Homework #2

(Due: Apr 11)

```
CLOSEST-POINT( P, n, p )
(Input is an array P[1 : n] of n > 0 points in the plane, and a special point p such that no circle centered at
p passes through more than one point in P. This function returns the point in P which is closest to p.)
1. j ← 1
2. for i ← 2 to n do
3.     if dist(p, P[i]) < dist(p, P[j]) then          {if p is closer to P[i] than P[j]}
4.         j ← i
5. return P[j]
```

Task 1. [80 Points] Average Case Analysis

The function CLOSEST-POINT($P,\ n,\ p$) given above finds the point in $P[1:n]$ which is closest to the given point p. We assume that no circle centered at p passes through more than one point in P. Observe that the number of times the assignment $(j \leftarrow i)$ in line 4 is executed depends on the order in which the points appear in P. This task asks you to compute the number of times line 4 is executed averaged over all $n!$ possible permutations of the points in P.

Let $c_{n,k}$ = number of permutations of n points for which line 4 is executed exactly k times, and also let $f_{n,k} = \frac{c_{n,k}}{n!}$ be the fraction of all possible permutations each of which results in precisely k executions of line 4. Then clearly, our required average A_n and its variance V_n are given by the following expressions.

$$A_n = \sum_k k f_{n,k} \quad \text{and} \quad V_n = \sum_k k^2 f_{n,k} - A_n^2$$

(*a*) **[20 Points]** Consider the following generating function for $f_{n,k}$'s.

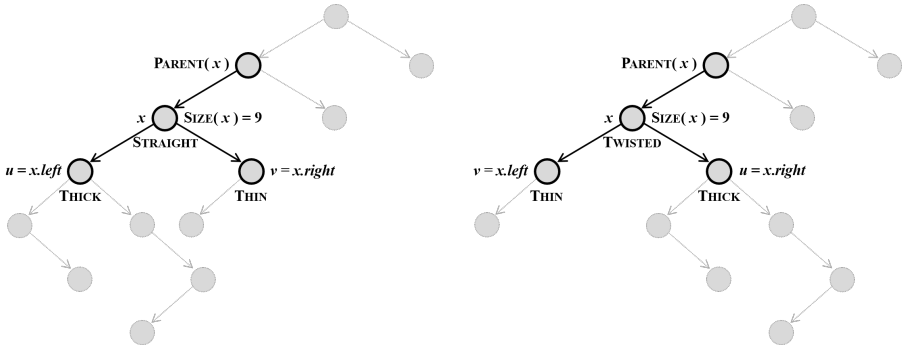
$$F_n(z) = f_{n,0} + f_{n,1} z + f_{n,2} z^2 + \ldots + f_{n,n} z^n$$

Show that $A_n = F_n'(1)$ and $V_n = F_n''(1) + F_n'(1) - (F_n'(1))^2$.

(*b*) **[20 Points]** Prove that for $n > 0$, $c_{n,k}$ can be described using the following recurrence relation.

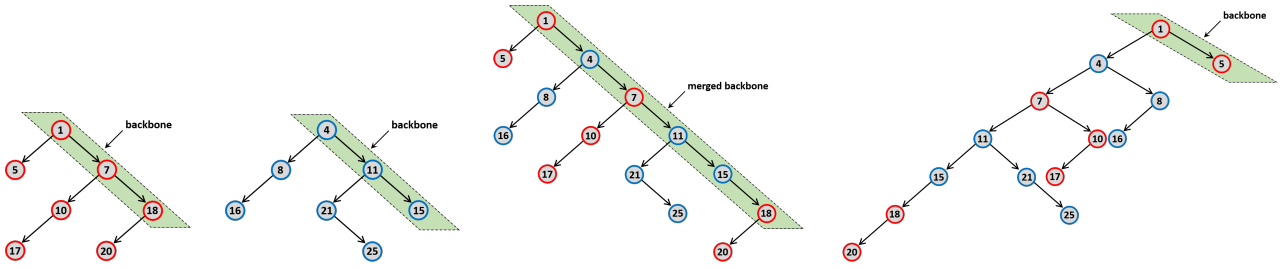
$$c_{n,k} = \begin{cases} 1 & \text{if } (n = 1 \ \wedge \ k = 0), \\ 0 & \text{if } (k < 0) \ \vee \ (n = 1 \ \wedge \ k \neq 0), \\ (n-1)c_{n-1,k} + c_{n-1,k-1} & \text{otherwise.} \end{cases}$$

(*c*) **[40 Points]** Use your results from parts (*a*) and (*b*) to show that $A_n = H_n - 1$ and $V_n = H_n - H_n^{(2)}$, where $H_n = \sum_{1 \le k \le n} \frac{1}{k} < \ln n + 1$ and $H_n^{(2)} = \sum_{1 \le k \le n} \frac{1}{k^2} < \frac{\pi^2}{6}$.



