Quiz 1 Solutions

Instructions:

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- Write your name below and circle your recitation at the bottom of this page.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages will be separated for grading.
- Make sure you write your first name on **every** page of the exam after the first page.
- **You are allowed one double-sided, letter-sized sheet with your own notes**. No calculators or programmable devices are permitted. No cell phones or other communication devices are permitted.
- Refrain from discussing this exam until Tue, 3/21.

Advice:

- You have 120 minutes to earn a maximum of 120 points. **Do not spend too much time on any single problem.** Read them all first, and attack them in the order that allows you to make the most progress.
- When writing an algorithm, a **clear** description in English will suffice. Using pseudo-code is not required.
- Do not waste time rederiving facts that we have studied. Simply state and cite them.

Question	Parts	Points
1: True or False	7	14
2: The Sin of Greed	2	13
3: The Last Pilot	3	21
4: LP Solving	2	15
5: An Avant-Garde Data Structure	3	30
6: Annihilating Triples	2	27
Total:		120

Name: ______________________________

Circle your recitation:	R01 Emanuele Ceccarelli **10 AM**	R02 Shraman Chaudhuri **11 AM**	R03 Mayuri Sridhar **12 PM**	R04 Kai Xiao **1 PM**	R05 Varun Mohan **2 PM**	R06 Isaac Grosof **3 PM**	R07 Devin Neal **11 AM**	R08 Lei "Jerry" Ding **12 PM**	R09 Sagar Indurkhya **1 PM**

Problem 1. [14 points] **True or False** (7 parts)
Please circle **T** or **F** for the following. *No justification is needed.*

(a) [2 points] **T F** For any graph G with n vertices and m edges, with distinct edge weights, there exist exactly m different MSTs of G.

Solution: False. If the edge weights are distinct, the MST is unique. See Lecture 6 notes.

(b) [2 points] **T F** Recall that the UNION–FIND data structure has amortized cost $O(\alpha(n))$ per operation, where $\alpha(n)$ is the very slowly growing inverse Ackerman function. Therefore, an adversary can never construct a sequence of n operations in which *one* of these operations will cost $\Omega(n)$.

Solution: False. Amortized analysis is a worst-case analysis over a sequence of operations. Some operations may indeed take $\Omega(n)$ time actual cost, as long as a sequence of n operations costs $O(n\alpha(n))$. See Recitation 2 notes.

(c) [2 points] **T F** If a flow network contains only integral capacities, then any max-flow must assign integral flows along each edge.

Solution: False. Consider a graph with source s, sink t and an intermediate node u. With two edges of unity capacity from s to u and an edge of unity capacity from u to t. It is easy to see the max-flow has value 1, which can be achieved by putting 0.5 flow on the two edges from s.

(d) [2 points] **T F** Given a list of integers of length n, we can output all of the largest $\lfloor \frac{n}{4} \rfloor$ elements in the list in linear time.

Solution: True. Use the median finding algorithm to find the $\frac{3n}{4}$-th ranked element, and split the list using this element as a pivot. Output everything in the bigger part.

(e) [2 points] **T F** In a weighted connected graph G, if s is a starting node in Prim's algorithm, then for any other vertex v, the path on the resulting MST from s to v is the shortest path.

Solution: False. Consider a graph G with vertices s, v, w and edge weights $e(s, v) = 3$, $e(s, w) = 2$, and $e(w, v) = 2$. Path on the MST from s to v is of cost 4, while the shortest path is 3. There are many examples possible.

(f) [2 points] **T F** We have a connected, undirected graph with weighted edges and at most one edge between every pair of vertices. Exactly one edge has weight 7, one edge has weight 9, and all other edges have greater weights. The edge with weight 9 is in *every* MST of the graph.

