Midterm 1

Name: Targaryen

SID: 0123456789

Name and SID of student to your left: Lannister

Name and SID of student to your right: Stark

Exam Room:
☐ 2060 VLSB ☐ 100 GPB ☐ 145 Dwinelle ☐ 120 Latimer ☐ 405 Soda (6-10 pm) ☐ 10 Evans ☐ 2050 VLSB ☐ Hearst Field Annex A1 ☐ 306 Soda (6-10 pm)

Please color the checkbox completely. Do not just tick or cross the box.

Rules and Guidelines

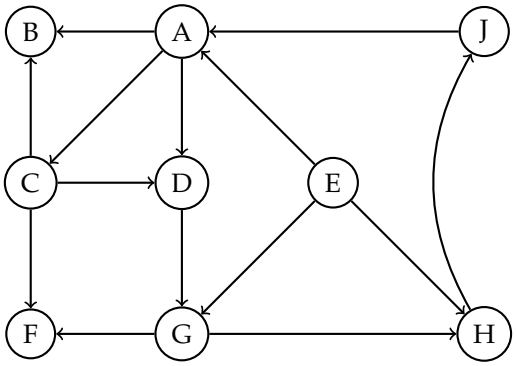
- **The exam is out of 110 points and will last 110 minutes.**
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- Write your student ID number in the indicated area on each page.
- Be precise and concise. **Write in the solution box provided.** You may use the blank page on the back for scratch work, but it will not be graded. Box numerical final answers.
- The problems may **not** necessarily follow the order of increasing difficulty. ***Avoid getting stuck on a problem.***
- Any algorithm covered in lecture can be used as a blackbox. Algorithms from homework need to be accompanied by a proof or justification as specified in the problem.
- Throughout this exam (both in the questions and in your answers), we will use ω_n to denote the first n^{th} root of unity, i.e., $\omega_n = e^{2\pi i/n}$.
- You may assume that comparison of integers or real numbers, and addition, subtraction, multiplication and division of integers or real or complex numbers, require $O(1)$ time.
- Good luck!

Discussion Section

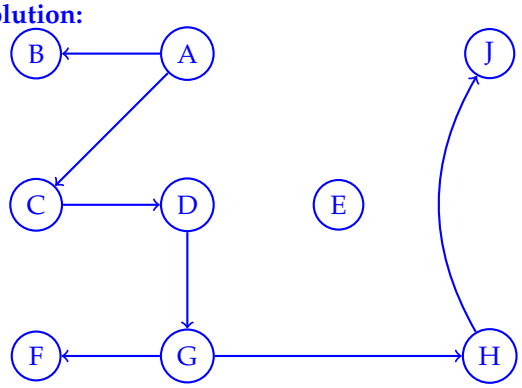
Which of these do you consider to be your primary discussion section(s)? Feel free to choose multiple, or to select the last option if you do not attend a section. **Please color the checkbox completely. Do not just tick or cross the boxes.**

- ☐ Antares, Tuesday 5 - 6 pm, Mulford 240
- ☐ Kush, Tuesday 5 - 6 pm, Wheeler 224
- ☐ Arpita, Wednesday 9 - 10 am, Evans 3
- ☐ Dee, Wednesday 9 - 10 am, Wheeler 200
- ☐ Gillian, Wednesday 9 - 10 am, Wheeler 220
- ☐ Jiazheng, Wednesday 11 - 12 am, Cory 241
- ☐ Sean, Wednesday 11 - 12 am, Wurster 101
- ☐ Tarun, Wednesday 12 - 1 pm, Soda 310
- ☐ Jerry, Wednesday 12 - 1 pm, Wurster 101
- ☐ Jierui, Wednesday 12 - 1 pm, Etcheverry 3113
- ☐ Max, Wednesday 12 - 1 pm, Etcheverry 3115
- ☐ James, Wednesday 2 - 4 pm, Dwinelle 79
- ☐ David, Wednesday 2 - 3 pm, Barrows 140
- ☐ Vinay, Wednesday 2 - 3 pm, Wheeler 120
- ☐ Julia, Wednesday 3 - 4 pm, Wheeler 24
- ☐ Nate , Wednesday 3 - 4 pm, Evans 9
- ☐ Vishnu, Wednesday 3 - 4 pm, Moffitt 106
- ☐ Ajay, Wednesday 4 - 5 pm, Hearst Mining 310
- ☐ Zheng, Wednesday 5 - 6 pm, Wheeler 200
- ☐ Neha, Thursday 11 - 12 am, Barrows 140
- ☐ Fotis, Thursday 12 - 1 pm, Dwinelle 259
- ☐ Yeshwanth, Thursday 1 - 2 pm, Soda 310
- ☐ Matthew, Thursday 1 - 2 pm, Dwinelle 283
- ☐ Don't attend Section.

