Лекция 5. Быстрая сортировка

Казахский национальный университет имени аль-Фараби

Кафедра компьютерных наук

Алгоритмы и структуры данных

Лектор: старший преподаватель, PhD, Кенжебек Ержан Ғалымжанұлы

Цель и план лекции

Цель лекции

Изучить алгоритм быстрой сортировки (QuickSort): выбор опорного элемента, схему разбиения Ломуто, а также научиться оценивать временную и пространственную сложность.

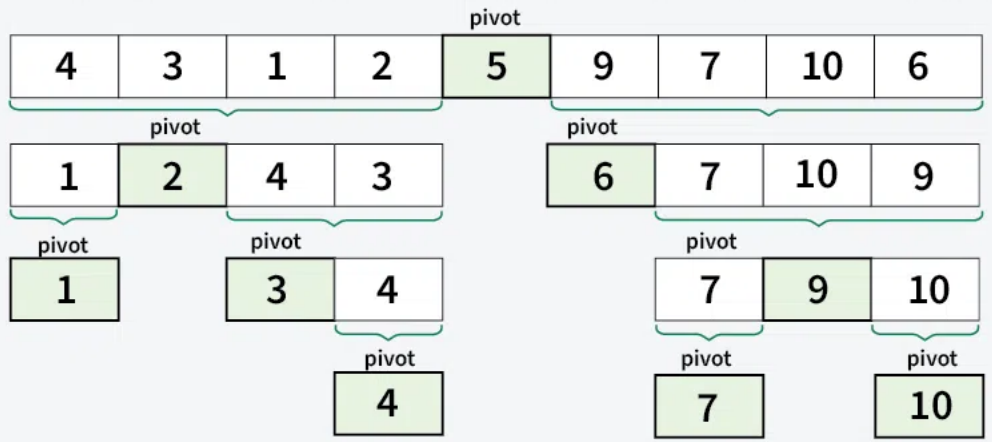
План лекции

1. Принцип «разделяй и властвуй» в QuickSort
2. Выбор опорного элемента
3. Алгоритм разделения (схема Ломуто)
4. Пошаговая иллюстрация работы
5. Анализ сложности быстрой сортировки

Quick Sort

- **Быстрая сортировка (QuickSort)** — это алгоритм сортировки, основанный на принципе «разделяй и властвуй», который выбирает элемент в качестве опорного и разделяет заданный массив вокруг выбранного опорного элемента, помещая его в правильное положение в отсортированном массиве.
- Алгоритм состоит в основном из трех шагов:
  1. **Выберите опорный элемент:** выберите элемент из массива в качестве опорного. Выбор опорного элемента может быть различным (например, первый элемент, последний элемент, случайный элемент или медиана).
  2. **Разделение массива:** перегруппируйте массив вокруг опорного элемента. После разделения все элементы меньше опорного элемента окажутся слева от него, а все элементы больше опорного элемента — справа.
  3. **Рекурсивный вызов:** Рекурсивно примените тот же процесс к двум разделенным подмассивам.
  4. **Базовый случай:** рекурсия останавливается, когда в подмассиве остается только один элемент, поскольку этот единственный элемент уже отсортирован.

Here, we have represented the recursive call after each partitioning step of the array.



Выбор поворотного момента

- Существует множество различных вариантов выбора точек опоры.
  - В качестве опорного элемента всегда выбирается первый (или последний) элемент . В приведенной ниже реализации в качестве опорного элемента выбирается последний элемент. Проблема такого подхода заключается в том, что он приводит к наихудшему результату, когда массив уже отсортирован.
  - Выберите случайный элемент в качестве опорного . Это предпочтительный подход, поскольку в нем нет закономерности, при которой происходит наихудший сценарий.
  - Выбор медианного элемента в качестве опорного является идеальным вариантом с точки зрения временной сложности, поскольку медиану можно найти за линейное время , а функция разбиения всегда делит входной массив на две половины. Однако в среднем это занимает больше времени, так как поиск медианы требует больших констант.

Алгоритм разделения

- Ключевым процессом в **быстрой сортировке** является операция **разделения (partition()).** Существует три распространенных алгоритма для разделения. Все эти алгоритмы имеют временную сложность O(n).

1. **Наивное разбиение** : Здесь мы создаём копию массива. Сначала помещаем все меньшие элементы, а затем все большие. Наконец, мы копируем временный массив обратно в исходный массив. Это требует дополнительного пространства O(n).

