**Приложение №2**

|  |  |
| --- | --- |
| **Дата** | 09.12.2023 |
| **Тема** | Построение пирамиды. Реализация пирамидальной сортировки на С#. Основные понятия оценки алгоритмов. Анализ верхней границы алгоритма. |
| **Задания** | 1. Правильно построить пирамиду. 2. Реализовать пирамидальную сортировку на С#. 3. Изучить основные понятия оценки алгоритмов. 4. Выполнить анализ верхней границы алгоритма. |

**1. Построение пирамиды**

Исходный массив делится пополам, вторая его половина уже принимается за пирамиду. Затем последовательно берутся элементы из первой половины и добавляются в пирамиду.

Может возникнуть вопрос: с чего это вдруг мы сразу полмассива считаем за пирамиду, а не строим ее шаг за шагом начиная с последнего элемента, добавляя туда потом предпоследний и т.д. до первого элемента? Дело все в том, что для второй половины нашего исходного массива основное свойство пирамиды - выполняется автоматически. Вернее, оно не нарушается, поскольку для элементов второй половины просто уже не будет потомков!!! Действительно: потомки для всех x[i], где i = n/2..n будут соответственно x[n+1]...x[2\*n+2], т.е. такие, которых в нашем массиве уже нет. А поэтому нет смысла для элементов второй половины строить пирамиду последовательно добавляя каждый элемент, т.к. в алгоритме текущий добавляемый элемент просто будет не с чем сравнивать, сыновей-то с такими индексами - просто нет!

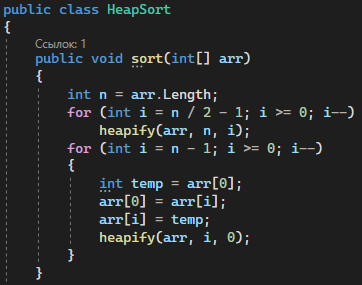
Так что спокойно принимаем вторую половину нашей последовательности как пирамиду и приступаем к следующему этапу построения - добавлению элементов.

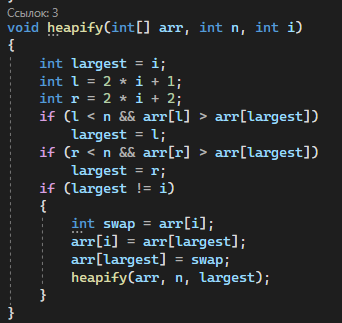
Чтобы добавить элемент в пирамиду - нужно поменять его с наибольшим ребенком, если последний превосходит добавляемый элемент. Затем тоже самое по отношению уже к новым его детям. Т.о. чтобы добавить в пирамиду новый элемент x[i], нужно:

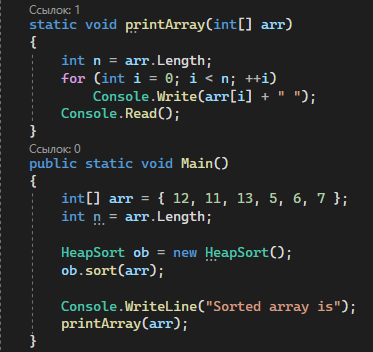
* Добавить элемент x[i] слева к массиву уже имеющейся пирамиды
* Найти максимального из его сыновей: maxChild = max( x[2\*i+1], x[2\*i+2] )
* Если x[i] >= maxChild - завершение процедуры, элемент занимает положенное ему место, добавление завершено, иначе - шаг 4.
* maxChild > x[i] - поменять их местами(т.е. поменять местами x[i] и его большего ребенка).
* Перейти к шагу 2, учитывая новое положение добавляемого элемента.

При этом, как можно догадаться - просеивание, о котором велась речь выше — это тоже добавление, поскольку после удаления максимального элемента, т.е. вершины пирамиды, мы ставим на его место последний элемент, а это посути - опять же добавление элемента в пирамиду. Поэтому для просеивания нам не нужно писать еще одну функцию, мы просто вызываем добавление по отношению к этому новому элементу.

**2. Реализация пирамидальной сортировки на С#**







**3.Основные понятия оценки сложности алгоритмов**

Основным показателем сложности алгоритма является время, необходимое для решения задачи и объём требуемой памяти.

Также при анализе сложности для класса задач определяется некоторое число, характеризующее некоторый объём данных – размер входа.

Итак, можем сделать вывод, что сложность алгоритма – функция размера входа.

Сложность алгоритма может быть различной при одном и том же размере входа, но различных входных данных.

Существуют понятия сложности в худшем, среднем или лучшем случае. Обычно, оценивают сложность в худшем случае.

Временная сложность в худшем случае – функция размера входа, равная максимальному количеству операций, выполненных в ходе работы алгоритма при решении задачи данного размера.

Ёмкостная сложность в худшем случае – функция размера входа, равная максимальному количеству ячеек памяти, к которым было обращение при решении задач данного размера.

**Порядок роста сложности алгоритмов**

Порядок роста сложности (или аксиоматическая сложность) описывает приблизительное поведение функции сложности алгоритма при большом размере входа. Из этого следует, что при оценке временной сложности нет необходимости рассматривать элементарные операции, достаточно рассматривать шаги алгоритма.

Шаг алгоритма – совокупность последовательно-расположенных элементарных операций, время выполнения которых не зависит от размера входа, то есть ограничена сверху некоторой константой.

Виды асимптотических оценок:

* O– оценка для худшего случая
* Ω – оценка для лучшего случая
* Θ – оценка для среднего случая

**4. Оценка верхней границы алгоритма**

На заре вычислительной техники предлагалось сравнивать число элементарных арифметических операций (сложения, умножения, деления), которые совершает алгоритм в процессе своей работы. Однако так можно делать или для очень простых программ или для математических алгоритмов. Еще можно попробовать замерить скорость работы алгоритма в миллисекундах. Однако на разном железе с разной производительностью мы будем получать разные результаты для одной и той же программы. И, кроме того, время работы может нелинейно возрастать с увеличением размерности (объема) входных данных. Например, все медленные алгоритмы сортировки элементов массива имеют квадратическую сложность https://proproprogs.ru/htm/structure_data/files/std-o-bolshoe-big-o--verhnyaya-ocenka-slozhnosti-algoritmov.files/image001.png, где n – число элементов массива. Поэтому такая характеристика тоже не подходит. Например, для n = 3 элементов получим одно значение времени, а для n = 10 заметно больше. И это время будет возрастать нелинейно, а квадратически! Вместо этого было предложено применять приближенную асимптотическую метрику для оценки сложности алгоритмов. В частности метрику «О большое» (от англ. Big O), которая показывает:

верхнюю границу динамики изменения вычислительной сложности алгоритма в зависимости от размера входных данных.

Чаще всего именно она интересна разработчикам. Например, для тех же медленных алгоритмов сортировки элементов массива длиной n, верхняя граница их вычислительной сложности (т.е. наихудший случай) определяется как https://proproprogs.ru/htm/structure_data/files/std-o-bolshoe-big-o--verhnyaya-ocenka-slozhnosti-algoritmov.files/image002.png. И, глядя на это выражение, мы адекватно можем представить изменение объема вычислений, при изменении величины n.

**Выводы:** Научился правильно составлять пирамиду. Реализовал пирамидальную сортировку на С#. Изучил основные понятия оценки алгоритмов. Выполнил анализ верхней границы алгоритма.

Студент: / Ульянов Никита Анатольевич