Problem Set No. 3
No Collaboration on 1 or 3;
Collaboration allowed on 2

1. (weighted median) (see CLRS Problem 9-2) For n distinct elements \(x_1, x_2, \ldots, x_n\) with positive weights \(w_1, w_2, \ldots, w_n\) such that \(\sum_{i=1}^{n} w_i = 1\), the weighted (lower) median is the element \(x_k\) satisfying
\[
\sum_{s_i < s_k} w_i < 1/2 \quad \text{and} \quad \sum_{s_i > s_k} w_i \leq 1/2
\]
Describe an O(n) time algorithm to find the weighted median. Hint: you may use a linear-time (unweighted) median-finding algorithm as a subroutine.

2. (thin heaps) (an alternative to Fibonacci heaps)

A thin heap is a collection of min-heap ordered trees, with ordered sibling sets, each node \(x\) of which has a rank \(r(x)\). The ranks have the following properties:

i. A node of rank \(r\) has either \(r\) or \(r-1\) children; in the former case we call the node thick; otherwise, thin.

ii. Every root is thick.

iii. The children of a thick node of rank \(r\) have ranks \(r-1, r-2, \ldots, 0\) from first to last; the children of a thin node of rank \(r\) have ranks \(r-2, r-3, \ldots, 0\) from first to last.

We call a thin heap consisting of a single tree a thin tree. We link two thin trees both of whose roots have equal rank by comparing the keys in the roots and making the root with larger key the new first child of the root of the other, and increasing the rank of the new root by one.

a. Describe how to implement make-heap, insert, minimum, extract-min, and union on thin heaps so that the worst-case time for minimum is O(1) and the amortized times of the other operations are O(1) for make-heap, O(1) for insert, O(r) for extract-min, and O(1) for union, where \(r\) is the maximum rank of any node in the heap. Hint: use an implementation and potential function similar to those for Fibonacci heaps.
b. Prove that a thin tree with a root of rank \( r \) contains at least \( F_r \) nodes. Conclude that the maximum rank of any node in a thin tree containing \( n \) nodes is \( O(\log n) \), and hence that the amortized time of extract-min is \( O(\log n) \).

c. We can perform decrease-key \((H, x, k)\) in a thin heap as follows. If \( k \) is greater than the current key of \( x \), declare an error. Otherwise, replace the current key of \( x \) by \( k \), delete \( x \) and its; subtree \( T \) from its sibling list, and make \( x \) a new root of \( H \); reduce the rank of \( x \) by one if it was previously thin. This creates a “hole” in the sibling list of the previous parent of \( x \), which may now violate (iii) above. Repair the violation by repeatedly applying the appropriate one of the following cases until no case applies:

**Case 1:** The hole is the first child of its parent. (a) If the parent is thick and not the tree root, we merely stop, the parent is now thin. If, on the other hand, the parent is thick and is the tree root, we decrease the rank of the parent by one and stop, in this case, the parent remains thick but has its rank decreased by one. (b) If the parent is thin, delete the parent and its subtree from its own sibling list, decrease the rank of the parent by two, and make it a new root. This creates a hole in the parent’s sibling list.

**Case 2:** The hole is not the first child of its parent. Let \( y \) be the predecessor of the hole in the sibling list. (a) If \( y \) is thin, reduce the rank of \( y \) by one. This makes \( y \) thick and fills the hole, but there is now a hole for the old rank of \( y \). (b) If \( y \) is thick, let \( z \) be the first child of \( y \). Delete \( z \) and its subtree from the list of children of \( y \), and use \( z \) and this subtree to fill the hole. This makes \( y \) thin, and leaves no hole.

Verify the correctness of this implementation of decrease-key. Show that the amortized time of decrease key is \( O(1) \) and the worst-case time is \( O(\log n) \).

d. Show that thin heaps can be implemented using only three printers per node, as compared to the four needed in the book’s implementation of Fibonacci heaps. Hint: the only nodes that need parent pointers are first children.

3. (Extra credit) Describe how to represent a binary tree using two pointers per node so that accessing the parent, left child, or right child of a node takes \( O(1) \) time. Apply your idea to implement thin heaps using only two pointers per node. (If necessary, you may use a bit or two per node)