Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Санкт-Петербургский государственный университет»
Факультет прикладной математики – процессов управления

РЕФЕРАТ

На тему

«Эмпирический анализ алгоритма пирамидальной сортировки»

Выполнил:Студент 3 курса,
группы 21.Б12-пу
Давлятшин Г.Р.

Санкт-Петербург, 2023

Содержание

Оглавление

Краткое описание

Пирамидальная сортировка — это алгоритм сортировки, основанный на структуре данных "куча" (heap). В процессе сортировки элементы массива последовательно добавляются в кучу, а затем извлекаются с учетом их приоритета, формируя отсортированный результат[1].

Идея алгоритма

Идея пирамидальной сортировки заключается в использовании структуры данных, известной как "куча" или "пирамида" (heap), для пошагового преобразования входного массива в бинарное дерево с определенными свойствами. В данном случае это "куча" — это такое бинарное дерево, где значение каждого узла больше или равно значения его дочерних узлов (для макс-кучи) или меньше или равно (для мин-кучи).

Алгоритм пирамидальной сортировки выполняется в два этапа. На первом этапе строится куча из входного массива. Затем, чтобы получить отсортированный массив, извлекается корень кучи (элемент с максимальным или минимальным значением, в зависимости от типа кучи) и помещается в конец массива. Процесс повторяется, уменьшая размер кучи на 1 и восстанавливая свойства кучи. Это повторяется до тех пор, пока весь массив не будет отсортирован.[2]

Основное преимущество пирамидальной сортировки заключается в её небольшом использовании дополнительной памяти (in-place sorting), что делает её эффективным вариантом для сортировки больших объемов данных.

Алгоритм

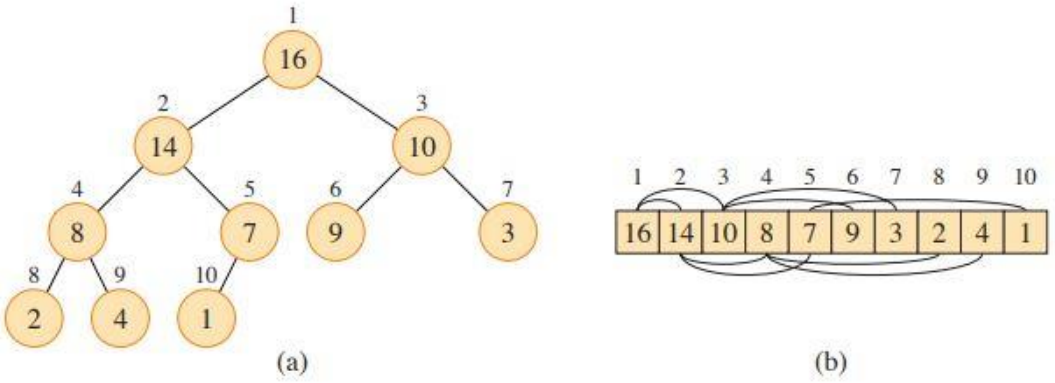
Перед описанием алгоритма стоит отметить, что кучу, в которой все элементы смещены влево можно представить в виде массива, в котором по номеру можно найти родителя и потомков каждого узла.[2]

Пусть i – номер элемента, тогда

Left child = i*2 + 1

Right child = Left child + 1 = i*2 + 2

Parent = $\left\lfloor \frac{i-1}{2} \right\rfloor$



Шаг 1: Построение кучи (Heapify) – O(n)

1. **Начальное состояние:**
   - Начнем с неотсортированного массива.
2. **Построение кучи:**
   - Поочерёдно добавляем элементы массива в массив, представляющий

нашу “кучу”. Затем сравниваем с родителем, если есть необходимость меняем текущий элемент и его родителя местами и повторяем процедуру текущего элемента уже на позиции его предыдущего родителя, если необходимости нет – останавливаемся. Отметим, что при таком способе построения “кучи” стоимость алгоритма будет составлять O(nlogn).