> **Solution:** True. Since the edge with weight 9 does not create a cycle with the edge with weight 7, it is always picked in Kruskal's algorithm.
>
> **Alternatively:** The edge with weight 9 is the second lightest edge in the graph (there are no other edges with its weight, and there is only one edge of lighter weight). If there were an MST that lacked the edge with weight 9, we could paste the weight 9 edge into the supposed MST, forming a cycle. The cycle must have at least three edges (because there is at most one edge between each pair of vertices). At least one of the three edges must have weight greater than 9, because the graph only has one edge with weight less than 9. Thus, when the heavier edge (or one of the heavier edges, if more than one) is removed, a spanning tree lighter than the supposed MST results. Thus, the supposed MST that lacks the weight 9 edge is not an MST, and all MSTs must contain the weight 9 edge.

(g) [2 points] **T F** If an LP is feasible then so is its dual LP.

> **Solution:** False. It could be that an LP is feasible but have no bounded optimum value. This would mean that the dual LP is infeasible.

Problem 2. [13 points] **The Sin of Greed** (2 parts)
You are the Benevolent Dictator of the Earth, and it is your 100th birthday. Because you are so benevolent, you have decided to gift control of the k most prominent cities in the world to n of your most trusted minions. Each of your minions has presented an unordered list ($\leq k$ in length) of cities that they would like to (exclusively) rule, and they will be completely satisfied if they are given only one of them. (They know that exhibiting too much ambition could be *very* unhealthy for them!)

(a) [5 points] You wish to satisfy the largest number of minions, and your first instinct, as always, is to use a greedy approach:

Iterate through the n minions, and for each minion, simply give them any city on their list that still remains unassigned. Prove that this algorithm does not necessarily maximize the number of satisfied minions (*i.e.*, construct an input on which this algorithm fails to produce an optimal assignment).

Solution:

Pre-Solution:

First of all, notice that this question is asking for an input on which the greedy algorithm fails to produce an optimal assignment: not a general case. Let us first think about what it means for an input to make the algorithm fail. It means that the greedy algorithm can produce an assignment that is worse than the optimal assignment. Notice that this means that the input needs to be picked so that there actually exists some optimal solution which is better than the performance of the greedy algorithm.

Solution: Let each city be denoted by an integer in $(1, \ldots, k)$. Given an input where $k = 2$, and $list_1 = 1, 2$ and $list_2 = 1$, our greedy algorithm might give city 1 to minion 1, and nothing to minion 2, whereas the optimal solution would give city 2 to minion 1 and city 1 to minion 2. The greedy algorithm satisfies one minion, and the optimal solution satisfies two. This shows that on this input the greedy algorithm is not optimal.

(b) [8 points] Now that we know the greedy method doesn't work, how would you find an assignment of cities to minions that always makes the maximum number of minions satisfied? Describe your algorithm and briefly analyze its running time. This running time should be polynomial in k and n.

Solution:

Pre-Solution:

From part (a), we know that a greedy approach doesn't immediately work. The first major choice we make when we're looking at this question is to decide whether we want to pursue the greedy approach and try to fix it or to look for a different technique altogether. It isn't immediately obvious which choice is better.

Let's take choice 1 - we want to change the greedy approach and develop an algorithm that maximizes the number of satisfied minions. To do this, we can carefully pick how to assign cities to unsatisfied minions. Maybe we could order the minions by how many cities they are willing to accept or assign cities based on how many minions are willing to accept that. However, it's possible to find counterexamples to both of these approaches. The key observation that should be made from exploring greedy algorithms is that we need to keep track of the degree of each minion (number of cities that the minion is willing to accept) and the degree of each city (how many minions want to rule it). This should lead students to see that this problem can be viewed graphically.

Once we try to represent this problem as a graph, it starts to look like a bipartite matching problem and the problem is reduced to properly constructing the network.

If we had instead chosen choice 2, then we would follow a similar train of thought, to find that the easiest way to model this problem is graphically.

Solution: Bipartite Matching. To find the optimal assignment from the flow network, we assign city i to minion j if the edge (i,j) has non-zero flow in our flow network after we finish running our flow algorithm. In the flow network we construct, there are 2 special vertices that represent the source and the sink. Then, there are n minions and k cities. Thus, there are $|V|$ = O($n + k$) vertices.

