Question 1

Consider the following Graph Dismantling Problem. Let $G = (V, E)$ be a connected undirected graph with $n \geq 4$ nodes $V = \{1, 2, \cdots, n\}$ and m edges. Let $e_1, e_2, \cdots, e_m$ be all the edges of G listed in some specific order. Suppose that we remove the edges from G one at a time, in this order. Initially the graph is connected, and at the end of this process the graph is disconnected. Therefore, there is an edge e_i, $1 \leq i \leq m$, such that just before removing e_i the graph has at least one connected component with more than $\lfloor n/4 \rfloor$ nodes, but after removing e_i every connected component of the graph has at most $\lfloor n/4 \rfloor$ nodes. Give an efficient algorithm that determines this edge e_i. Assume that G is given to the algorithm as a (plain) **linked list** of the edges appearing in the order $e_1, e_2, \cdots, e_m$.
The worst-case running time of your algorithm must be **asymptotically better than** $O(mn)$. More efficient solutions will get more marks. Hint: See An application of disjoint-set data structures in pages 562-564 of CLRS (Chapter 21).

Solution. We consider the general idea: We know that our goal is to get the removed edge that results in each connected component having at most $\lfloor \frac{n}{4} \rfloor$ nodes. What can be done is use this value as a "*threshold value*". Namely, the idea is that we will assume the linked list is doubly linked and contains a pointer to the tail, and so we can iterate from the *back of the list of edges*, which will allow us to "rebuild" edges to re-connect initially empty nodes. Thus we will reach a point where every connected component has nodes at the threshold value, and when we re-connect an edge that makes just one of these connected components exceed the threshold value by 1, then we know that this is the edge that would have been removed to result in every connected component having nodes at most $\lfloor \frac{n}{4} \rfloor$.

We will be using a **disjoint set forest** as the basis of our algorithm's data structure, with the following information:

- Each disjoint set tree will have a **size** attribute that stores the size of the tree. We assume it is stored as data separate, but attached to, the tree.
- We will be using the weighted-rank (smaller tree unions with the root of the larger tree) and path-compression heuristics to reduce time.

We also have access to a graph representation (see Piazza @288). Thus, we can describe our algorithm as follows:

1. Using the graph adjacency list representation, loop through the n vertices to create n singleton sets and set the value $threshold = \lfloor \frac{n}{4} \rfloor$. This represents the graph once all edges are removed. Set all tree $size = 1$.
2. Starting from the last edge in the linked list given, process this edge as follows:
    - For an edge (u, v), call $FIND - SET(u)$ and $FIND - SET(v)$ to get their respective disjoint set trees through their representatives, r, s respectively.

- call $UNION(r, s)$ to join these two trees together using their respective representatives. Sum their *size* attributes to get the size of the new tree immediately. If UNION is called on two vertices in the same set then don't do anything.
- Compare the *size* attribute of this newly-created tree to the $threshold = \lfloor \frac{n}{4} \rfloor$. If it *exceeds* the *threshold*, then return this edge because this would have been the edge removed to satisfy at least one connected component to be above the *threshold*

Using a fact from CLRS 21.4 we show that it has an upper bound of $O(n + m\alpha(n))$, where $\alpha(n)$ will be described.

1. Each MAKE-SET operation simply stores the vertex in a single-node disjoint tree instantiated, so each is simply $O(1)$, and a sequence n of these is then $O(n)$.

2. We are processing m edges in a list (of length m). The worst case situation is processing the entire list. Within each processing, we call FIND-SET twice and UNION at most once to combine these two trees. Also, size comparison takes $O(1)$, (because we save the tree representatives we can skip the FIND-SET steps in UNION, just linking the trees). For a total of m iterations, we have a sequence of $3m$ operations. From 21.4 the section shows that an arbitrary sequence of $3m$ FIND-SET and UNION operations will have running time $O(3m\alpha(n))$, where $\alpha(n)$ is a superlinear but extremely slow-growing function. While this is mathematically tight, in practical use by 21.4 we know that $\alpha(n) \leq 4 < 5$ so our worst case running time is linear with $O(3m * 5) = O(m)$