- Существует и более выгодный по времени алгоритм построения кучи за O(n). Преобразуем массив в кучу, начиная с последнего уровня дерева и применяя процедуру "просеивания вниз" (heapify) для каждого элемента. Это обеспечивает, что каждый узел больше или равен своим дочерним узлам (макс-куча) или меньше или равен (мин-куча).

Шаг 2: Извлечение элементов из кучи и формирование отсортированного массива – O(nlogn)

3. **Изменение максимального минимального элемента:**
   - Извлекаем максимальный элемент (корень макс-кучи) – меняем его с последним элементом массива. Затем восстанавливаем свойство кучи: последовательно проверяем элемент, ставший корнем дерева, на выполнение свойства кучи и при необходимости “просеиваем его вниз”
4. **Повторение:**
   - Повторяем процесс извлечения для оставшихся элементов кучи, уменьшая размер кучи на 1 на каждом шаге.
5. **Получение отсортированного массива:**
   - После извлечения всех элементов, у нас получается отсортированный массив.

Области применения

Пирамидальная сортировка имеет несколько областей применения в силу своих характеристик. Вот некоторые из них:

- **Сортировка больших объёмов данных**: Пирамидальная сортировка работает в пределах O(n log n) в худшем случае и выполняется на месте (in-place), что делает ее эффективным методом для сортировки больших объемов данных без необходимости использования дополнительной памяти.
- **Приложения в базах данных**: Эффективность и отсутствие необходимости в дополнительной памяти делают пирамидальную сортировку привлекательным выбором для сортировки результатов запросов в базах данных.
- **Приоритетные очереди**: Пирамидальные структуры данных также используются для реализации приоритетных очередей, где наивысший (или наименьший, в зависимости от типа кучи) приоритет выталкивается и обрабатывается первым.
- **Алгоритмы выборки**: Извлечение максимального (или минимального) элемента из кучи — основная операция в пирамидальной сортировке. Это свойство делает алгоритм полезным в алгоритмах выборки, где необходимо быстро получить элемент с максимальным (минимальным) значением.

Хотя пирамидальная сортировка может не всегда быть самым быстрым алгоритмом сортировки на практике, она остается полезным инструментом в контексте конкретных требований задачи и особенностей данных.

История разработки

Пирамидальная сортировка была введена в 1964 году Робертом Флойдом (Robert W. Floyd) и Дж. Робертом Бойром (J. Robert Boyer). Алгоритм также был представлен в работе Тони Хоара (Tony Hoare) в 1961 году, но из-за ограничений доступа к работе Хоара на тот момент описание алгоритма Флойда и Бойра получило более широкое распространение.[2]

Априорный анализ

1. **Временная сложность пирамидальной сортировки:**
   Построение кучи из исходного массива имеет временную сложность O(n), в то время как сама сортировка (разбор кучи) – O(nlogn), где n – количество элементов в массиве, откуда имеем logn уровней в куче. Таким образом пирамидальная сортировка всегда имеет временную сложность O(nlogn)[2]
2. **Пространственная сложность:**
   Пирамидальная сортировка выполняется на месте (in-place), требуя всего (1)O(1) дополнительной памяти, что делает её привлекательным вариантом для сортировки больших объемов данных с ограниченными ресурсами памяти.

Описание входных данных и генератора

Описание входных данных

Входные данные для алгоритма пирамидальной сортировки — это массив данных, который требуется отсортировать. Этот массив может содержать любые элементы, такие как числа, строки или пользовательские объекты, которые могут быть сравнены между собой для установления порядка.