Since each minion is connected to at most k cities, there are O(nk) edges between the minions and the cities. There are exactly k edges from the source to the cities and exactly n edges from the minions to the sink. Thus, in total there are $|E|$ = O($nk+n+k$) = O(nk) edges.

If we used Edmonds-Karp, the algorithm would take O($|E|^2|V|$) = O($n^2k^2(n + k)$) time.

If we used Ford-Fulkerson, the algorithm would take O($f|E|$) time where f is the value of the max flow. The max flow is min(n,k), since the maximum number of satisfied minions is upper bounded by the number of minions or the number of cities. This would take O($\min(n, k) \cdot nk$) time.

Thus, we use Ford-Fulkerson to find the max flow.

Problem 3. [21 points] **The Last Pilot** (3 parts)
The Alliance Fleet sent a squadron of X-wing starfighters to disrupt the supply lines of the First Order's mining and refining operations in the artificial asteroid belt of Pressy's Tumble, which the First Order is mining from as fast as they possibly can.

You are the last remaining pilot, and owing to the work of countless Bothan spies, you have a complete map of their hyperspace traffic from the (single) mine, through all the refineries, ending at the First Order's headquarters. You have a hyperspace blocker that you can place along *one* of these hyperspace lanes to block it, and your job is to use it to permanently reduce the overall supply that reaches the First Order.

More formally, you have a hyperspace map represented as a graph M, starting at the mine, and ending at the First Order headquarters, with a set of lanes L (size of $|L| = m$), each with a maximum per day traffic load $\ell : L \to \mathbb{R}^+$, and the current yield from the mines or refineries entering each lane (averaged per day), $y : L \to \mathbb{R}^+ \cup \{0\}$.

(a) [5 points] At first, you are not even sure if it is possible to execute this mission. However, the flight command assures you that, given that the First Order is moving the materials at the maximum possible rate, there always exists a single lane whose blocking will permanently reduce the overall supply that can reach its destination. That is, no matter how First Order rearranges its transport traffic, the resulting yield arriving to it per day will drop. (You should assume that the initial overall supply of First Order is positive.)

Supply a proof to show that indeed such a *critical lane* always exists.

Solution:

Pre-Solution:

Perhaps the toughest part of this problem is figuring out what we're asked to prove. We immediately recognize that this is a graph problem, with lanes (edges) and refineries (vertices) where there's yield (flow) from the mine (source) to the First Order HQ (sink). After this, the first important observation is that "the First Order is moving the materials at the maximum possible rate" which means the network is at max-flow. We know that the max-flow is equal to the min-cut. Therefore, we need to just prove that there always exists an edge in the min-cut.

Solution:

If a flow network has positive max-flow, it must also have a min-cut of the same value. Lowering the capacities of any edge in a min-cut necessarily creates a smaller min-cut. Therefore, by max-flow, min-cut duality, lowers the max-flow.

(b) [8 points] Give a simple $O(m)$ time algorithm for finding such a critical lane whose existence you proved in part (a).

Solution:

Pre-Solution:

From lecture, we know that having the max-flow also gives us the min-cut. And we are already given that the network is at max-flow. Therefore the question here is how to find an edge from the min-cut defined by this max-flow. We start by recalling that according to Ford-Fulkerson, once a network is at max-flow, there are no more augmenting paths from s to t. Therefore, s and t will be disconnected in the residual network. We can simply then try to find one of these edges separating the two sides of the network. Sine the runtime required is $O(m)$, this means we need to use BFS/DFS to do this.

Solution:

Let M be the hyperspace map. Let the mine be the source s, and the headquarters be the sink, t. First compute the residual network for f on M. Since f is a max-flow, s and t must be disconnected by the edges of a minimum-cut. Run BFS to detect the set S of all edges reachable from s. Then iterate through each edge in M to find an edge that goes from S to outside of S. Return this edge since it must be in a minimum cut.

Constructing the residual network takes $O(m)$ time, running BFS takes $O(m)$ time and iterating through each edge checking both endpoints takes $O(m)$ time. So overall takes $O(m)$ time.

