CMPS 102
Hw8 Solutions

1. Let m be an integer in the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and consider the following problem: determine m by asking 3-way questions, i.e. questions with at most 3 possible answers. For instance, one could ask which of 3 specific subsets m belongs to.

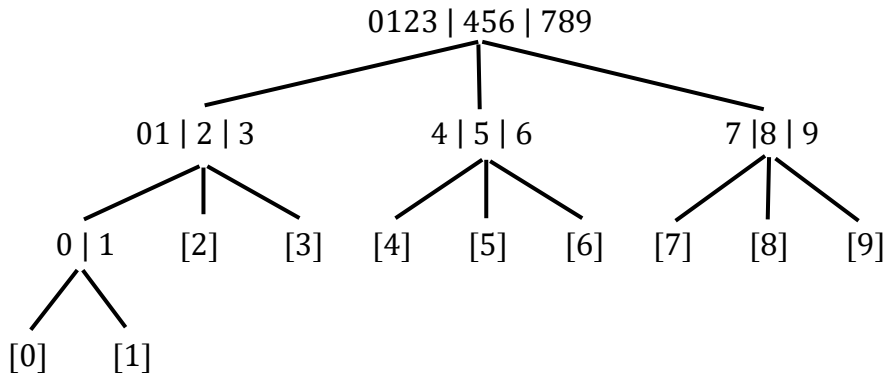
   a. Give a decision tree argument showing that at least 3 such questions are necessary in worst case. In other words, prove that no correct algorithm can solve this problem by asking only 2 questions in worst case.

      Proof:
      Any algorithm that solves this problem can be represented by ternary decision tree, since each question can have at most 3 answers. Since there are 10 possible verdicts, the height of such a tree must satisfy $h \geq \lceil \log_3(10) \rceil = 3$. Therefore no such algorithm can ask less than 3 questions in worst case. ■

   b. Design an algorithm that will solve this problem by asking 3 such questions in worst case. Express your algorithm as a decision tree.

      Solution:
      Each internal node below represents a question asking whether m belongs to one of 3 possible subsets of $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. For instance $0123 \mid 456 \mid 789$ represents the question: "does m belong to $\{0, 1, 2, 3\}$, to $\{4, 5, 6\}$, or to $\{7, 8, 9\}$?" Verdicts are placed in brackets "[ ]".

      Decision tree:



2. **Bar Weighing Problem**
Assume we are given 12 gold bars numbered 1 to 12 where 11 bars are pure gold and one is counterfeit: either gold-plated lead (which is heavier than gold), or gold-plated tin (lighter than gold). The problem is to find the counterfeit bar and what metal it is made of using only a balance scale.

Any number of bars can be placed on each side of the scale, and each use of the scale produces one of three outcomes: either the left side is heavier, or the two sides are the same weight, or the right side is heavier.

a. Give a decision tree lower bound for the (worst case) number of weighing's that must be performed by any algorithm solving this problem.

Theorem
Any algorithm that solves this problem must perform at least 3 weighing's in worst case.

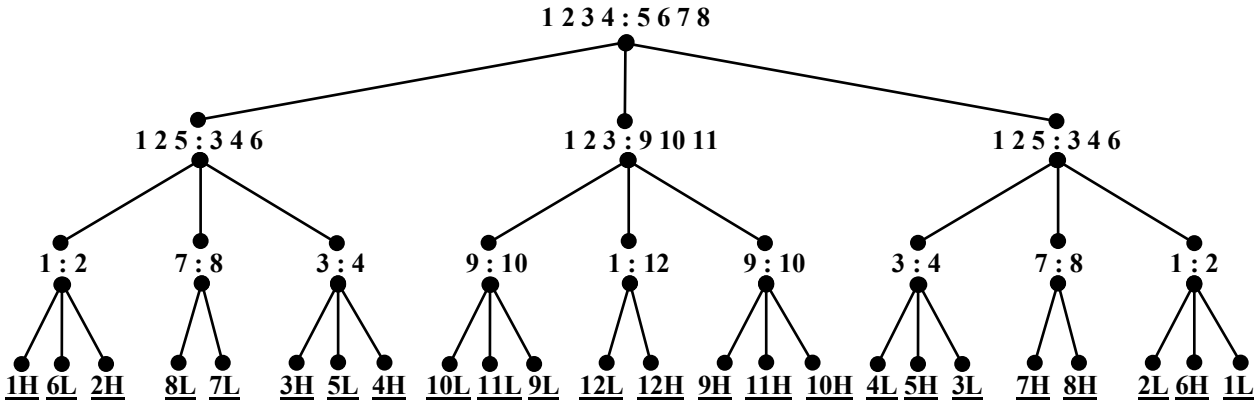
Proof: Since there are 3 possible outcomes to each weighing, the decision tree for this problem is a ternary tree. Observe that there are 24 distinct verdicts, which may be represented as: $\{1L, 2L, 3L, \ldots, 12L, 1H, 2H, 3H, \ldots, 12H\}$. For instance, $8L$ is the conclusion that bar 8 is light (i.e. tin) and all other bars are genuine, while $11H$ represents the conclusion that bar 11 is heavy (lead) and all others are pure gold.