2. **Разбиение Ломуто** : В этой статье мы использовали именно этот алгоритм. Это простой алгоритм, в котором мы отслеживаем индексы меньших элементов и постоянно меняем их местами. Мы использовали его здесь из-за его простоты.

3. **Метод Хоара** : это самый быстрый из всех методов. Здесь мы обходим массив с обеих сторон и постоянно меняем местами больший элемент слева и меньший справа, пока массив не разбит на части.Подробнее см. сравнение методов Хоара и Ломуто .

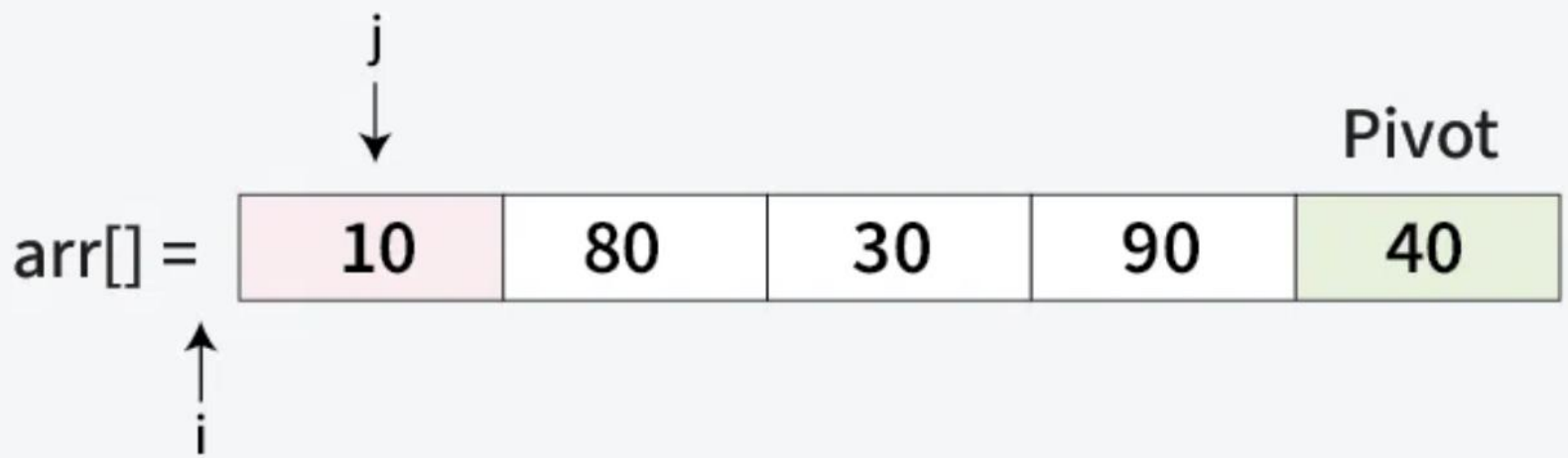
-

Принцип работы алгоритма разбиения Ломуто с иллюстрацией.

- Логика проста: мы начинаем с самого левого элемента и отслеживаем индексы меньших (или равных) элементов как **i** . Во время обхода, если мы находим меньший элемент, мы меняем местами текущий элемент с **arr[i]** . В противном случае мы игнорируем текущий элемент.

01 Step

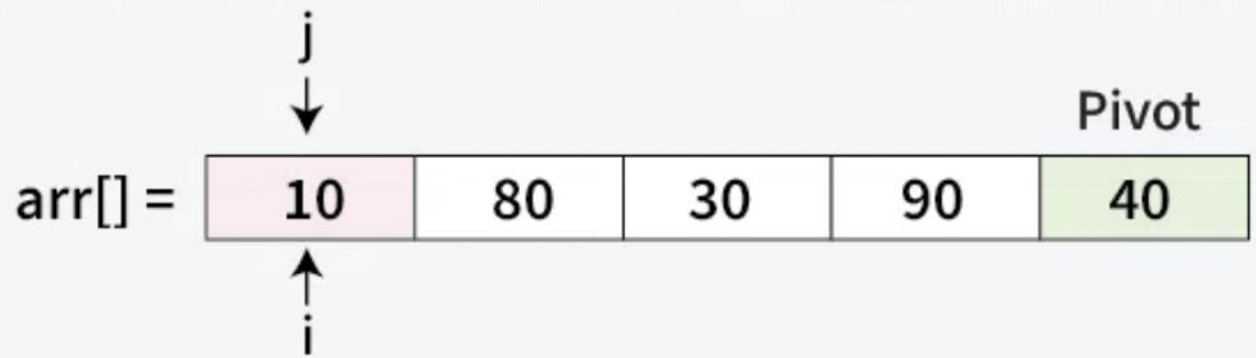
Pivot Selection: The last element arr[4] = 40 is chosen as the pivot.
Initial Pointers: i = -1 and j = 0.