Common Misconception:

Most students correctly identify that at max-flow, in the residual network, all the edges of the min-cut will be saturated (residual capacity zero). However, this does not mean that all saturated edges are part of a min-cut. There could be edges in a network that are saturated, but not on a minimum cut. It is therefore not sufficient to simply find any saturated edge.

(c) [8 points] After you successfully (and quickly!) completed your mission, the flight command asks you to estimate how many more hyperspace blockers would be needed to completely shut down First Order's operation, i.e., to reduce the overall supply reaching the First Order to zero. Before you are able to think about your answer, your helpful astromech droid `R2-PIE2` makes the following claim: since the number of hyperspace jumps one has to make to reach First Order from the mine is at least k, you will never need more than $\frac{m}{k}$ hyperspace blockers to completely shut down First Order's operation.

Is `R2-PIE2` right? Justify your answer by either providing a proof of the claim or presenting a counterexample.

Warning: Beware of this problem, it might be (slightly) more challenging. Best to attempt it only once you finish solving the rest of the problems.
(Note: Julius Caesar ignored this warning...)

Solution: `R2-PIE2` is right. (Of course, he is! He got A+ in 6.046 he took at MIT, i.e., Multigalactic Institute of Technology.)

Pre-Solution:

This problem requires thinking more deeply about a flow network. The problem is asking whether we can disconnect the source and sink by removing only $\frac{m}{k}$ edges. Thus, if we think of each edge in the network as having capacity 1, we want to know whether there exists a cut of size $\leq \frac{m}{k}$. We can first reason about this intuitively. Suppose there were more than $\frac{m}{k}$ edges in every cut. How many cuts can there be? We know that each s-t path has length at least k. So maybe we can make k cuts? And maybe they could all be disjoint? If we could do this, it would clearly mean that we had more than m edges in the graph. We can then formalize this intuition.

Solution:

If the distance between the mine vertex s and Last Order vertex t is at least k, it means that if we consider k cuts S_i, for $1 \leq i \leq k$, where the cut S_i contains all the vertices at the distance less than i from s in M (viewing each edge as having length 1) then:

- Each S_i is an s-t cut;
- Each edge in M crosses *at most* one such cut S_i.

But, as there is k cuts and m edges, the above means that there has to exists cut S_{i^*} such that the number of edges crossing it is at most $\frac{m}{k}$. So, blocking all edges of this cut would separate s from t shutting down First Order operation.

Alternatively:

Consider a modified network where each edge had capacity 1. We know that in this case, the max-flow of the network is exactly the number of edge-disjoint paths from s to t, since each edge can carry at most 1 unit of flow. But since we know that each path has at least k edges, there can only be $\frac{m}{k}$ total edge-disjoint paths. Thus the max-flow is at most $\frac{m}{k}$. By max-flow min-cut duality, this implies that the min-cut is at most $\frac{m}{k}$.

Problem 4. [15 points] **LP Solving** (2 parts)
Consider the following LP:

$$\begin{aligned} \min \quad & 24x_2 + 7x_3 - x_4 \\ \text{s.t.} \quad & 2x_1 + 2x_2 - x_4 \geq 0 \\ & x_1 + 3x_2 - x_3 \leq 2 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

(a) [7 points] Find the dual of the LP given above.

Solution:

Pre-Solution:

This is a LP, so the first thing we want to do before finding the dual is to convert it to standard form. Standard form can either have a minimization objective with inequalities that all are $\geq$ signs, or it can have a maximization objective with inequalities that are all $\leq$ signs. We have a minimization problem, but we have both a $\geq$ sign and a $\leq$ sign, so we flip the $\leq$ sign by multiplying both sides by -1.

Once we have the problem in standard form, we can write down the dual. One way to make it easier to keep track of things is to write down the constraint vector b, the objective function vector c, and the constraint matrix A.

Note: Many students correctly noted that minimizing x is equivalent to maximizing $-x$, in that the same value for x will achieve both goals. However, these two have objective functions with different optimal values that are negatives of each other, so that must be accounted for.