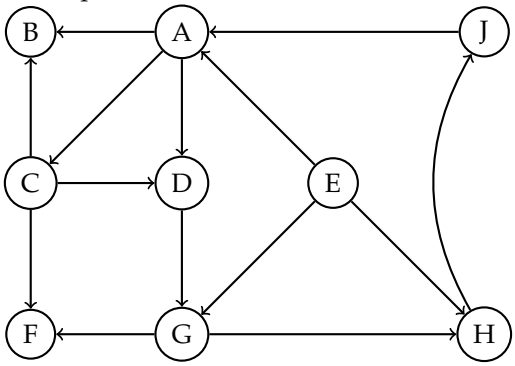
1. (**6 points**) Execute a DFS traversal on the graph shown below starting at node A, breaking ties alphabetically. **Draw the DFS Tree/Forest in the box below.**



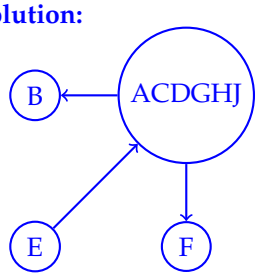
Solution:



2. (**4 points**) Draw the DAG of strongly connected components for the above graph (the directed graph is reproduced here for convenience)



Solution:



3. (**12 points**) For each of these pairs of functions $(f(n), g(n))$, identify whether $f(n) = O(g(n))$ or $g(n) = \Omega(f(n))$ or $f(n) = \Theta(g(n))$. Pick the most specific of the three options in each case, and pick at most one option in each case.

$f(n)$	$g(n)$	$f(n) = O(g(n))$	$f(n) = \Omega(g(n))$	$f(n) = \Theta(g(n))$
n^2	$(n + \log n)(n + 5)$	○	○	●
3^n	$2^n \cdot n^4$	○	●	○
$2^{\sqrt{n}}$	$n^{\log n}$	○	●	○
$n \log n$	$n^{1.01}$	●	○	○

4. (**12 points**) Write down the solutions to the following recurrence relations (only the final answer needed). Assume $T(0) = T(1) = 1$ and write down the most specific bound that you can derive.

   (a) $T(n) = 8T(n/2) + O(n^2)$

   Solution: $O(n^3)$. $log_2(8) = 3 > 2$. Hence we get $T(n) = O(n^3)$ using Master theorem.

   (b) $T(n) = 4 \cdot T(n-2)$

   Solution: $\Theta(2^n)$. It takes $n/2$ steps to go from n to 0 or 1 (depending on whether n is odd or even). So we get $T(n) = \Theta(4^{n/2}) = \Theta(2^n)$

   (c) $T(n) = T(n - n^{1/3}) + 1$

   Solution: $O(n^{2/3})$. If we recursively keep subtracting $n^{1/3}$ from n, it takes $10n^{2/3}$ steps to get to $n/2$. So we can construct another recurrence as follows $T(n) = T(n/2) + O(n^{2/3})$. This gives $T(n) = O(n^{2/3})$

5. (**8 points**) Given a directed graph $G = (V, E)$, for a vertex v define $numVisited[v]$ as,

$numVisited[v]$ = Number of nodes including v that are visited if we start exploring the graph at node v. Equivalently, number of nodes visited by $explore(v)$ if we start DFS from vertex v with $visited[w]$ initialized to $False$ for all vertices w

In a directed graph G, the values of $numVisited[v]$ are as follows:

