ЛАБОРАТОРНАЯ РАБОТА №3

По дисциплине: Алгоритмы и структуры данных
Тема работы: **Алгоритмы методов сортировки: сортировка с помощью двоичной кучи (Heap sort)**
Цель работы: выполнить программную реализацию и визуализацию алгоритма сортировки с помощью кучи
Количество часов: 4

Содержание работы:

1. Описание главной функции
2. Добавление функций алгоритма сортировки
3. Анализ времени работы алгоритма для разных наборов чисел
4. Выводы

Методические указания по выполнению

Описание сортировки. Бинарная пирамида (binary heap) – это структура данных, представляющая собой объект-массив, который можно рассматривать как почти полное бинарное дерево. Каждый узел этого дерева соответствует определенному элементу массива. Дерево полностью заполнено на всех уровнях, за исключением, возможно, наинизшего, который заполняется слева направо. Массив *A*, представляющий пирамиду, содержит два атрибута: количество элементов в массиве *n*, и *heap_size* – количество элементов пирамиды, которые содержатся в массиве *A*. То есть, только элементы подмассива [1..*heap_size*] являются корректными элементами пирамиды, где 0<=*heap_size*<=*n*. Корнем дерева является *A*[1], а для заданного индекса *i*-го узла можно легко вычислить индексы его родительского, левого и правого дочерних узлов.

```
Parent (i)
      return i/2

Left (i)
      return 2i

Right (i)
      return 2i+1
```

Основное свойство невозрастающей пирамиды: для каждого узла *i*, кроме корня, выполняется неравенство: $A[Parent(i)] \geq A[i]$. Наибольший элемент находится в корне дерева.

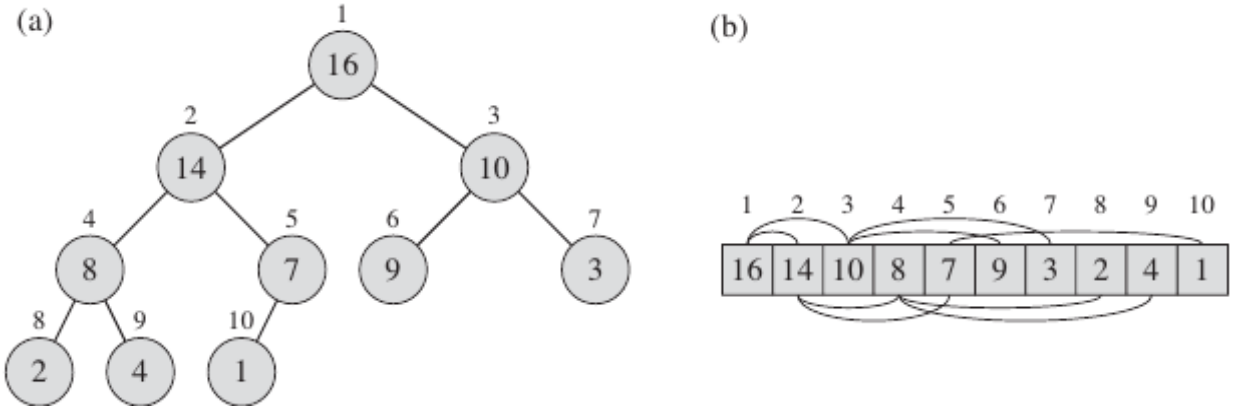

Рис. 3. Невозрастающая пирамида, представленная в виде бинарного дерева (а) и массива (б)

Высота пирамиды определяется как высота ее корня (число ребер от этого узла в самом длинном нисходящем пути к какому-то листьев дерева). Пирамида из n-элементов имеет высоту $h=lg\ n$ (где $lg\ n$ обозначает $log_2\ n$). Время выполнения основных операций в пирамиде пропорционально высоте дерева, т.е. $O(lg\ n)$.

Рассмотрим основные операции над пирамидой:

- **Heapify** – сохраняет свойство невозрастающей пирамиды ($O(lg\ n)$).