(*a*) Node x is *straight* $\Rightarrow$ its right child is thin (*b*) Node x is *twisted* $\Rightarrow$ its right child is thick

Figure 1: A non-leaf node is *straight* provided its right child is thin, otherwise it is *twisted*.



(*a*) min-heap-ordered binary tree T_1 (*b*) min-heap-ordered binary tree T_2 (*c*) after min-heap-order preserving merging of backbones of T_1 and T_2 (*d*) UNION(T_1, T_2) is obtained by twisting all but the last node on the merged backbone of T_1 and T_2

Figure 2: Unioning two min-heap-ordered binary trees.

Task 2. [100 Points] I've Come to Set a Twisted Thing Straight (Suzanne Vega/Solitude)

Consider an arbitrary binary tree T with $T.root$ pointing to the root node of the tree. If $T.root =$ NULL then T is empty. Each node x of T contains three fields: $x.key$, $x.left$ and $x.right$. An arbitrary numerical value can be stored in $x.key$. If x has a left child/subtree then $x.left$ points to that child, otherwise $x.left =$ NULL. Similarly for $x.right$. The number of nodes in the subtree rooted at x (including x itself) is given by SIZE(x). If x is not the root then PARENT(x) is the parent of x. See Figure 1 for an example. Note that x does not store SIZE(x) and PARENT(x). We have defined them for convenience.

We say that a non-root node x is *thin* provided SIZE(x) $\leq$ SIZE(PARENT(x))/2, otherwise x is *thick*. A non-leaf node x is called *straight* provided its right child is thin, otherwise we call it *twisted*. A *twist* operation on x swaps the two children of x. As a result, if x was originally twisted it becomes straight, and if it was originally straight it can either remain straight (when both its children have the same size) or become twisted. Figure 1 shows an example.

The path of T that starts from the root and follows only the right pointers is called the *backbone* of T. See Figure 2 for examples.

We assume that T is min-heap ordered. Clearly, MAKE-HEAP(T) that creates an empty min-heap ordered binary tree T, and FIND-MIN(T) that returns the minimum key in T (or NULL if T is empty) can be implemented to run in $\mathcal{O}(1)$ worst-case time.

Figures 2 and 3 show how to compute the union of two min-heap ordered binary trees by first merging their backbones and then twisting each node (except the one at the bottom) on the merged backbone in an attempt to shorten the length of the new backbone.

(a) [**10 Points**] If T has n nodes, argue that any root to leaf path of T can include at most $\lfloor \log_2 n \rfloor$ thin nodes.

(b) [**10 Points**] Use your result from part (a) to argue that the backbone of T cannot contain more than $\lfloor \log_2 n \rfloor$ straight nodes.

(c) [**20 Points**] Design INSERT(T, k) that inserts a new key k into T, and DELETE-MIN(T) that deletes the smallest key from T using the UNION function as a subroutine.

(d) [**30 Points**] Use the accounting method and your result from part (b) to prove an $\mathcal{O}(\log n)$ amortized time bound for each of INSERT, DELETE-MIN and UNION operations, where n is the total number of nodes in the tree(s) involved.

(e) [**30 Points**] Repeat part (d) using the potential method.

UNION(T_1, T_2)

(Inputs are two heap-ordered binary trees T_1 and T_2. Output is a new heap-ordered binary tree T obtained by merging the backbones of T_1 and T_2, and then twisting each node (except the last) on the new backbone.)

1. ***new tree*** T
2. $T.root \leftarrow$ MERGE($T_1.root$, $T_2.root$)
3. ***if*** $T.root \neq$ NULL ***then*** TWIST($T.root$)
4. ***return*** T

TWIST(x)

(Input is a tree node x. This function twists each node (i.e., swaps the two subtrees rooted at the node) except the last node on the backbone of the subtree rooted at x.)

1. $z \leftarrow x.right$
2. ***if*** $z =$ NULL ***then return***
3. $x.right \leftarrow x.left$
4. $x.left \leftarrow z$
5. TWIST(z)

MERGE(x, y)

(Inputs are two tree nodes x and y. The subtrees rooted at these two nodes are heap-ordered. This function merges the backbones of those two subtrees maintaining heap-order, and returns the root of the merged backbone.)

1. ***if*** $x =$ NULL ***then return*** y
2. ***else if*** $y =$ NULL ***then return*** x
3. ***if*** $x.key < y.key$ ***then***
4. $\quad x.right \leftarrow$ MERGE($x.right$, y)
5. $\quad$ ***return*** x
6. ***else***
7. $\quad y.right \leftarrow$ MERGE(x, $y.right$)
8. $\quad$ ***return*** y

Figure 3: Given two heap-ordered binary trees T_1 and T_2, UNION(T_1, T_2) returns a new heap-ordered binary tree T by merging the backbones of T_1 and T_2 maintaining heap-order, and then twisting each node (except the last) on the merged backbone.