигнальні елементи розміщені у вихідний масив. Оскільки завчасно відомо, що у вихідному масиві повинен бути присутній $r - p + 1$ елемент, то виконавши таку кількість кроків можна зупинитись.

```
Merge(A, p, q, r)
1     n1 = q − p + 1
2     n2 = r − q
3     Створити масиви L[1..n1+1] та R[1..n2+1]
4     for i = 1 to n1
5          do L[i] = A[p+i-1]
6     for j =1 to n2
7          do R[j] = A[q+j]
8     L[n1+1] = ∞
9     R[n2+1] = ∞
10    i = 1
11    j = 1
12    for k = p to r
13         do if L[i] ≤ R[j]
14              then A[k] = L[i]
15                   i = i + 1
16              else A[k] = R[j]
17                   j = j + 1
```

Лістинг 6.1 Процедура злиття `Merge`

Детально опишемо роботу процедури `Merge`. В рядку 1 обраховується довжина n_1 підмасиву $A[p…q]$, а у рядку 2 - довжина n_2 підмасиву $A[q+1…r]$. Далі в рядку 3 створюються масиви L („лівий”) та R („правий”), довжини яких відповідно $n_1 + 1$ та $n_2 + 1$. В двох циклах **for** в рядках 4-5 та 6-7 елементи масиву $A[p…q]$ та $A[q+1…r]$ копіюються у масиви L та R відповідно. В рядках 8 та 9 останнім елементам масивів L та R приписуються сигнальні значення.

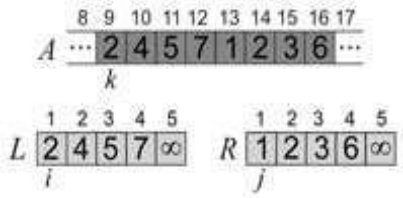
Як показано на рис. 6.1, в результаті копіювання та додавання сигнальних елементів отримуємо масив L з послідовністю чисел $[2, 4, 5, 7, \infty]$та масив R з послідовністю $[1, 2, 3, 6, \infty]$. Світло-сірі комірки масиву A містять кінцеві елементи, а світло-сірі комірки масивів L та R – значення, які ще тільки необхідно скопіювати в масив A. У темно-сірих комірках A містяться значення, які будуть замінені іншими, а в темно-сірих масивів L та R – значення, які вже скопійовані назад в A.

В рядках 10-17 лістингу 6.1 виконуються $r – p + 1$ основних кроків, в ході

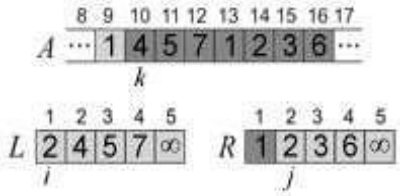
кожного з яких відбуваються маніпуляції з інваріантом циклу:

Перед кожною ітерацією циклу **for** в рядках 12-17, підмасив $A[p \ldots k-1]$ містить $k-p$ найменших елементів масивів L та R у відсортованому порядку. Окрім того, елементи $L[i]$ та $R[j]$ є найменшими елементами L та R, які ще не були скопійовані у A.

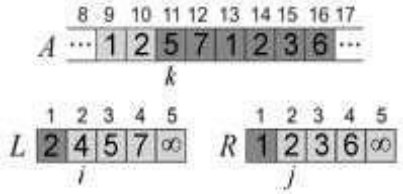
Необхідно показати, що цей інваріант циклу зберігається перед першою ітерацією даного циклу **for**, що кожна ітерація циклу не порушує його, і що з його допомогою можна продемонструвати коректність алгоритму, коли цикл закінчує свою роботу.



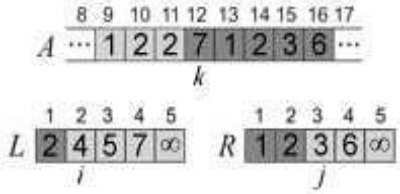
а)



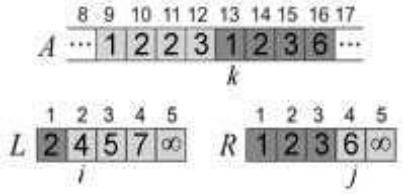
б)



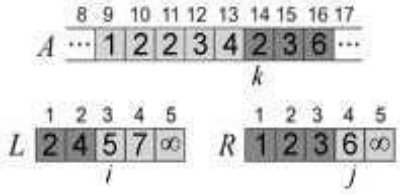
в)



