CS3230 Challenge 2

Name: Guo Bohao Tutorial Group: T09 Student ID: A0117428E

Question Statement: In this task, you are to find a hidden permutation P = (p 1 ,p 2 ,...,p n) contained inside a black box.

You can ask the black box some questions of form Q = {q 1 ,q 2 ,...,q k }, where Q is a subset of the set {1,2,...,n}.The black box will return you the set {p q 1 ,p q 2 ,...,p q k } by printing all its elements in a random order. Find the best lower bound for the number of questions required to find P. The best lower bound is a function f(n) such that to find P of size n you must ask at least f(n) questions, and there exists an algorithm to find P using exactly f(n) questions. Also, design an algorithm to find P by asking f(n) questions.

- You are encouraged to present your algorithm in pseudo code, clearly and understandably.
- Analyse the time complexity of your algorithm.
- Prove the correctness of your algorithm.

Solution:

Idea:

We first think of the answers to the questions posed to the black box as sets of elements in a Venn diagram. The total number of elements that exist in the Venn diagram is set equals to the number of elements in the black box since we need to have "seen" all elements, via the different questions, in the black box to discover the hidden permutation order.

We know that for a Venn diagram with K sets, there are at most 2^K disjoint regions formed in the Venn diagram. In addition, we assume that we do not know anything about the elements in the black box beforehand such as what are the different elements in the box. This prevents us from using the region between the universal set and the union of the K sets (i.e. Universal set ∩ (Union of K sets)) since we can't determine the Nth element by taking the universal set and excluding the (N-1 elements) in the K sets since the universal set is not known beforehand, hence we are left with 2^K -1 useable regions.

We then think of asking K questions as placing elements of the black box into the K sets, obviously one element can occupy multiple sets at the same time. In order to exactly determine hidden permutation in the black box, in other words, the order of each element, we have to place the elements such that at most one element exist in each disjoint region in the Venn diagram. We see this has to be true since if two or more elements exist in the same disjoint region, we do not have any information to further differentiate between the aforementioned elements.

We then show that in order to exactly determine the hidden permutation in the black box with n elements, **we need to ask at least ceil[$\log_2(n+1)$] questions.**

Suppose we only ask ceil[$\log_2(n+1)$]-1 = k – 1 questions,

The maximum number of regions from k – 1 questions = 2^(k – 1) - 1

Max number of regions = (2 ^ (ceil[$\log_2(n+1)$])) /2 – 1 ≤ 2n /2 – 1 = n – 1 for n > 1

By the pigeonhole principle, when we have n elements and n-1 disjoint regions, there is at least one region with 2 elements which prevents us from determining exactly the hidden permutation of the black box. **Hence by proof by contradiction, we have to ask at least ceil[$\log_2(n+1)$] questions.**

Algorithm:

Using the below algorithm, we only have to ask ceil[$\log_2(n+1)$] questions to determine the hidden permutation which is the lower bound found above.

Generating the questions to be asked to the black box:

We note that each disjoint region can be generated by considering the following intersections of the following K sets, A1, A2, A3… AK for the K questions being asked.

Define Ai ‘ := the complementary set to Ai for i from 1 to K

Region 1 = A1 ∩ A2 ∩ A3… ∩ AK – this is the region that is common to all K sets

Region 0 = A1 ‘ ∩ A2 ‘ ∩ A3 ‘ … ∩ AK’ – this is the region between the universal set and the union of the K sets which was mentioned earlier and we will ignore this region as it is not useable to us.

…. (One to One correspondence between each region and the Ai / Ai ‘ combination)

Region j = A1 ‘ ∩ A2 ‘ ∩ A3 ‘ … ∩ AK’ – each disjoint region corresponds to a particular combination of whether to use the set Ai or Ai ‘, for i from 1 to K, in the overall intersection

From here, we once again see that the maximum number of disjoint region is 2^K since the choice to take Ai or Ai ‘ is independent of the other decisions and there are only two choices and we minus 1 for the region 0 that is not useable.

Next, we generate the question using the following strategy,

First, we initialise a vector (size K) of vector of int, listOfQuestions

Iterate using (m = 1; m < 2^K; m++)

The number 2^K in binary has K+1 bits hence we iterate up to 2^K -1 which has K bits that all have value 1, this upper bound corresponds to region 1 above. When m = 1, there is only one bit with value 1 in the 0-th bit, this corresponds to the region where all sets except A1 has been replaced by its complementary set.

As we iterate through, we further iterate through each bit of m and use a bit mask to check the value of each bit, if the bit value at the i-th bit of the number m is 1, we add the number m to the vector, listOfQuestion[i], remember that bit numbering i starts from 0 to K-1th bit.

Each vector in listOfQuestion is then used as the K questions asked to the black box.

Correctness:

Using this "bit-counting" strategy, we guarantee there is at most one element in each disjoint region. And furthermore, by doing the respective comparisons between the answer sets received, we can recover the element at a particular index in the hidden permutation by exploiting the 1-to-1 correspondence between the element at a disjoint region and the Ai / Ai ' combination. However, to get the complement of a particular answer set, we first get the universal set by combining the answer sets by union, next we remove elements in ANS_i from the universal set to get ANS_i ', the complementary set. As pre-processing, we can first generate all the complementary sets first to ease later calculations. To generate the hidden permutation, we just need to find the elements for each index from 1 to N.

e.g. to obtain the element at index 2 – binary representation of 2 is 000…010, hence every answer except ANS_2 is replaced by its complement.

We find the common element between answer set 2 ANS_2 and the complement sets of every answer set except answer set 2.

Which is equivalent to getting A1 ' ∩ A2 ∩ A3 ' … ∩ AK'

Time complexity:

To generate the questions:

We iterate from 1 to 2^K where 2^K = N : O(N)

We iterate through each bit of m, which has total K bits, where K ≈ O(lg N)

Overall Time Complexity: O(N lg N)

To generate the hidden permutation from the answer sets returned by the black box:

Pre-processing – create the complementary sets and the universal set:

Merge K lists of O(N) elements into universal set using hashset: average case O(K N) = O(N lg N)

Create K complementary sets using hashmap: average case O(K N) = O(N lg N)

Finding corresponding element for one index using described strategy using hashset: average case O(K N) since there are K lists of O(N) elements to compare to find the common element

Finding hidden permutation = finding for each index from 1 to N: average case O(K N^2) =O(N^2 lg N)

Overall Time Complexity: average case O(N^2 lg N), worst case(N^3 lg N)