Solution:

In standard form, the LP is

$$\begin{aligned} \min \quad & 24x_2 + 7x_3 - x_4 \\ \text{s.t.} \quad & 2x_1 + 2x_2 - x_4 \geq 0 \\ & -x_1 - 3x_2 + x_3 \geq -2 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

The dual of the above LP is given by:

$$\begin{aligned} \max \quad & -2y_2 \\ \text{s.t.} \quad & 2y_1 - y_2 \leq 0 \\ & 2y_1 - 3y_2 \leq 24 \\ & y_2 \leq 7 \\ & -y_1 \leq -1 \\ & y_1, y_2 \geq 0 \end{aligned}$$

(b) [8 points] What is the optimal objective value of the original LP? Justify your answer.

Solution:

Pre-Solution:

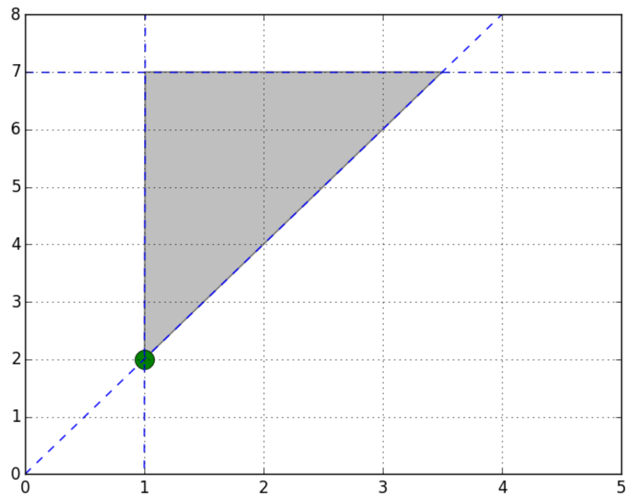
Strong duality tells us that we only need to solve the original LP or its dual because the optimal answer will be the same to both of them. The dual looks a lot easier to deal with because it only has 2 variables, so we will tackle that problem. For problems with 2 variables, it should be easy to visualize the problem as a graphical problem in 2D, so we'll try plotting the inequalities on a graph.

Solution: We'll solve the dual LP because it is easier to deal with.

We can solve this LP graphically. (To slightly simplify this, notice that we can drop the non-negativity constraint on y_1 as we also have $y_1 \geq 1$.) The inequalities we have to take into account are

- $2y_1 - 3y_2 \leq -24$
- $2y_1 - y_2 \leq 0$
- $0 \leq y_2 \leq 7$
- $1 \leq y_1$

These inequalities define the bounded region show below:



We see that we only have to check the 3 vertices of the bounded region. Since we are maximizing $-2y_2$, we're really just trying to find the vertex with the smallest y_2 value. The solution is $y_1^* = 1$ and $y_2^* = 2$. So, the optimal objective value is $\boxed{-4}$ and, by strong duality, this is exactly the objective value of the original (primal) LP.

Note: There are many alternative solutions to this problem involving algebraic manipulation of the inequalities. For example, noticing that $y_2 \geq 2y_1 \geq 2$ implies $-2y_2 \leq -4$.

Then, noticing that $(y_1, y_2) = (1, 2)$ satisfies all the constraints and also achieves this upper bound of -4 leads to the conclusion that -4 is the optimal value. Doing algebra on just the orignal LP can also be used to solve this problem.

Problem 5. [30 points] **An Avant-Garde Data Structure** (3 parts)
While studying for the 6.046 quiz, Ben Bittdiddle, in a moment of artistic inspiration, came up with a new, avant-garde variant of a queue data structure. This data structure supports the following operations on keys that are positive integers:

PUSH(x) Creates an element with the key x and places it onto the top of the data structure. This has unit cost.

POP Returns and removes the topmost element from the data structure with unit cost.