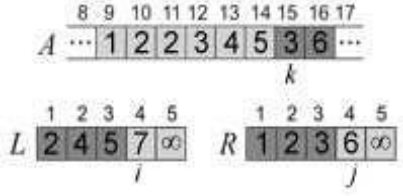
г)



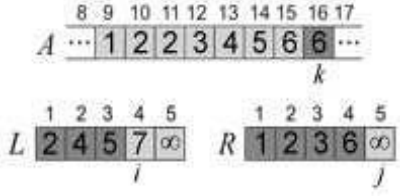
д)



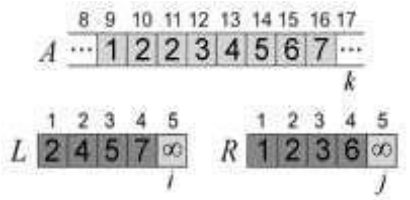
е)



є)



ж)



з)

Рис. 6.1. Приклад виконання процедури `Merge`

1. *Ініціалізація*. Перед першою ітерацією циклу $k = p$, тому підмасив $A[p\ldots k–1]$ порожній. Він містить $k – p = 0$ найменших елементів масивів L та R. Оскільки $i = j = 1$, елементи $L[i]$ та $R[j]$ – найменші елементи масивів L та R, які ще не скопійовані у A.
2. *Збереження*. Щоб пересвідчитись, що інваріант циклу зберігається після кожної ітерації, спочатку припустимо, що $L[i] \le R[j]$. Тоді $L[i]$ – найменший елемент, який ще не скопійовано в A. Оскільки в підмасиві $A[p\ldots k–1]$ міститься $k – p$ найменших елементів, після виконання рядку 14, в якому значення елементу $L[i]$ приписується елементу $A[k]$, в підмасиві $A[p\ldots k]$ буде міститись $k – p + 1$ найменший елемент. В результаті збільшення параметру k циклу **for** та значення змінної i (рядок 15), інваріант циклу відновлюється перед наступною ітерацією. Якщо $L[i] > R[j]$, то в рядках 16 та 17 виконуються відповідні дії, які також зберігають інваріант циклу.
3. *Завершення*. Алгоритм завершується, коли $k = r + 1$. У відповідності з інваріантом циклу, підмасив $A[p\ldots k–1]$ (тобто підмасив $A[p\ldots r]$) містить $k – p = r – p + 1$ найменших елементів масивів L та R у відсортованому порядку.

   Сумарна кількість елементів в масивах L та R дорівнює $n_1 + n_2 + 2 = r – p + 3$. Всі вони, окрім двох найбільших, скопійовані назад в масив A, а два елементи що залишились є сигнальними.

Щоб показати, що час роботи процедури `Merge` дорівнює $\Theta(n)$, де $n = r – p + 1$, зазначимо, що кожний рядок 1-3 та 8-11 виконується протягом фіксованого часу; довжина циклів **for** у рядках 4-7 дорівнює $\Theta(n_1 + n_2) = \Theta(n)$, а в циклі **for** у рядках 12-17 виконується n ітерацій, на кожну з яких витрачається

фіксований час.

Тепер процедуру `Merge` можна використовувати в якості підпрограми в алгоритмі сортування злиттям. Процедура `MergeSort(A, p, r)` виконує сортування елементів в підмасиві $A[p \ldots r]$. Якщо виконується нерівність $p \geq r$, то в цьому масиві елементів міститься не більше одного, тому він є відсортованим. У протилежному випадку відбувається розбиття, під час якого обраховується індекс q, який розбиває масив $A[p \ldots r]$ на два підмасиви: $A[p \ldots q]$ з $\lceil n/2 \rceil$ елементами та $A[q+1 \ldots r]$ з $\lfloor n/2 \rfloor$ елементами.

```
MergeSort(A, p, r)
1    if p < r
2         then q = ⌊(p+r)/2⌋
3              MergeSort(A, p, q)
4              MergeSort(A, q + 1, r)
5              Merge(A, p, q, r)
```

