Пирамидальная сортировка

Лесников Юрий, ceagest

1 Пирамидальная сортировка

Хотим отсортировать массив $\mathcal{A}$ по неубыванию.

- Построим бинарную кучу (на максимум) на основе массива $\mathcal{A}$ (в самом массиве, то есть inplace).
- Будем извлекать по очереди элементы пирамиды и ставить их в конец массива (это возможно, поскольку после извлечения из пирамиды в конце массива образуется неиспользованная ячейка (та же самая идея, что и в сортировке выбором, только эффективнее).
- В итоге получим отсортированный массив.

1.1 Анализ алгоритма

- Время работы пирамидальной сортировки составляет $\mathcal{O}(n \log n)$.
- Потребление доп. памяти пирамидальной сортировки составляет $\mathcal{O}(1)$ (если не использовать рекурсию для реализации кучи)
- Сортировка не является устойчивой.

Первое верно, так как время на извлечение из кучи составляет $\mathcal{O}(\log n)$, а построить кучу на максимум можно за $\mathcal{O}(n)$. Второе — поскольку все операции можно без создания дополнительных массивов. Третье, поскольку в пирамиде равные элементы могут уйти в разные поддеревья пирамиды.

Замечание 1.1. *Пирамида основана только на сравнениях* $\implies$ *сортировка тоже основана на сравнениях. Заметим, что она работает за* $\mathcal{O}(n \log n)$ *даже в худшем случае.*

2 Bottom-Up HeapSort

- Заметим, что каждый раз, когда мы заменяем корень, на элемент с конца (далее x), мы берем один из самых маленьких элементов в куче (ведь он находится на последнем ярусе).
- Очевидно, этот элемент будет спускаться довольно глубоко при SiftDown из корня.
- Давайте вместо этого поставим в корень $-\infty$ и просеем ее вниз (теперь достаточно сравнить два ребенка, ведь просеиваемый элемент очевидно меньше).
- После просеивания заменим $-\infty$ на x.
- Далее вызовем SiftUp.
- Такой способ неплохо сокращает число сравнений, и неплохо может ускорить алгоритм, особенно если сравнения дорогие.