FUTUREPOP Reads the key k of the topmost element in the data structure with unit cost. Then removes k elements from the top of the data structure at an additional cost of k and reads ahead k more elements to return the value of that key (kth from the current top) at an additional cost of k (without removing these next k elements.) If an operation attempts to reach past the end of the data structure, it returns NULL.

(a) [6 points] Assume that at some point in time the data structure contain keys in the following order (top-to-bottom): $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Draw the state of this data structure after each of the following commands is executed (in sequence). Also, compute the cost of each operation.

(i) POP

Solution: The operation returns the value 1, and after the operation, the data structure contains $\{2, 3, 4, 5, 6, 7, 8, 9\}$. The cost is 1.

(ii) FUTUREPOP

Solution: The operation returns the value 5, and after the operation the data structure contains $\{4, 5, 6, 7, 8, 9\}$. The cost is 5.

(b) [10 points] Consider a sequence of n operations that started with the data structure being empty. What is the maximum number of times, asymptotically, that any particular given element could be accessed *before* being removed in that sequence of operations? Justify your answer. ("Accessed" means either read or removed.)

Solution: Consider an initial PUSH operation that puts the key 314 onto the data structure. Now consider a large number of iterations of the sequence of two operations: PUSH(1) and FUTUREPOP. Each FUTUREPOP operation reads the topmost of the two elements whose key is 1, then removes the topmost element, and finally reads and returns the key for the remaining element with key 314 (the original element PUSHed onto the data structure). This original element is read (accessed) once every time the sequence of two operations is executed, and it is never removed. (One could add a final POP to actually remove it). Thus, this sequence pattern asymptotically accesses the original element $n/2$ times [or $\Theta(n)$]for a sequence of n operations.

Note: Clearly, the number of accesses is upper bounded by $\Theta(n)$ simply because each operation can only access an element at most once. Students who provided an example where an element is accessed $\Theta(n)$ times received full credit. Many students thought that the maximum number of accesses was upper bounded by $\Theta(\log n)$ because they were under the impression that all the FUTUREPOP operations had to happen consecutively after all the PUSH operations (they made arguments that the number of elements after the target element has to halve at every FUTUREPOP operation in order for there to be an access).

(c) [14 points] Perform an analysis to show what the asymptotic amortized cost per operation is for this data structure.

Solution:

Pre-Solution:

Each element can be PUSHed once (at a cost of 1), POPped once (at a cost of 1), and accessed an arbitrary number of times through being read in a FUTUREPOP operation. In amortized analysis, we account for costs, particularly of expensive operations, by paying for them in smaller increments during the process that sets up the situation that makes it possible for them to be incurred. Here, it is worth noting that every time an element is read in a FUTUREPOP operation, another element is removed from the data structure. Thus, in an accounting sense, we could double charge the removal part of the FUTUREPOP to pay for the reading part. This is an advantage, because elements can be removed only once but can be read an arbitrary number of times. Even though an element may be accessed an unbounded number of times through FUTUREPOPs, it would require an unbounded number of elements to be removed to pay for it. Because an element may be removed either through a POP or SUPERPOP operation, and the cost of a FUTURE POP operation could be quite large, it is easier to imagine paying for the removal when an element is added to the data structure, by charging three for each PUSH. As we have outlined the accounting method here, in the solution below we will use the potential method.

Solution:

Define a potential equal to twice the number of elements in the data structure. This is a valid potential in that it starts at zero when the data structure is empty and can never be negative.

$\hat{c}_{\text{push}} = c_{\text{push}} + \Delta\Phi = 1 + 2 = 3$
$\hat{c}_{\text{pop}} = c_{\text{pop}} + \Delta\Phi = 1 - 2 = -1$
$\hat{c}_{\text{futurepop}} = c_{\text{futurepop}} + \Delta\Phi = 2k + 1 - 2k = 1$

The amortized cost of any operation is at most 3, so the asymptotic amortized cost is $\Theta(1)$.