3. As a summary, the worst case running time is hence $O(n+m)$, which is asymptotically better than $O(mn)$

□

Question 2

A manufacturing company producing Product X maintains a display showing how many Product Xs they have manufactured so far. Each time a new Product X is manufactured, this display is updated. Interestingly, this company prefers base 3 notation; instead of using bits (0 and 1) as in base 2 notation, they use trits (0, 1 and 2) to represent this number. The total cost of updating the display is $\$(c + dm)$, where c is a fixed cost to open up the display, d is the cost of changing each display digit, and m is the number of digits that must be changed. For example, when the display is changed from 12201222 to 12202000 (because 12201222 + 1 = 12202000 in base 3) the actual cost to the company is $\$(c + 4d)$ because 4 digits must be changed. The manufacturing company wants to amortize the cost of maintaining the display over all of the Product Xs that are manufactured, charging the **same amount** to each. In other words, we are interested in the **amortized cost** of incrementing the display by one.

Let $A(n)$ be the amortized cost per display change operation when we perform a sequence of n successive display changes, starting from the number 0. For example, $A(3) = c + \frac{4}{3}d$. In this question, we want you to give an upper bound on $A(n)$. More precisely, consider the list:

$$c + \frac{4}{3}d, c + \frac{17}{12}d, c + \frac{3}{2}d, c + \frac{7}{4}d, c + 2d, c + \frac{5}{2}d$$

What is the smallest cost B in the above list such that $A(n) \leq B$ for all $n \geq 1$?
Prove the correctness of your answer in two ways: first analyse $A(n)$ using the aggregate method, and then analyse $A(n)$ using the accounting method with an appropriate charging scheme. Make sure you justify why your answer is the smallest such B from the above list.

1. **Aggregate method** Let $m(n)$ be the total number of digit changed in n increments starting from 0. Let k be the tritwidth of the display required to hold a decimal number n. There is such relationship between k and n

   $$k = \lfloor \log_3(n) \rfloor + 1$$

   The least significant trit change display on every increment, hence changes a total of n times; The digit left to the least significant bit change display every 3 increments, hence changes a total of $\lfloor \frac{n}{3} \rfloor$ times. Therefore, we have

   $$m(n) = \sum_{i=0}^{k} \left\lfloor \frac{n}{3^i} \right\rfloor = \sum_{i=0}^{\lfloor \log_3(n) \rfloor + 1} \left\lfloor \frac{n}{3^i} \right\rfloor \leq n \sum_{i=0}^{\infty} \frac{1}{3^i} = n \frac{1}{1 - 1/3} = \frac{3}{2}n$$

   for $n \geq 1$. Hence the total cost equates to

   $$c + dm(n) \leq c + \frac{3}{2}dn$$

   The amortized cost per display change is therefore

   $$A(n) \leq \frac{c + \frac{3}{2}dn}{n} = c' + \frac{3}{2}d$$

   where $c' \in \mathbb{R}$ for all $n \geq 1$

2. **Accounting Method** We assign an amortized charge of $\frac{3}{2}$ to display change from 0 to 1 or from 1 to 2. We use 1 credit to pay for the cost of changing the bits and store the $\frac{1}{2}$ as credit for later use. Note at any point, a digit of 1 holds $\frac{1}{2}$ credit and a digit of 2 holds 1 credit, which is used to pay for the reset from 2 to 0. In incrementing a base three number, there is exactly one digit which change from 0 to 1 or from 1 to 2, and possibly a series of reset from 2 to 0. Note that the reset from 2 to 0 is paid by the 1 credit stored in digit 2. Hence the amortized cost of incrementing display

by one is at most $\frac{3}{2}$, coming from the display change from either 0 to 1 or from 1 to 2. Note that for any display, a 0 stores 0 credit, a 1 stores $\frac{1}{2}$ credit, and a 2 stores 1 credit. Since the number of 0, 1, and 2 is nonnegative for all $n \geq 1$, hence the credit is nonnegative at any point in a sequence of n increments. Therefore, we proved that the amortized cost provides an upper bound on the actual cost. Note we scale the charge $\frac{3}{2}$ by a constant to represent the cost of changing each digit and c the cost to open up the display, we then have

$$A(n) \leq c + \frac{3}{2}d$$

for all $n \geq 1$

3. Now we prove $B = c + \frac{3}{2}d$ is the smallest B such that $A(n) \leq B$ for all $n \geq 1$ by proving $A(n) > B$ where $B = c + \frac{17}{12}d$ for some $n \geq 1$. Let $n = 9$, then $d = 4 \times 3 + 1 = 13$. Then the amortized cost per display change is then

$$c + \frac{13}{9}d > c + \frac{17}{12}d$$

So then $c + \frac{17}{12}d$ fails to bound $A(n)$. Since we proved that $B = c + \frac{3}{2}d$ provides upper bound on $A(n)$, therefore it is the smallest B in the list that satisfies $A(n) \leq B$ for all $n \geq 1$