Vertex	A	B	C	D	E	F	G	H	I	J	K
numVisited	5	5	6	5	6	2	6	6	2	10	10

(a) Draw all the sink SCCs

Solution: $\{F, I\}$

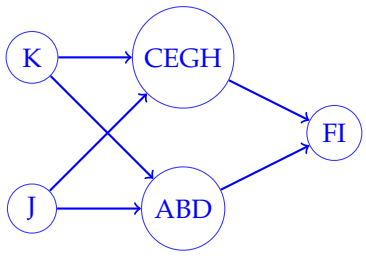
(b) Draw all the source SCCs

Solution: $\{J\}, \{K\}$

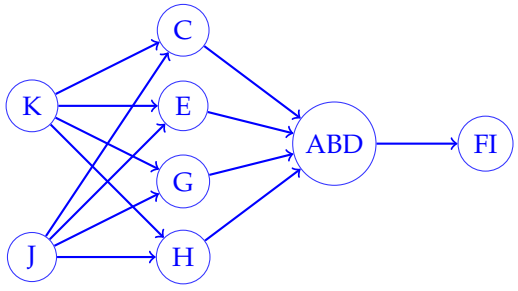
(c) Draw the DAG of SCCs

Solution: There were a few possible:

i.



ii.



6. (**4 points**) Vertex v appears immediately after u in a linearization of a DAG, but then there is no edge $u \to v$. Then what can you say about the following statements?

   (a) There is a path from u to v.

   ○ Always True ○ Sometimes True ● Never True

   Solution: There cannot be. If the path was only length one, that would imply an edge $u \to v$. Otherwise there's a path $u \to w_1 \to \ldots \to w_n \to v$. But then u and v cannot come consecutively, one of the w_i must come before v in the linearization.

   (b) There is a path from v to u.

   ○ Always True ○ Sometimes True ● Never True

   Solution: This is impossible. If there was, then v must appear before u in any valid linearization.

7. (**6 points**) In each of these cases, write the tightest bound possible.

   (a) In an undirected graph on n vertices, deleting an edge can increase the number of connected components by at most

   $\boxed{1}$

   Solution: At most the edge can disconnect the two connected components on either side, which increases the number of components by one.

   (b) In a directed acyclic graph on n vertices, deleting an edge can increase the number of strongly connected components by at most

   $\boxed{0}$

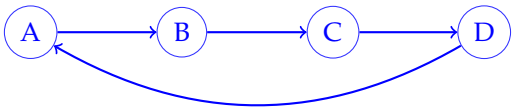
   Solution: In a DAG, an SCC with more than one vertex implies the existence of a cycle. So all SCCs must be single vertices, and deleting an edge would not change the number of SCCs.

   (c) In a directed graph on n vertices, deleting an edge can increase the number of strongly connected components by at most

   $\boxed{n-1}$

   Solution:

   Consider the strongly connected path on n vertices:

   Removing any edge turns this from 1 SCCs into n SCCs. This is the maximum possible increase, the number of SCCs is lower bounded by 1 and upper bounded by n.

8. (**16 points**)

For each of the statements below indicate whether they are TRUE or FALSE:

(a) In a particular execution of DFS on a directed graph G, it so happened that for every vertex v, the corresponding call to $explore(v)$ did not visit any new node but terminated immediately. Then G must have n strongly connected components.

Solution: True. Clarification: $G = (V, E), n = |V|$. On any graph with a strongly connected component, the first $explore(v)$ that explores one vertex in the SCC will explore all of the vertices in the SCC. As the number of vertices each explore does is at most 1, the number of SCCs must be at least n, which means that they are all single vertices.

(b) All the cycles of a graph $G = (V, E)$ can be listed in time $O(|V| \cdot (|V| + |E|))$ by executing the DFS traversal $|V|$ times, once from each vertex.

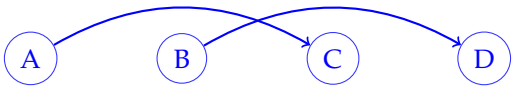
Solution: False. In a nutshell, the graph may not even have a polynomial number of cycles to begin with. Consider the fully connected graph on n vertices. Every nonempty subset of the vertices adds at least one unique cycle. There are 2^{n-1} nonempty subsets of vertices, so the number of cycles is at least exponential. Then no polynomial-time algorithm can lists all the cycles for this graph.