Quick sort

02 Step

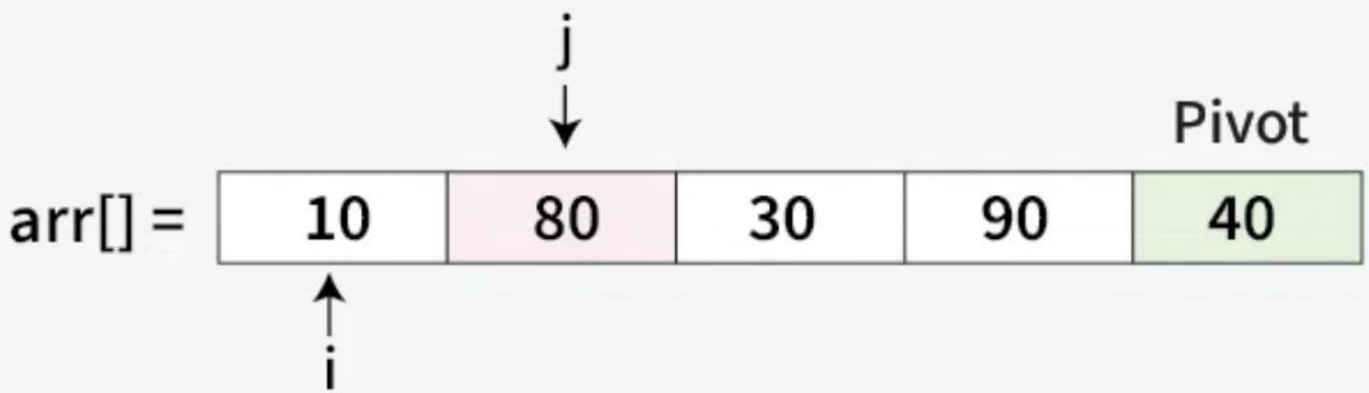
Since, arr[j] < pivot (10<40)
Increment i to 0 and swap arr[i] with arr[j]. Increment j by 1



Quick sort

03 Step

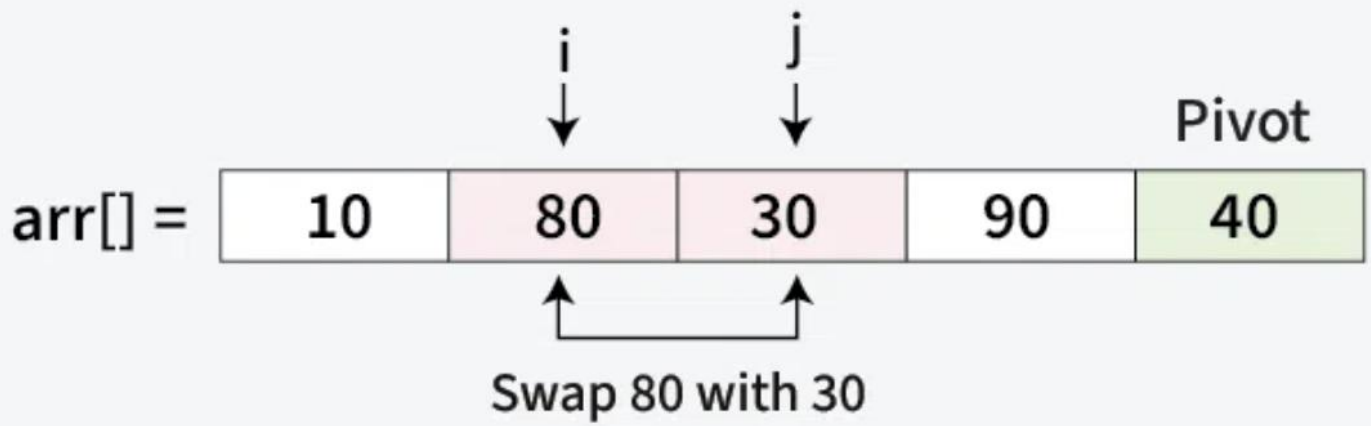
Since, arr[j] > pivot (80>40)
No swap needed. Increment j by 1