Question 3

A multi-set is like a set where repeated elements matter. For example, $\{1, 8, 3\}$ and $\{3, 3, 8, 1, 8\}$ represent the same set but dierent multi-sets. Consider the abstract data type that consists of a multi-set of integers S and supports the following operations:

1. INSERT(S, x): insert integer x into multi-set S;
2. DIMINISH(S): delete the $\lceil |S|/2 \rceil$ largest integers in S. (For example, if $S = \{3, 3, 8, 1, 8\}$, then after DIMINISH(S) is performed, $S = \{3, 1\}$.)

A simple data structure for this ADT, and the corresponding algorithms for each operation, are as follows: The elements of S are stored in an unsorted linked list, and the size of S is stored in a variable n.

1. INSERT(S, x): Append x at the end of the list; increment n.
2. DIMINISH(S):

(a) $m \leftarrow \textsc{Med}(S)$ find the median of the elements in S

(b) loop over all elements in S: keep all those $< m$, delete all those $\geq m$

(c) add as many copies of m as required to have exactly $\lfloor n/2 \rfloor$ elements remaining

(d) $n \leftarrow \lfloor n/2 \rfloor$

The above uses an algorithm Med that returns the median of any list of integers. To simplify this question, we use a slightly non-standard definition of median: Med returns the $\lceil n/2 \rceil$-th largest integer in the list (including duplicates); for example $\textsc{Med}([3, 2, 1, 1, 3, 3, 3])$ returns 3. Also for this question, assume that Med performs at most $5n$ pairwise comparisons between integers during its execution on any list of length n.
Prove that when we execute an arbitrary sequence of Insert and Diminish operations starting from an empty set S, the amortized cost of an operation, in terms of the *number of pairwise comparisons between the elements of the set*, is constant. To prove this, use the accounting method to analyse the amortized complexity of the Insert and Diminish algorithms: (a) state clearly what charge you assign to each operation, (b) state and prove an explicit credit invariant, and (c) use your credit invariant to justify that the amortized cost of an operation is constant.

Proof. Since cost is defined by the number of pairwise comparisons between elements, then Insert yields 0 cost and in Diminish, Med costs at most $5n$ and delete costs exactly n, where n is the size of the linked list. Now we charge 12 for each Insert and 0 for each Diminish operation. The credit is stored in each node in the linked list during Insert and pays for Med and delete operation during Diminish. The former operations can be paid with 5 credit per node while the latter can be paid with 1 credit per node. We claim that

$$P(k) : \text{after any } k\text{-th step during a sequence of } m \text{ operations,}$$
$$\text{the credit } c \text{ stored in each node satisfies } c \geq 12$$

Now we prove credit invariant holds by induction.
Basis: The initial linked list is empty, hence $P(0)$ is trivially true
Inductive step: Assume that $P(k)$ is true, i.e. every node after k-th step has $c \geq 12$. Now we prove $P(k+1)$ holds. If Insert is the $k+1$ step in the sequence, then a new node with $c = 12$ is inserted into the linked list. All node in the linked list has $c \geq 12$ hence $P(k+1)$ holds. If Diminish is the $k+1$ step in the sequence, then we decrement the credit stored in each node by 6. The total credit paid amounts to $6n$ where n is length of the linked list, which is enough to pay for both Med and delete operation. At this moment, we have $c \geq 12 - 6 = 6$ for each node. During the deletion step and subsequent addition step (step 2 and 3), the length of linked list shrinks from n to $\lfloor n/2 \rfloor$, by transferring the credit of deleted node to ones that remains, the remaining nodes at least doubles in credit, hence $c \geq 6 * 2 = 12$. Hence $P(k+1)$ holds.
By simple induction, the credit invariant holds. Since the number of nodes in the linked list is nonnegative, the total credit is always nonnegative. Hence the total charge provides

an upper bound on total cost of a sequence of m operations. Therefore the worst case amortized sequence cost is $12m$, i.e. a sequence of m INSERT. The amortized cost of an operation is hence $\frac{12m}{m} = 12$, which is a constant □