Problem 6. [27 points] **Annihilating Triples** (2 parts)
You got an internship at CERN and, thanks to your 6.046 expertise, you were tasked with the important job of designing an algorithm for detecting the existence of *annihilating triples* in the stream of particles generated in the experiment. Specifically, the problem that you want to solve is: given a list P of n integer number that represents the "charges" of each particle, check if there exists an *annihilating triple* in it, that is a triple of three distinct elements from P whose sum is equal to 0. Each charge is an integer and *non-zero* number between $-314 \cdot n$ and $314 \cdot n$, and all of them have *distinct* values.

So, for example, if the list is $P = \{1, -4, -5, 2, 3\}$ then it contains annihilating triples, e.g., $\{1, -4, 3\}$ and $\{-5, 2, 3\}$. On the other hand, if $P = \{1, -4, 5, 2, 4\}$ then there are no annihilating triples. In particular, the triple $\{-4, 2, 2\}$ is not annihilating as it uses the same element twice.

(a) [15 points] Devise a fast algorithm for this problem and briefly analyze its running time.

Solution:
Pre-Solution:
There are two key pieces of information that can help orient us in how to solve this problem.

1. The problem asks us to compute sums over a *bounded* range, which suggests that we may be able to use the FFT.
2. Since $0 \notin P$, an annihilating triple must consist of either two positive numbers and one negative number or one positive number and two negative numbers.

Solution:
Define $d = 314 \cdot n$.

1. Construct the degree $2d$ polynomial $A(x) = \sum_{i=0}^{2d} a_i x^i$, where a_i is 1 if $i - d \in P$ and 0 otherwise. This operation takes $O(n)$ time. Note that we must add d to the value members of P so that $A(x)$ is a linear sum of non-negative powers of x.
2. Compute the polynomial $B(x) = A^3(x)$, which can have degree $6 \cdot d$ now, by using the FFT to multiply the polynomials. This operation takes $O(n \log n)$ time. Note that the coefficient b_{3d} is the total number of triples of numbers from P that sum up to zero.
3. Some of the triples counted in b_{3d} might use the same number twice, so we need to discount these triples. To compute the number of such false annihilating triples:
   (a) Compute the degree $4d$ polynomial $C(x) = \sum_{i=0}^{4d} c_i x^i$, with $c_{2(i+d)} = 1$ if $i \in P$. (All the other c_is are zero.) This operation takes $O(n)$ time.
   (b) Use FFT to compute the polynomial $F(x) = A(x)C(x)$ of degree $6d$. The coefficient f_{3d} is the number of triples that sum to 0 and use one number exactly twice. This operation takes $O(n \log n)$ time.

4. Return *TRUE* if $b_{3d} - 3f_{3d}$ is non-zero; otherwise return *FALSE*.

The overall runtime is $O(n \log n)$ time.

(b) [12 points] Show how to extend your algorithm from part (a) to make it actually output an annihilating triple, if one exists. This extension should not increase the asymptotic running time of your algorithm.

Solution:

From part (a), we can determine if an annihilating triple exists. If one does exist, either it consists of two positive numbers and one negative number, or one positive number and two negative numbers. W.L.O.G. suppose that the annihilating triple consists of two positive numbers and one negative numbers.

1. Let $P_+ = \{i \in P | i > 0\}$ and $P_- = \{i \in P | i < 0\}$. Let these sets be represented as an $O(n)$ length bit-vectors allowing us to check for membership in $O(1)$ time.
2. Let $A_+(x) = \sum_{i \in P_+} x^i$. We can compute the polynomial $B_+(x) = A_+^2(x) - A_+(x^2)$ in $O(n \log n)$ time by using the FFT for polynomial multiplication. Note that the i-th coefficient of $B_+(x)$ is non-zero if and only if there exists two distinct members of P_+ that sum to i.
3. Scan through the coefficients of $B_+(x)$ - if the i-th coefficient of B_+ is non-zero and $-i \in P_-$, then there exists an annilhilating triple $(-i, j, i - y_a)$ with $j, (i-j) \in P_+$.
4. To find j, scan through P_+ and checking whether $(i-j) \in P_+$ and $j \neq i - j$. This can be done in $O(n)$ time.

The overall runtime is $O(n \log n)$ time.