Brief solutions to homework 4

1. **Solution to problem 1**

   (a) Fix a node v_j. It may only have incoming edges from nodes v_i with $j+1 \le i \le n$; each of the latter picks v_j as its destination node with probability $\frac{1}{i-1}$. Hence the expected number of incoming edges to v_j is given by

   $$\sum_{i=j+1}^{n} \frac{1}{i-1} = \sum_{i=j}^{n-1} \frac{1}{i} = H_{n-1} - H_{j-1} = \Theta(\ln(n-1)) - \Theta(\ln(j-1)) = \Theta\left(\ln \frac{n}{j}\right)$$

   (b) Let X be the number of nodes with no incoming edges. Let X_j be an indicator r.v. such that $X_j = 1$ if and only if node v_j has no incoming edges. Then [1]

   $$X = \sum_{i=1}^{n} X_j$$

   Note that v_j has no incoming edges if none of the outgoing edges of $v_{j+1}, \ldots, v_n$ enter v_j. Since new nodes select the destination of their outgoing edge uniformly at random among the existing nodes, and independently of the selections of previous nodes, the probability that v_j has no incoming edges is given by

   $$\begin{aligned}\Pr[X_j = 1] &= \left(1 - \frac{1}{j}\right) \cdot \left(1 - \frac{1}{j+1}\right) \cdots \left(1 - \frac{1}{n-1}\right) \\ &= \frac{j-1}{j} \cdot \frac{j}{j+1} \cdots \frac{n-2}{n-1} = \frac{j-1}{n-1}\end{aligned}$$

   By linearity of expectation, we have

   $$E[X] = \sum_{j=1}^{n} E[X_j] = \sum_{j=1}^{n} \Pr[X_j = 1] = \sum_{j=1}^{n} \frac{j-1}{n-1} = \frac{1}{n-1} \sum_{j=0}^{n-1} j = \frac{1}{n-1} \cdot \frac{n(n-1)}{2} = \frac{n}{2}$$

2. **Solution to problem 2**

   Let X_i be an indicator r.v. that takes on the value 1 if and only if we update b_{max} upon seeing the i-th bid. Then the r.v. $X = \sum_{i=1}^{n} X_i$ counts the total number of updates of b_{max}. By linearity of expectation,

   $$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \Pr[X_i = 1] \tag{1}$$

   Note that we update b_{max} upon seeing the i-th bid if it is larger than bids $1, \ldots, i-1$. Since bids appear uniformly at random, any of the first i bids is equally likely of being the largest. Hence $Pr[X_i = 1] = 1/i$. Substituting in equation (1), we get

   $$E[X] = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\ln n)$$

[1] You can get a more accurate expression for X by noting that node 1 always has an incoming edge (hence $X_1 = 0$), while node n does not (hence $X_n = 1$). This will not affect the asymptotic nature of our results.

3. **Solution to problem 3**

   (a) This is a one-sided Monte Carlo algorithm. If it returns a number, it is always a correct answer. However, it might fail to produce the correct answer (that is, return **error**) with some probability.

   (b) The running time of this algorithm is $O(n)$ (we assume we can select a random item in constant time).

   (c) The success probability of this algorithm is at least $\frac{1}{2} - \frac{1}{2n}$ for every input size n.

   (d) The failure probability of this algorithm is at most $\frac{1}{2} + \frac{1}{2n}$ for all n. To reduce the failure probability (hence amplify the success probability), we can repeatedly and independently run the algorithm k times. Then the failure probability of this new algorithm is at most

   $$\left(\frac{1}{2} + \frac{1}{2n}\right)^k,$$

   since the new algorithm fails only if each of the k independent executions of `Approximate Randomized Median` fails.

   For $n \geq 10$, the failure probability of the new algorithm is upper bounded by

   $$\left(\frac{1}{2} + \frac{1}{2n}\right)^k \leq 0.6^k.$$

   Requiring that the above is smaller than 0.01 yields that 10 iterations suffice to improve the success probability to more than 99% while maintaining a linear running time. (In fact, we are typically interested in large n in applications. In that case, $\left(\frac{1}{2} + \frac{1}{2n}\right) \approx \frac{1}{2}$. So 7 iterations would suffice for, say, $n \geq 80$.)

Solutions to recommended exercises

1. **Solution to recommended exercise 1**

   We model the problem as a graph problem. Construct a graph G as follows: add a vertex i for each currency c_i; add an edge (i, j) between every pair of nodes such that $R[i, j] > 0$ with weight $w_{ij} = -\ln R[i, j]$.

   Note that $\left(\prod_{\ell=1}^{k-1} R[i_\ell, i_{\ell+1}]\right) \cdot R[i_k, i_1] > 1$ implies that $\left(\sum_{\ell=1}^{k-1} w[i_\ell, i_{\ell+1}]\right) + w_{i_k i_1} < 0$.

   So we are looking for a negative-length cycle in G.

   A reasonable assumption is that a conversion between every pair of coins is possible. So the graph is strongly connected and there's a path between any two vertices in the graph. So we may pick any vertex as the origin vertex s and run the slightly modified Belman-Ford algorithm that computes shortest s-t paths *and* detects negative cycles reachable from s. Running time: $O(mn) = O(n^3)$.

2. **Solution to recommended exercise 2**

(a) Each iteration returns $q = r$ and the size of the subproblem is reduced by 1:

$$T(n) = T(n-1) + \theta(n) = \theta(n^2)$$

(Draw a recursion tree or use substitution to convince yourself.)

(b) See Algoritm 1 below.

Algorithm 1

PARTITION'(A, p, r)

```
x = A[r]
q = p − 1, t = p − 1
for j = p to r − 1 do
    tmp = A[j]
    if tmp ≤ x then
        t = t + 1
        exchange A[t] with A[j]
        if tmp < x then
            q = q + 1
            exchange A[q] and A[t]
        end if
    end if
end for
exchange A[t+1] with A[r]
return (q+1, t+1)
```

(c) See Algorithm 2 below.

Algorithm 2

RANDOMIZED-PARTITION' calls **PARTITION'**

(rest everything is same)

QUICKSORT'(A, p, r)

```
while p < r do
    q, t = PARTITION'(A, p, r)
    QUICKSORT'(A, p, q-1)
    QUICKSORT'(A, t+1, r)
end while
```

(d) Let the sorted permutation be $z_1, \ldots, z_n$,

$$\text{Probability that } z_i \text{ and } z_j \text{ are compared} = \frac{2}{j - i + 1 + l}$$

where $l =$ number of elemets z_k such that $z_k = z_i$ or $z_k = z_j$ and $k < i$or $k > j$

$$\text{since } l \geq 0, \frac{2}{j-i+1+l} \leq \frac{2}{j-i+1}$$

$\Rightarrow$ The expected number of comparisons in the modified analysis are no more than that in the original analysis.

$$\Rightarrow T'(n) = O(n \log n)$$

3. **Solution to recommended exercise 3**

   (a) Note that the tail recursive algorithm makes the same calls to PARTITION in terms of A, p, and r values.

   (b) An input permutation which is already sorted in ascending order will reduce the size of the subproblem by 1 in each recursion and hence the depth of the stack would be $\Theta(n)$

   (c) The key is to recurse on the shorter subarray: i.e.

if $(q-p) > (r-q)$ **then**
    recurse on (A, q+1, r) // Right subarray
    $r = q - 1$
else
    recurse on (A, p, q-1) // Left subarray
    $p = q + 1$
end if

4. Recommended exercise 4 is not part of the second exam.