Лістинг 6.2. Процедура сортування `MergeSort`

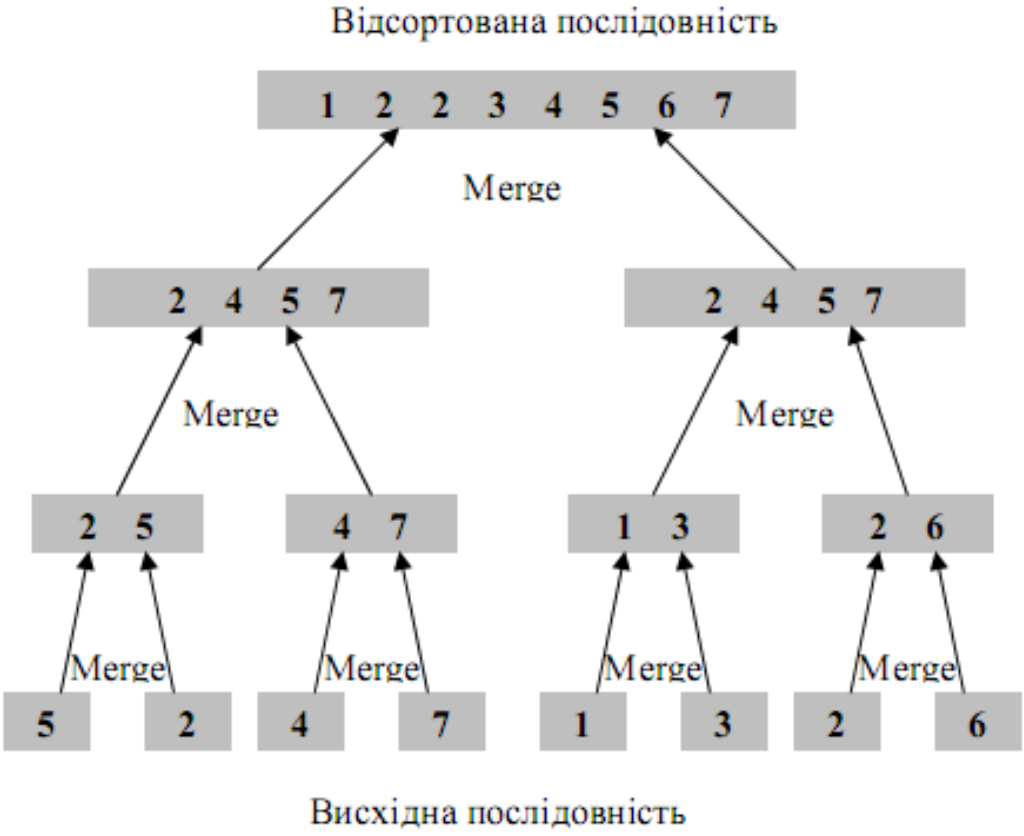


Рис 6.2. Процес сортування масиву A = [5, 2, 4, 7, 1, 3, 2, 6]

Щоб відсортувати послідовність $A = [A[1], A[2],\ldots,A[n]]$, викликається процедура `MergeSort`(A, 1, *length*[A]), де *length*[A] = n. На рис. 6.2 наводиться приклад роботи цієї процедури, якщо n – ступінь двійки. В ході роботи відбувається попарне об'єднання одноелементних послідовностей у відсортовані послідовності довжини 2, потім – попарне об'єднання двоелементних послідовностей у відсортовані послідовності довжини 4 і т.д., допоки не будуть отримані дві послідовності довжиною $n/2$, які об'єднаються у кінцеву відсортовану послідовність довжиною n.

6.2. Аналіз алгоритму сортування злиттям

Якщо алгоритм рекурсивно звертається сам до себе, час його роботи часто описується за допомогою *рекурентного рівняння*, в якому повний час, потрібний для розв'язку всієї задачі з об'ємом входу n, виражається через час розв'язку допоміжних підзадач. Потім дане рекурентне рівняння розв'язується за допомогою певних математичних методів (див. тему 4) і встановлюються межі ефективності алгоритму.

