CSE 6140

Assignment 1

Bhanu Verma - 903151012

**Question 1** - Simple Complexity



a. 443

b. 1002

c. 10002

d. n

e. n

f.

* 1. Time Complexity -
* Data Size - n
* Given code has two while loops, outer one iterates for n times. So, we get from there.
* Inner loop counts the number of powers of 3 in n, and hence time complexity for this step becomes 3 which is nothing but .
* Hence, total time complexity for the given algorithm becomes .
* Given algorithm outputs every number starting from 1 till n and the number of powers of 3 contained in that number in the form of a tuple. Here is the output for n = 9 - (1,0), (2,0), (3,1), (4,0), (5,0), (6,1) …. (9,2).

Question 2 -

**Question 3** - Divide & Conquer

Let’s have a look at the data given in the question:

* Number of Tags - n
* Each tag can be represented as i where
* Rule: if then they light up on touching

Here is the algorithm that uses Divide & Conquer approach to solve this problem in :

* Keep on dividing the problem size into two equal halves, until problem size becomes 1.
* Use recursion for above step
* Base Case - when problem size becomes 1, return that tag
* Now for each level above base level, tap two tags i.e. check if their ids are same.
* If they light up i.e. if their ids are same return any tag to the upper level.
* If they don’t light up, check if any of them is in the majority by comparing each of these tags with the problem size at current level for that node.
* If in majority, return that tag. Else return Null or None.
* Intuition of finding majority is equivalent to finding Boyer Moore’s majority vote algorithm. [1]
* This step ensures that there will maximum of comparisons at each level and we have divided our problem into two equal halves at each level, depth of our tree or total number of levels are .
* Hence, total time complexity for our algorithm become .

Time complexity can also be deduced from Master’s theorem i.e.

where is a monotonically increasing

= number of sub-problems,

= size of each problem,

= time of combine step

For our solution, as we are dividing our problem into two sub-problems at each step and as well. for algorithm is as we are making comparisons for combining solutions.

Hence, using master’s theorem our time complexity becomes **.**

Here is the working code for this problem:



**Question 4** - Implementation of Kruskal’s Algorithm for finding minimum spanning tree in a multi-graph.

Here is a list of data structures that I have used for my implementation:

* *Union-Find* - Best time complexity for Union-Find can be reduced to by using weighted quick-union with path compression and when done for *m* nodes, we get as *m\*.*
* *Dictionary* - I have used this for avoiding any computation for those edges which have been added to graph but exist multiple times with different weights. But as we have already sorted the edges in the order of increasing weight, we can be assured that if an edge with same *source* and *destination* is observed again, then it’s weight will be more than the the edge that was observed for the first time and hence we can just ignore it. Dictionary helps us in detecting collisions in *(1)* time.

**Time Complexity**:

Before we start calculating time complexities for both the methods, let’s define few variables.

- total number of edges

- total number nodes

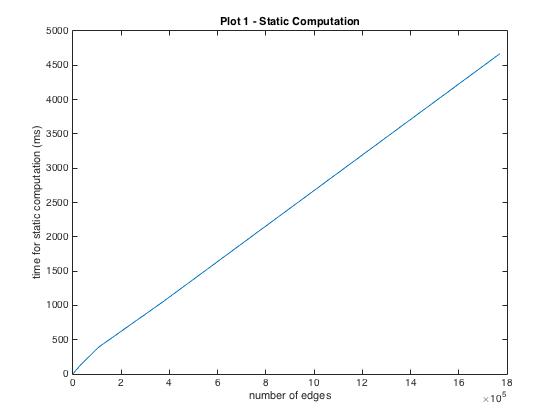
where 🡪 Eq. 1

* *computeMST( )* - First step is sorting the edges as per their weights, time complexity for this step is , but from Eq. 1 it can be easily deduced that , so time complexity for this becomes . Now, we just need to find time complexity for union find. Starting from an empty data structure, any sequence of m unions (total number of edges) and find operations on n objects (total number of nodes) takes . Moreover, for this case can be considered as constant hence for union-find becomes linear i.e. . So, for computeMST( ), .
* *recomputeMST( )* - Let’s list the algorithm for recomputing MST:
  + (u, v) is the new edge e\*, find shortest path, p, between u and v in T, MST.
  + find the edge in p that has the maximum weight .
  + If , then  the MST for , with
  + If not, then  is the MST of  (graph + edge)

Only step that can take some significant time in this algorithm is find shortest path, which takes linear time i.e. for a minimum spanning tree.

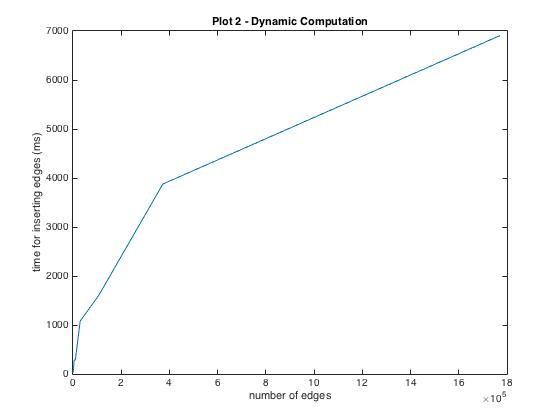
**Static Computation Plot**

Our forstatic computation is but we see a linear graph for our plot. As number of nodes are not too much for the given graph, can be considered a constant and does not really change the linearity and hence we get a straight line.



**Dynamic Computation Plot**

Our for dynamic computation is **n i.e. linear** and we should again expect a linear graph. But see a small curve in the beginning of the graph and that can be explained by the fact that for the first 4 graphs we are adding more number of edges in comparison to the initial number of edges. But as the initial number of edges get more than 1000, we see that our plot becomes linear.



**References**:

1. <https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_majority_vote_algorithm>
2. <https://www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf>
3. Code Reference for UnionFind: <https://www.ics.uci.edu/~eppstein/PADS/UnionFind.py>