Описание генератора

Класс *RandomInputGenerator* предоставляет функцию *GenerateRandomNumArray*, возвращающую массив целых случайных чисел и получающую на вход размер этого массива, а также диапазон, в котором будут генерироваться элементы:
https://github.com/GeorgeD615/HeapSortAnalytics/blob/main/RandomInputGenerator.cs

Реализация алгоритма

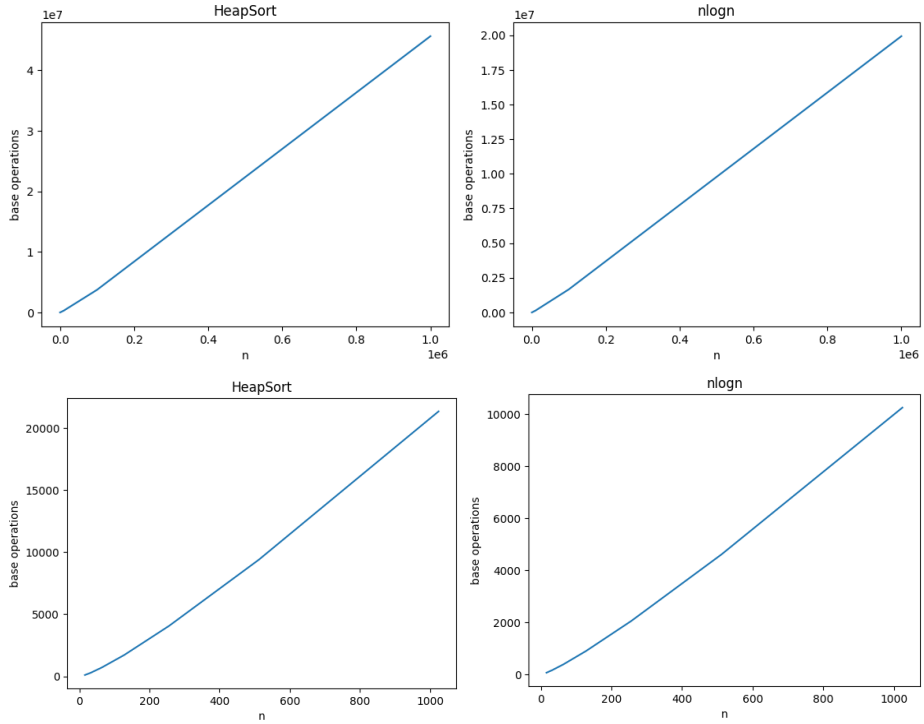
Алгоритм реализован на языке c#:
https://github.com/GeorgeD615/HeapSortAnalytics/blob/main/Program.cs

Вычислительный эксперимент

Для проведения эксперимента за единицы измерения трудоёмкости было выбрано количество базовых операций (под базовыми операциями понимаются операция сравнения и swap (поменять местами))

Количество элементов в массиве	Количество тестов	Среднее количество выполненных базовых операций
100	50	1255
1000	50	20756

10000	50	290332
100000	50	3734220
1000000	50	45616469
16	50	103
32	50	276
64	50	706
128	50	1714
256	50	4064
512	50	9398
1024	50	21340



Как видно из графиков, асимптотика похожа на теоретическую.

Анализ полученных данных

Проанализируем входные данные, в частности отношения значений измеренной трудоёмкости при удвоении размера входных данных:

$$\frac{T(32)}{T(16)} = 2{,}679612 \quad \frac{T(64)}{T(32)} = 2{,}557971 \quad \frac{T(128)}{T(64)} = 2{,}427762 \quad \frac{T(256)}{T(128)} = 2{,}371062$$

$$\frac{T(512)}{T(256)} = 2{,}3125 \quad \frac{T(1024)}{T(512)} = 2{,}270696$$

Если рассмотреть соотношение $\frac{T(n)}{n \cdot \log n}$ на больших числах, мы увидим, что оно также стремится к положительной константе, которая примерно равна 2. На основании этого можно утверждать, что алгоритм принадлежит классу сложности $O\ (n\ \log(n))$.

Источники

- [1] Статья "Assigning Meanings to Programs" (Communications of the ACM, 1964)
- [2]"Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein: В разделе 6.4 представлено обширное описание пирамидальной сортировки, включая доказательства корректности и временной сложности.