Quick sort

04 Step

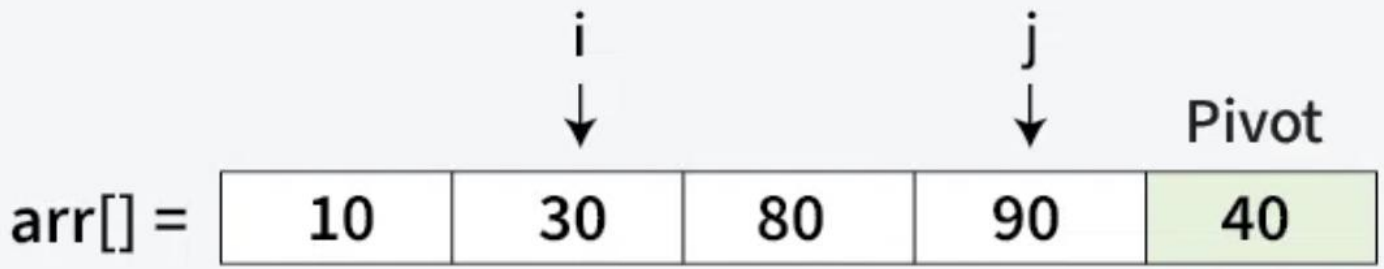
Since, arr[j] < pivot (30<40)
Increment i by 1 and swap arr[i] with arr[j]. Increment j by 1



Quick sort

05 Step

Since, arr[j] > pivot (90>40)
No swap needed. Increment j by 1



06 Step

Since traversal of j has ended. Now move pivot to its correct position, Swap arr[i + 1] = arr[2] with arr[4] = 40.

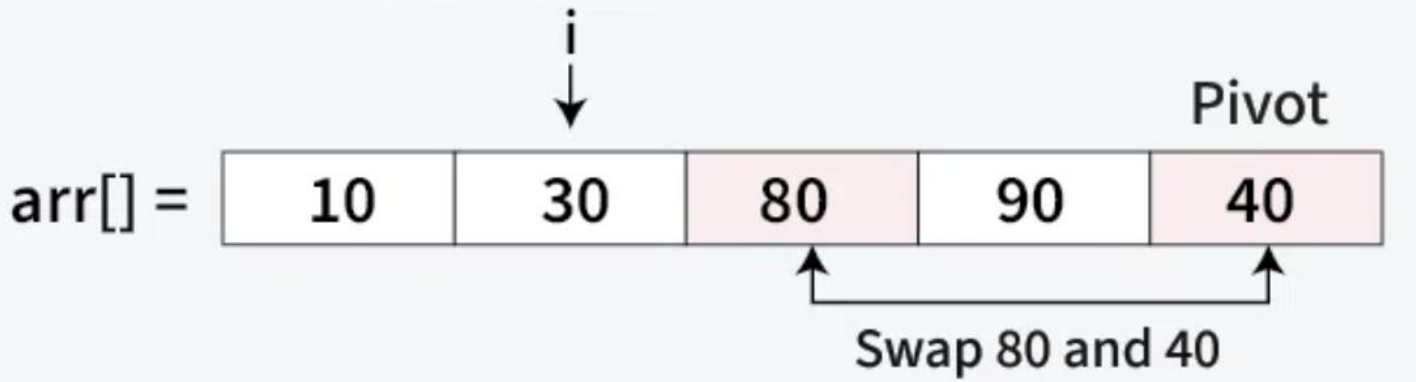
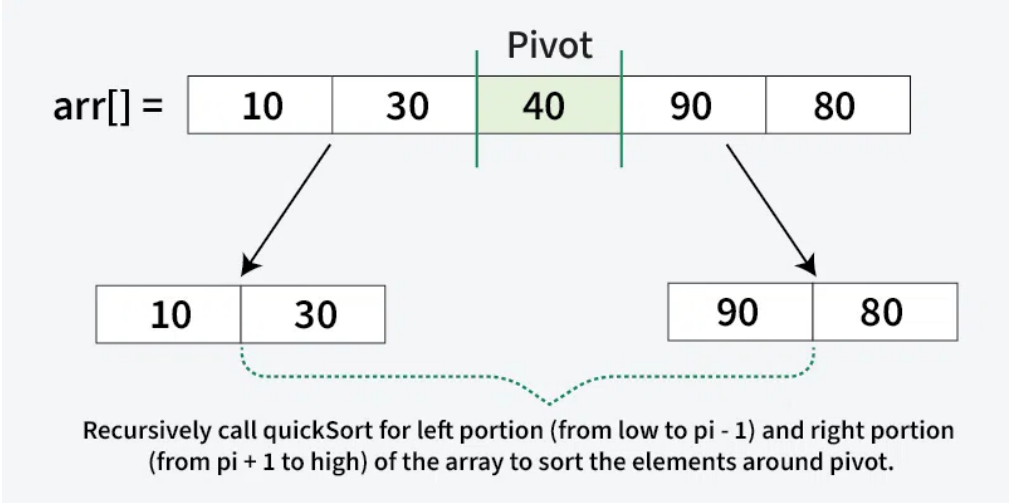