- **Build_Heap** – создает невозрастающую пирамиду из неупорядоченного входного массива (за линейное время).
- **Heapsort** – сортирует массив, не используя дополнительную память ($O(n \lg n)$).

```
Heapify (A,i)
1. l = Left (i)
2. r = Right (i)
3. if l ≤ heap_size and A[l] > A[i]
4.     largest = l
5. else largest = i
6. if r ≤ heap_size and A[r] > A[largest]
7.     largest = r
8. if largest ≠ i
9.     обменять A[i] и A[largest]
10.    Heapify (A,largest)
```

```
Build_Heap (A)
1. heap_size = n
2. for i=n/2 downto 1
3.      Heapify (A,i)
```

Алгоритм пирамидальной сортировки состоит из двух частей.

I. Сначала вызывается функция *Build_Heap()*, после выполнения которой массив становится невозрастающей пирамидой. Элемент с номером *n/2* является последним элементом, у которого могут быть дочерние узлы (узлы, начиная с узла под номером *n/2+1*, являются листьями, и уже являются упорядоченными одноэлементными пирамидами). Поэтому в *Build_Heap()* выполняется вызов функции *Heapify()* для всех узлов, начиная с узла под номером *n/2*.

В функции *Heapify()* выполняются следующие действия: выбирается больший узел из элементов $A[i]$, $A[Left(i)]$ и $A[Right(i)]$, его индекс присваивается переменной *largest* (строки 1-7). Если наибольшим узлом является один из дочерних, то узлы $A[i]$ и $A[largest]$ меняются местами (стр. 8-9). То есть для узла *i* и его дочерних восстанавливается свойство невозрастающей пирамиды. Но после этого исходное значение элемента $A[i]$ находится в узле с индексом *largest*, и поддерево с корнем *largest* может нарушать основное свойство пирамиды. Поэтому рекурсивно вызывается функция *Heapify()* уже для этого поддерева (стр. 10).

```
Heapsort (A)
1. Build_Heap(A)
2. for i=n downto 2
3.     обменять A[1] и A[i]
4.     heap_size = heap_size – 1
5.     Heapify (A,1)
```

II. После этого наибольший элемент находится в корне $A[1]$. Поэтому его следует поменять местами с последним. Затем, узел под номером *n* исключается из пирамиды: уменьшается размер пирамиды на 1 (*heap_size*). После этого восстанавливается основное свойство для оставшихся элементов в пирамиде – вызывается функция *Heapify()*. Так продолжается до тех пор, пока в куче не останется 1 элемент.

Особенности алгоритма. Использует специализированную структуру данных – пирамиду. Как и сортировка вставкой не использует дополнительной памяти, поэтому сочетает в себе плюсы сортировки слиянием и вставкой.

Время работы. Время работы, как и у сортировки слиянием, является логарифмической функцией $O(n \lg n)$.

Задачи

1. Покажите работу функции *Max_Heapify(A,3)* с массивом $A=\{25,18,5,10,15,12,3,4,2,8,6,9,7,1\}$ для поддержки свойства невозрастания пирамиды.

2. Покажите работу функции *Heapsort(A)* по сортировке массива $A=\{5,7,3,9,6,8,4,1\}$ в невозрастающем порядке.

3. Реализуйте процедуру *Heap_Delete(i)* – удаление элемента с индексом *i* из кучи.

Пособия и инструменты

1. MS Visual Studio 2008 / 2010 / 2012 / 2013
2. Data Structure Visualizations. [Электронный ресурс] – Режим доступа: http://www.cs.usfca.edu/~galles/visualization/java/download.html.

Вопросы для защиты лабораторной работы

1. Дайте асимптотическую оценку работы алгоритма пирамидальной сортировки.
2. Какие особенности у данного алгоритма.

Литература

1. Кормен Т.Х. Алгоритмы: построение и анализ, 3-е издание. : Пер. с англ. / Т.Х. Кормен, Ч.И. Лейзерсон, Р.Л. Ривест, К. Штайн. – М.: Издательский дом «Вильямс», 2013. – 1328 с.
2. Algorithms and Data Structures with implementations in Java and C++ [Electronic Resource]. – URL: http://www.algolist.net/.
3. Data Structure Visualizations / David Galles, Department of Computer Science // University of San Francisco [Electronic Resource]. – URL: http://www.cs.usfca.edu/ ~galles/visualization/Algorithms.html.
4. Data Structures and Algorithms / Java Applets Centre [Electronic Resource]. – URL: http://www.cosc.canterbury.ac.nz/mukundan/dsal/appldsal.html.
5. Анимированные визуализации структур данных / VISUALGO [Electronic Resource]. – URL: http://ru.visualgo.net/.