Question #1. The University fundraising department has managed to purchase an odd lot of water bottles at a steep discount. The lot includes n Yeti and n Stanley bottles of various shapes and capacities. Some bottles are cylindrical, some are spherical, and some are rectangular. In addition the colors of the bottles are random. The fundraising department wants to give each donor to the "Fill-er-up" campaign one Yeti and one Stanley bottle where each bottle in the pair holds the exact same amount of liquid. The seller of the odd lot has guaranteed that for every Yeti bottle in the odd lot there is exactly one Stanley of the same capacity. Unfortunately, the "odd" part of the odd lot, in addition to the weird shapes and colors, is no bottle is marked with its capacity.

You have been asked to devise an algorithm to match pairs of bottles by their capacity. Each matched pair of bottles must be one Yeti and one Stanley bottle of the same capacity. However, the fundraising folks don't want to spend any money on any fancy testing equipment so the only test you can perform to match a pair of bottles is to fill one bottle with water and then pour the contents into the other bottle. This test will tell you if the two selected bottles have the same capacity. Your goal is to devise an algorithm that minimizes the number of tests required to match up all n pairs of bottles in the odd lot.

a.) Devise an algorithm to match the water bottles where each pair is one Yeti and one Stanley.

Answer. Assume if the first condition in the if statement is false, the second condition will not be checked. Also assume the implementation of the set is a hashed set with a perfect hash function and has an insert and read time complexity of $O(1)$ (makes use of a hash table) [2]. Next, assume the function test returns true if the bottles have the same capacity and false otherwise. The algorithm is shown below:

```
//Input: 2 arrays yeti and stanley with n elements
//Output: a set of pairs with a yeti and stanley
          of the same capacity
procedure cups(yeti[1...n], stanley[1...n]) //n > 0
1.      pairs := ∅
2.      for i := 1 to n:
3.          for j := 1 to n:
4.              if((yeti[i], stanley[j]) ∉ pairs
                    and test(yeti[i], stanley[j])):
5.                  pairs ∪ {(yeti[i], stanley[j])}
6.      return pairs
end procedure
```

b.) Provide a convincing argument that your algorithm is correct.

First, the algorithm will terminate as it loops a finite number of times using a for loop. Next, the algorithm checks every Yeti against every Stanley to determine if they are the same capacity by looping through both arrays, they will not be checked if the pair of cups have already been paired. The result is all of the cups paired together which is stored in

the hashed set pairs.

c.) Show the worst case performance of your algorithm.

The cost table for the search procedure is shown below:

line	cost
1	1
2	$n+1$
3	$n(n+1)$
4	n^2
5	n^2
6	1

Therefore, the total cost of the algorithm is:

$$T(n) = 3 + n + 2n^2 + n(n+1) = 3 + 2n + 3n^2 \in O(n^2).$$

Question #2. Solve the following recurrence exactly for n assuming n is a power of 2.

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2T(n/2) + \lg(n), & \text{otherwise} \end{cases}$$

First, to solve the recurrence, using the fact the n is a power of 2, the following substitution will be made:

$$k = \lg(n)$$
$$n = 2^k.$$

Using the substitution, the recurrence becomes:

$$T(2^k) = \begin{cases} 1, & \text{if } k = 0 \\ 2T(2^{k-1}) + k, & \text{otherwise} \end{cases}$$

Next, we will specifically look at

$$2T(2^{k-1}) + k$$

which is the same as

$$2(2T(2^{k-2}) + k - 1) + k = 2^2T(2^{k-2}) + 2^1k - 2^1 + 2^0k$$

which is the same as

$$2^2(2T(2^{k-3}) + k - 2) + 2^1k - 2^1 + 2^0k$$

which is equal to

$$2^3T(2^{k-3}) + 2^2k - 2^2(2) + 2^1k - 2^1 + 2^0k.$$

Using summation notation, on the i^{th} substitution we get:

$$T(2^k) = 2^iT(2^{k-i}) + k\Sigma_{m=0}^{i-1}2^m - \Sigma_{m=1}^{i-1}2^m m.$$

Next we will use the following summation formulas found in [1]:

$$\Sigma_{i=0}^{n}2^i = 2^{n+1} - 1$$
$$\Sigma_{i=1}^{n}i2^i = 2^{n+1}(n-1) + 2$$

in $T(2^k)$ below:

$$T(2^k) = 2^iT(2^{k-i}) + k(2^{i-1+1} - 1) - 2^{i-1+1}(i - 1 - 1) - 2$$

which is equal to:

$$T(2^k) = 2^iT(2^{k-i}) + k(2^i - 1) - 2^i(i - 2) - 2.$$

Next, since we want to make substitutions until $k = 0$, i.e., we will make k substitutions. Therefore, we will substitute $i = k$ below:

$$T(2^k) = 2^kT(2^{k-k}) + k(2^k - 1) - 2^k(k - 2) - 2.$$

Since $k - k = 0$, and $2^0 = 1$, which is the recurrences base case being 1, we will multiply $T(1)$ as shown below:

$$T(2^k) = 2^k + k(2^k - 1) - 2^k(k - 2) - 2.$$

Next we will distribute everything out below:

$$T(2^k) = 2^k + k2^k - k - k2^k + 2(2^k) - 2.$$

Simplifying the above expression results in:

$$T(2^k) = 3(2^k) - k - 2.$$

Reversing the substitution of n gives us:

$$T(n) = 3n - \lg(n) - 2.$$

Question #3. Consider an unordered array A of n integers where duplicates are allowed. Give an algorithm to remove all duplicates from the array in $O(n)$.

a.) Devise an $O(n)$ algorithm to remove the duplicates.

Answer. Assume the implementation of the set is a hashed set with a perfect hash function and has an insert and read time complexity of $O(1)$ (makes use of a hash table) [2]. Also assume the set can be iterated in the same form as an array. The algorithm is shown below:

```
//Input: An array A with n elements
//Output: The modified array A with distinct elements
procedure removeDuplicates(A[1...n])
1.      S := ∅
2.      count := 0
3.      for i := 1 to n:
4.          if(A[i] ∉ S)
5.              count := count + 1
6.          S ∪ {A[i]}

        //copy set S to A
7.      for i := 1 to count:
8.          A[i] = S[i]

        //delete remaining spots in A
9.      for i := count + 1 to n:
10.         delete A[i]
end procedure
```

b.) Provide a convincing argument that your algorithm is correct.

First, the algorithm will terminate as all loops loop a finite number of times using for loops. Next, the algorithm adds an item to the set S which is the collection of unique items in the set. Whenever a unique item is found (by the condition A[i] $\notin S$) a count variable is increased. The next for loop copies the contents of the set into the array using the count variable, and the final array deletes (removes) the remaining items in the array also using the count variable. The count variable functions as the last index in the new version of A (A with only the distinct elements) ensuring A is modified with only the distinct elements.

c.) Show the worst case performance of your algorithm.

The cost table for the search procedure is shown below:

line	cost
1	1
2	1
3	$n+1$
4	n
5	n
6	n
7 + 9	$n+2$
8 + 10	n

Therefore, the total cost of the algorithm is:

$$T(n) = 5 + 6n \in O(n).$$

Sources:

[1]: Introduction to the Design and Analysis of Algorithms, Levitin, 3rd Edition, ISBN: 978-0-13-231681-1

[2]: https://www.geeksforgeeks.org/hash-table-data-structure/