Иллюстрация алгоритма быстрой сортировки

- На предыдущем шаге мы рассмотрели, как процесс **разбиения** массива перестраивает его на основе выбранного **опорного элемента** . Затем мы применяем тот же метод рекурсивно к меньшим подмассивам слева **и** справа **от** опорного элемента. Каждый раз мы выбираем новые опорные элементы и снова разбиваем массивы. Этот процесс продолжается до тех пор, пока не останется только один элемент, который всегда отсортирован. Как только каждый элемент окажется на своем месте, весь массив будет отсортирован.

- На изображении ниже показано, как рекурсивный метод обращается к меньшим подмассивам слева **и** справа **от** опорного **элемента** :



Анализ сложности быстрой сортировки

- **Временная сложность:**
  - **Наилучший случай:** ($\Omega(n \log n)$), возникает, когда опорный элемент делит массив на две равные половины.
  - **В среднем** , опорный элемент делит массив на две части, но не обязательно равные.
  - **Наихудший случай:** ($O(n^2)$), возникает, когда в качестве опорного элемента всегда выбирается наименьший или наибольший элемент (например, в отсортированных массивах).
- **Вспомогательное пространство:**
  - **Наихудший сценарий: сложность O(n)** из-за несбалансированного разбиения, приводящего к искаженному дереву рекурсии, требующему стека вызовов размером O(n).
  - **Наилучший сценарий: сложность O(log n)** в результате сбалансированного разбиения, приводящего к сбалансированному дереву рекурсии со стеком вызовов размером O(log n).

Выводы

1. QuickSort выбирает опорный элемент, разделяет массив вокруг него и рекурсивно сортирует полученные части.
2. Эффективность зависит от выбора опорного элемента: первый, последний, случайный или медиана.
3. В лучшем и среднем случае сложность O(n log n); в худшем — $O(n^2)$, если опорным всегда оказывается наименьший или наибольший элемент.
4. Дополнительная память определяется стеком рекурсии: O(log n) в лучшем случае и O(n) в худшем.

Контрольные вопросы

1. Назовите основные шаги алгоритма быстрой сортировки.
2. Какие способы выбора опорного элемента существуют и как они влияют на скорость?
3. Как работает разбиение по схеме Ломуто?
4. При каких входных данных QuickSort работает за $O(n^2)$?
5. От чего зависит расход дополнительной памяти в QuickSort?
6. Сравните QuickSort с сортировкой вставками и выбором.

Список литературы

Основная литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. 3-е изд. — М.: Вильямс, 2013.
2. Седжвик Р., Уэйн К. Алгоритмы на Java. 4-е изд. — М.: Вильямс, 2013.
3. Кнут Д. Искусство программирования. Т. 3. Сортировка и поиск. 2-е изд. — М.: Вильямс, 2007.

Дополнительная литература

4. Скиена С. Алгоритмы. Руководство по разработке. 2-е изд. — СПб.: БХВ-Петербург, 2011.
5. Вирт Н. Алгоритмы и структуры данных. — М.: ДМК Пресс, 2010.
6. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. 4th ed. — MIT Press, 2022.

Электронные ресурсы

7. VisuAlgo — визуализация алгоритмов и структур данных: visualgo.net