Отримання рекурентного співвідношення для часу роботи алгоритму, який заснований на принципі „розділяй та володарюй", базується на трьох етапах, які відповідають цій парадигмі. Позначимо через $T(n)$ час розв'язку задачі, розмір якої дорівнює n. Якщо розмір задачі достатньо малий, скажімо, $n \le c$, де c – деяка заздалегідь відома стала, то задача розв'язується безпосередньо за фіксований час, який позначається через $\Theta(1)$. Припустимо, що наша задача ділиться на a підзадач, об'єм кожної з яких дорівнює $1/b$ від об'єму висхідної задачі. Для алгоритму сортування методом злиття a та b дорівнюють 2, хоча в загальному випадку зовсім не обов'язково, щоб $a = b$. Якщо розбиття задачі на допоміжні підзадачі відбувається протягом часу $D(n)$, а об'єднання розв'язків підзадач до розв'язку початкової задачі – протягом часу $C(n)$, то ми отримуємо наступне рекурентне рівняння:

$$T(n) = \begin{cases} \Theta(1), & \text{якщо } n \le c; \\ aT(n/b) + D(n) + C(n), & \text{в протилежному випадку.} \end{cases}$$

Для спрощення аналізу швидкодії алгоритму `MergeSort` припустимо, що n дорівнює степеню двійки. Проте даний алгоритм працює однаково ефективно для загального випадку.

Сортування одного елементу методом злиття триває фіксований проміжок часу. Якщо $n > 1$, час роботи алгоритму розподіляється наступним чином:

- *Розділення.* Під час розділення визначається, де знаходиться середина підмасиву. Ця операція триває фіксований час, тому $D(n) = \Theta(1)$.
- *Рекурсивний розв'язок*. Рекурсивно розв'язуються дві підзадачі, об'єм кожної з яких складає $n/2$. Час розв'язку цих підзадач становить $2T(n/2)$.
- *Комбінування*. Процедура `Merge` для n-елементного масиву виконується протягом часу $\Theta(n)$, тому $C(n) = \Theta(n)$.

Виходячи з того, що $D(n) + C(n) = \Theta(1) + \Theta(n) = \Theta(n)$, отримуємо вираз рекурентного рівняння для методу злиття в найгіршому випадку:

$$T(n) = \begin{cases} \Theta(1), & \text{якщо } n = 1; \\ 2T(n/2) + \Theta(n), & \text{якщо } n > 1. \end{cases} \qquad (6.1)$$

Перепишемо рівняння (6.1) у вигляді:

$$T(n) = \begin{cases} c = const, & \text{якщо } n = 1; \\ 2T(n/2) + cn, & \text{якщо } n > 1. \end{cases}, \qquad (6.2)$$

де константа c позначає час, який необхідний для розв'язку задачі, розмір якої дорівнює 1, а також питомий час, який потрібний для розділення та комбінування (у випадку, якщо розміри цих проміжків часу різні, то константою c можна позначити найбільший з них).

Процес розв'язку рекурентного співвідношення (6.2) показаний на рис. 6.3. В частині *а* показаний час $T(n)$, який представлений у частині *б* у вигляді еквівалентного дерева, що отримується з рекурентного рівняння (6.2). Коренем цього дерева є доданок cn, а два піддерева представляють дві менші рекурентні послідовності $T(n/2)$. В частині *в* показаний черговий крок рекурсії. Далі продовжується розкладення кожного вузла, який входить у дерево шляхом розбиття на складові частини, виходячи з рекурентного співвідношення. Так

відбувається до тих пір, допоки розмір послідовності не буде дорівнювати 1, а час її виконання – константі *c*. Отримане дерево наводиться в частин *г*. Дерево складається з $\lg n+1$ рівнів (адже його висота дорівнює lg*n*), а кожний рівень дає сумарний вклад в повний час роботи алгоритму, який дорівнює *cn*. В цьому легко переконатись: час роботи в корені дорівнює *cn*, час роботи на другому рівні – $c(n/2)+c(n/2)=cn$. В загальному випадку рівень *i* має 2^i вузлів, кожний з яких виконується протягом часу $c\left(n/2^i\right)$, тому повний час для всіх вузлів рівня *i* становить $2^i c\left(n/2^i\right)=cn$. На нижньому рівні є *n* вузлів, кожний з яких дає вклад *c*, що в сумі становить знову *cn*.