The height h of a decision tree representing any algorithm for this problem must therefore satisfy $h \geq \lceil \log_3(24) \rceil = 3$. Therefore at least 3 weighing's must be performed (in worst case) to identify the counterfeit bar. ■

b. Design an algorithm that solves this problem with (worst case) number of weighing's equal to the lower bound you found in (a). Present your algorithm by drawing a decision tree, rather than pseudo-code.

Solution:
The following decision tree represents one algorithm (among many) that identifies the counterfeit bar in exactly 3 weighing's.

Each internal node represents a particular set of bars on the left and right side of the balance. For instance, a node labled `abc:def` would mean to place bars `a`, `b` and `c` on the left side of the balance, and place bars `d`, `e` and `f` on the right. The left child is taken if the balance tilts left, indicating that the combination `abc` is heavier than `def`. Similarly the right child is taken if the balance tilts right, and the middle child is taken if the balance does not move, which implies the bar combinations `abc` and `def` are equal in weight.



c. Alter the problem slightly to allow the possibility that all 12 bars are pure gold. Thus, there is one additional possible verdict: "all gold". Make a minor change to your algorithm in part (b) so that it gives a correct answer to this more general problem.

Solution:
Observe that if all bars are pure gold, then all weighing's (of equal numbers of bars) will balance. Therefore, by adding a single leaf to the above tree labeled AllGold as the middle child of the 1:12 node, we obtain an algorithm that solves this problem in the more general setting.

3. **Water Jug Problem** (Problem 8-4: page 206 of CLRS 3rd edition)
Suppose that you are given *n* red and *n* blue water jugs, all of different shapes and sizes. All red jugs hold different amounts of water, as do the blue ones. Moreover, for every red jug, there is a blue jug that holds the same amount of water, and vice versa.

It is your task to find a grouping of the jugs into pairs of red and blue jugs that hold the same volume of water. To do so, you may perform the following operation: pick a pair of jugs, one red, one blue, fill the red jug with water, and then pour the water into the blue jug. This operation will tell you if the two jugs hold the same amount of water, and if not, which one holds more water. Assume that such a comparison takes one unit of time. Your goal is to find an algorithm that solves this problem. Remember that you may not directly compare two red jugs or two blue jugs.

a. Find an algorithm that uses $\Theta(n^2)$ comparisons (in worst case) to group the jugs into pairs.

Solution:
Call the red jugs $R_1, R_2, \ldots, R_n$ and the blue jugs $B_1, B_2, \ldots, B_n$. Begin by comparing R_1 to each of the n blue jugs until a match is found. At most $n-1$ comparisons are necessary to determine a match, since if no match has been found during $n-1$ comparisons, the last blue jug must match, and no additional comparison is necessary. Next compare R_2 to the $n-1$ unmatched blue jugs. This time at most $n-2$ comparisons are necessary to determine a match. Next

compare R_3 to at most $n-3$ blue jugs. In general, one compares R_i to at most $n-i$ blue jugs. The last red jug R_n doesn't need to be compared to any blue jugs since there is only one possibility left at that point. Once this process is complete, a matching of red and blue jugs has been attained. The total number of comparisons performed is at most

$$\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Theta(n^2)$$

■

b. Prove a lower bound of $\lceil \log_3(n!) \rceil$ for the worst case, and $\log_3(n!)$ for the average case number of comparisons to be performed by any algorithm that solves this problem.