(c) Recall the recursive $Select(S, k)$ algorithm for selecting the k^{th} smallest element in a list of numbers S. This algorithm has asymptotically the same expected runtime for finding the smallest element ($k = 1$) or the median ($k = n/2$), but asymptotically different worst case runtimes.

Solution: False. It is true that the expected time for recursive `Select`(S, k) doesn't depend on k. However, you can come up with $\mathcal{O}(n)$ worst case examples for both $k = 1$ and $k = n/2$. For the smallest element, the worst case is to always pick the largest element as the pivot. For the median, the worst case is to pick the largest or smallest element as the pivot. In each of these only one element is thrown per iteration of the algorithm, resulting in $\mathcal{O}(n)$ worst-case time.

(d) Let $(v_1, \ldots, v_n)$ denote a linearized ordering of the vertices of a Directed Acyclic Graph (DAG) G. If we add an edge $v_n \to v_1$ then the resulting graph can still be a DAG.

Solution: True. Consider the following DAG:



As the picture suggests, $[A, B, C, D]$ is a valid linearization. If we add an edge $D \to A$, the graph is still a DAG.

In general, we can do this iff there does not exist a path $v_1 \to v_{i_2} \to \ldots to v_{i_k} \to v_n$, because adding an edge $v_n \to v_1$ would create a cycle. If there is no such path, then adding the edge wouldn't create a cycle.

(e) 8^{th} roots of unity are a subset of 16^{th} roots of unity.

Solution: True. The 8th roots of unity are $e^{2\pi i 1/8}, \ldots e^{2\pi i 8/8}$, and the 16th roots of unity are $e^{2\pi i 1/16}, \ldots e^{2\pi i 16/16}$. But $e^{2\pi i (2k)/16} = e^{2\pi i k/8}$, so we listed all the 8th roots of unity when we listed all the 16th roots of unity.

(f) Fix n to be a power of 2. FFT can be used to evaluate a degree n polynomial at the $2n^{th}$ roots of unity in time $O(n \log n)$.

Solution: True. We are evaluating on more points than we need, but FFT will still give us the correct evaluations on $2n$ roots of unity.

(g) Fix n to be a power of 2. FFT can be used to evaluate a degree $2n$ polynomial at the n^{th} roots of unity in time $O(n \log n)$.

Solution: True. It might not be helpful because the polynomial is too high-degree to take the inverse FFT from only n points, but the FFT still gives us the evaluation on n roots of unity.

(h) Adding a new edge to the graph always increases the number of DecreaseKey operations in Dijkstra's algorithm.

Solution: False. The `DecreaseKey` operation only happens when we traverse an edge and visit a vertex that is on the queue, but our just-discovered distance to the vertex is smaller than the vertex's current distance. If we traverse an edge that is really high capacity, the vertex will no longer be on the queue and no `DecreaseKey` will be called.

(i) Strassen's matrix multiplication is a sick algorithm.

Solution: True. If you disagree, I dare you to come up with better

9. (**5 points**) Let n be a power of 2 and let $\{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$ denote the n^{th} roots of unity. Suppose

$$E[0, \ldots, n/2-1] \leftarrow FFT(a_0, a_2, \ldots, a_n)$$

and

$$O[0, \ldots, n/2-1] \leftarrow FFT(a_1, a_3, \ldots, a_{n-1})$$

then write pseudocode to compute $FFT(a_0, \ldots, a_n)$ given arrays E and O.

Solution:
`for` $j \leftarrow 0, \ldots, n/2-1$:
  `A`$[j] = E[j] + \omega^j O[j]$
  `A`$[j+n/2] = E[j] - \omega^j O[j]$
$FFT(a_0, \ldots a_{n-1}) = A$

10. **Testing efficiently** (**8 points**)

At most k people among a population of n people in a city have been afflicted by a deadly virus. The city council needs to identify the k people infected so as to treat them.

The infection can be detected by testing the patient's blood for the virus. Unfortunately, each test is very expensive and the city council would like to minimize the number of tests carried out.

One can mix the blood samples from a subset of people, and test at one shot if at least one in the subset is infected. Specifically, for any subset S of people, $test(S : subset)$ carries out one blood test and returns YES if at least one in S is infected and NO otherwise.