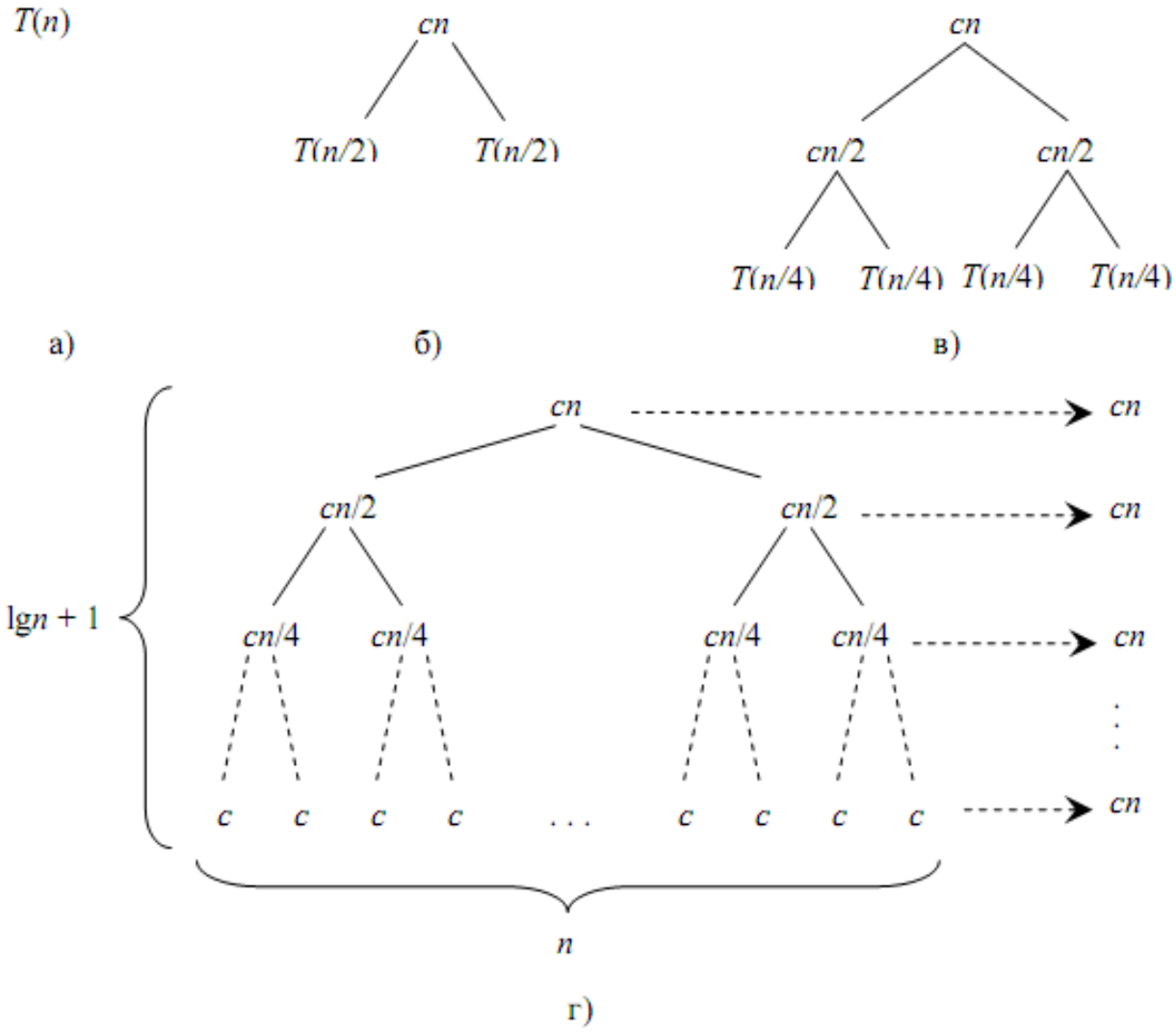


Рис. 6.3. Побудова дерева рекурсіє для рівняння $T(n)=2T(n/2)+cn$

Щоб знайти повний час роботи алгоритму, потрібно скласти час кожного рівня. Маючи $\lg n+1$ рівнів, кожний з яких виконується протягом часу *cn*,

отримуємо загальний час $cn(\lg n + 1) = cn\lg n + cn$. Нехтуючи членами менших порядків та константою c, в результаті отримуємо $\Theta(n\lg n)$ – час роботи процедури `MergeSort`.

Література

1. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К. Алгоритмы: построение и анализ, 2-е издание. : Пер. с англ. – М. : Изд. дом "Вильямс", 2011. – 1296 с. (Глава 2, розділ 2.3.)
2. Dasgupta, Sanjoy, Papadimitriou, Christos, Vazirani, Umesh (2006) Algorithms. McGraw-Hill Science/Engineering/Math. ISBN-13 978-0073523408. (Глава 2, розділ 2.5).