Proof:
Each comparison described above has 3 possible outcomes, and hence any algorithm solving this problem is represented by a ternary tree. There is a total of $n!$ ways to match the *n* red jugs to the *n* blue jugs, so this problem has $n!$ possible verdicts. Any algorithm solving this problem can be represented by a decision tree whose height satisfies $h \geq \lceil \log_3(n!) \rceil$ and whose average height satisfies $a \geq \log_3(n!)$. These quantities therefore serve as lower bounds on the worst case and average case number of comparisons, respectively. ■

c. Find an algorithm for this problem that runs in (average case) time $\Theta(n \log n)$. (Hint: modify the algorithms Partition() and Quicksort().)

Solution:
The input for our algorithm will be two arrays $A[1 \cdots n]$ and $B[1 \cdots n]$ each containing the same set of *n* distinct numbers, in different orders. Our goal is to simultaneously sort these two arrays without ever comparing two elements in the same array. The desired matching is then given by $A[i] == B[i]$ for $1 \leq i \leq n$. The following algorithms Match() and MatchPartition() are based on Quicksort() and Partition(), respectively. Both algorithms $\text{Match}(A, B, p, r)$ and its subroutine $\text{MatchPartition}(A, B, p, r)$ have as precondition that $A[p \cdots r]$ and $B[p \cdots r]$ contain the same $r - p + 1$ distinct elements.

Match(*A*, *B*, *p*, *r*)
1. if $p < r$
2. $\quad q = \text{MatchPartition}(A, B, p, r)$
3. $\quad \text{Match}(A, B, p, q-1)$
4. $\quad \text{Match}(A, B, q+1, r)$

The subroutine $\text{MatchPartition}(A, B, q, r)$ re-arranges the subarrays $A[p \cdots r]$ and $B[p \cdots r]$, returning an index *q* such that $A[q] == B[q]$ and such that both

$$A[p \cdots (q-1)] < A[q] < A[(q+1) \cdots r],$$

and

$$B[p \cdots (q-1)] < B[q] < B[(q+1) \cdots r].$$

The precondition on $A[p \cdots r]$ and $B[p \cdots r]$ implies, with the above inequalities, that the subarray pairs $A[p \cdots (q-1)]$, $B[p \cdots (q-1)]$ and $A[(q+1) \cdots r]$, $B[(q+1) \cdots r]$ satisfy the very same condition, namely that each pair contains the same distinct elements. This guarantees that the recursive calls to Match() on lines 3 and 4 will be valid.

MatchPartition(A, B, p, r)

```
1.  i = p − 1
2.  for j = p to r − 1
3.      if A[j] == B[r]
4.          A[j] ↔ A[r]  (swap)
5.      if A[j] < B[r]
6.          i++
7.          A[j] ↔ A[i]
8.  i = p − 1
9.  for j = p to r − 1
10.     if B[j] < A[r]
11.         i++
12.         B[j] ↔ B[i]
13. A[r] ↔ A[i + 1]
14. B[r] ↔ B[i + 1]
15. return (i + 1)
```

Lines 1-7 partition $A[p \cdots r]$ around the pivot $B[r]$, along the way finding the pivot in $A[p \cdots r]$ and placing it at $A[r]$. At this point we have $A[p \cdots i] < B[r] < A[(i+1) \cdots (r-1)]$ and $A[r] == B[r]$. Lines 8-12 similarly partition $B[p \cdots r]$ around $A[r]$. Lines 13 and 14 correctly place the pivot in both arrays, and line 15 returns the pivot index. Observe that the comparisons on lines 3, 5 and 10 are always between elements of different arrays. Thus MatchPartition() performs $3(n-1)$ such comparisons on a subarray of length *n*.

The analysis of the (average case) runtime of $\text{Match}(A, B, 1, n)$ is almost identical to that of $\text{Quicksort}(\text{A}, 1, \text{n})$. (See the lecture notes on the average case runtime of Quicksort().) The recurrence for of $\text{Match}(A, B, 1, n)$ is

$$t(n) = 3(n-1) + \left(\frac{2}{n}\right) \cdot \sum_{q=1}^{n-1} t(q)$$

One checks that the effect of the factor 3 in the first term is to multiply the solution itself by 3. Thus $t(n) = 3(-4n + 2(n+1)H_n) \sim 6n \ln(n) = \Theta(n \log n)$. ■