Describe an algorithm that uses $O(k \log n)$ tests to algorithm to find all infected people in the city. (try to use 6 sentences or less)

Solution:
We considered several different solutions:

(a) *(Full credit)* Write a recursive algorithm that takes a set of people of size n. It checks in one operation if anyone in the entire set has the virus. If not, then return that no one is infected. Otherwise, split the people into two halves and recurse, and merge the set of infected found in each half.

If we analyze the recursion tree produced, at the first level we have one call, at the second level we have two calls, the third we have four calls, and so on. At each level, each of the k infected people can appear in at most k of the subsets at that level, which means that at most $\mathcal{O}(k)$ tests are done at each level. The number of levels is $\log_2 n$ (we divide by two each time), so this gives us $\mathcal{O}(k \log n)$ tests in total.

This analysis does not actually produce a tight bound. For the first $\log k$ levels we will do strictly less than k tests each, in fact we will do $1 + 2 + 4 + \ldots + k$ tests, which is in $\mathcal{O}(k)$ tests. For the remaining $\log n - \log k$ levels we will do at most k work, which gives a total bound of $\mathcal{O}(k + k(\log n - \log k))$. If $k \ll n$, then this is $\mathcal{O}(k \log(n/k))$ tests.

(b) *(Full credit)* To find a single infected person, you can essentially do binary search. Split the n people into two groups, and test each group There must be at least one of the groups that the blood test detects the virus in. Recurse on any group that had an infected person to find them. This procedure takes $\mathcal{O}(\log n)$ time.

Repeat the procedure k times, sending home the infected person you find each time. This takes $\mathcal{O}(k \log n)$ time total.

(c) *(Full credit)* Split the people into $2k$ groups and test all. At most k must have a person with a virus in them, ignore the other k groups. Recurse until $2k > n$, at which point just scan the remaining people. This is $\mathcal{O}(k \log n)$ time as well.

11. **Moving Battalions**

You are given an undirected unweighted graph $G = (V, E)$ that represents a network of cities. There are two army battalions that are positioned at two designated vertices $s_1, s_2 \in V$.

The battalions move during the day and rest at night. Each day a battalion can move to a neighboring city, or stay where it is. The two battalions would like to reach their target destinations $t_1, t_2 \in V$ respectively.

No city can host both the battalions for a night, so the two battalions can never spend the same night in the same city. Construct a graph H such that we can recover the optimal plan by running BFS on H.

The number of vertices and edges in H should be polynomial in number of vertices in G, i.e., $|V(G)|^c$ for some fixed constant c.

(a) (**7 points**) Vertices of H:

Solution:
The vertices of H correspond to all the possible "positions" of the batallions. There are two possible ways to encode this in a graph:

i. The vertices of H are the same as those of the graph product of G with itself , i.e., $V(H) = V(G) \times V(G) = \{(u_1, u_2) | u_1 \in V(G), u_2 \in V(G)\}$.

ii. $V(H) = \{(u_1, u_2) | u_1 \in V(G), u_2 \in V(G), u_1 \neq u_2\}$. This is to encode that two batallions are not in the same city at any given time.

In both cases, $|V(H)| = \mathcal{O}(|V(G)|^2)$

(b) (**8 points**) Edges of H:

Solution:
There are two possible solutions again corresponding to the vertex set of H defined in part (a)

i. (u_1, u_2) is adjacent to (v_1, v_2) iff either
- $u_1 = v_1$ and u_2 is adjacent to v_2 in G and $v_1 \neq v_2$ **OR**
- $u_2 = v_2$ and u_1 is adjacent to v_1 in G and $v_1 \neq v_2$ **OR**
- u_1 is adjacent to v_1 in G and u_2 is adjacent to v_2 in G and $v_1 \neq v_2$

The $v_1 \neq v_2$ encodes that two batallions are not in the same city at any given time.

ii. (u_1, u_2) is adjacent to (v_1, v_2) iff either
- $u_1 = v_1$ and u_2 is adjacent to v_2 in G **OR**
- $u_2 = v_2$ and u_1 is adjacent to v_1 in G **OR**
- u_1 is adjacent to v_1 in G and u_2 is adjacent to v_2 in G

In both cases, $|E(H)| = \mathcal{O}(|E(G)|^2) = \mathcal{O}(|V(G)|^4)$

12. **Adding Coins**

In the nation of Transformia, there are t different kinds of coins with integer denominations $c_1, \ldots, c_t$ in the range $[1, \ldots, n]$ in circulation.

We will now devise an algorithm that takes as input the values $c_1, \ldots, c_t$ in the range $[1, \ldots n]$ and an integer k, and lists all possible values that can be expressed as a sum of exactly k coins. Here the k coins expressing a value need not be distinct.

For example, if $\{c_1, c_2\} = \{1, 7\}$ and $k = 2$, then the set of values that can be expressed are $\{1+1, 1+7, 7+7\} = \{2, 8, 14\}$.

(a) (**6 points**) Find a polynomial which can be used to obtain all possible values that can be expressed as a sum of exactly k coins.

Solution: Exponentiation converts multiplication to addition. So define,

$$p(x) = \sum x^{c_1} + \ldots + x^{c_t} \tag{1}$$

Notice that $p(x)^k$ contains a sum of terms, where each term has the form $x^{c_{i_1}} \ldots x^{c_{i_k}} = x^{c_{i_1} + \ldots + c_{i_k}}$ where $i_1, \ldots, i_k$ need not be distinct. Therefore, we just need to check all the monomials x^n which appear in $p(x)^k$ since that means that they have a coefficient greater than 0 and hence n can be represented as the sum of exactly k coins.

(b) (**2 points**) We will now design an algorithm to compute the polynomial using FFT. What is the order of FFT we would use for this purpose?

Solution: $O(nk)$. Notice that while $p(x)$ has degree n, we are multiplying $p(x)$ k times and will finally get a polynomial of degree nk.

(c) (**5 points**) Describe an algorithm that computes the polynomial as efficiently as you can. You can use *FFT, InverseFFT and Polynomial multiplication via FFT* in a blackbox fashion in your description. (Try to use 6 sentences or less)

Solution:
Solution 1 (full credit): Given that $q(x) = p(x)^k$ is a polynomial of degree nk, it suffices to interpolate at $nk+1$ points via FFT and then use inverse FFT to get the coefficients of $q(x)$, provided we can compute $p(x)^k$ at any given point efficiently. Thus, consider the $(nk+1)^{th}$ roots of unity $\{1, \omega, \omega^2, \ldots, \omega^{nk}\}$. Compute $p(x)$ at these $nk+1$ points and exponentiate the output to the order k, i.e., compute $p(\omega^j)^k$ as this is just powering a real number to power of k which takes $O(logk)$ time and we are doing this for $nk+1$ points. Finally, we run InverseFFT to get the coefficients of $q(x)$
Solution 2 (full credit): We solve this problem using divide and conquer. Consider a procedure PolyPower(p,k) which takes a polynomial p and a natural number k and returns $p(x)^k$. Recursively compute $r(x) = PolyPower(p, k/2)$. Finally call *Polynomial multiplication via FFT*(r, r). The degree of $r(x)$ is $nk/2$.
Solution 3 (partial credit): Instead of recursively calling PolyPower in the above solution, we can just compute $p(x)^2, p(x)^2 \times p(x)$ and so on till $p(x)^{k-1} \times p(x)$ via *Polynomial multiplication via FFT*. The degree of the polynomials increase as $n, 2n, 3n \ldots, nk$.

(d) (**1 point**) What is the runtime of your algorithm?

Solution:
Solution 1 (full credit): FFT operation = $O(nk\log(nk))$, powering nk real number to power k takes $O(nk\log(k))$, inverse FFT takes $O(nk\log(nk))$ for a total runtime of $O(nk\log(nk) + nk\log(k)) = O(nk\log(nk))$.
Solution 2 (full credit): There are $\log(k)$ levels of recursion and in each step we are spending $O(nk\log(nk))$ work as we are doing polynomial multiplication of degree $O(nk)$ so we get a runtime of $O(nk\log(nk)\log(k))$. **Solution 3 (partial credit):** There are k multiplication operations and again doing polynomial multiplication of degree $O(nk)$ to get a runtime of $O(nk^2\log(nk))$.

13. **Testing efficiently again**(Extra Credit)

The city council from Question 10 is now dealing with a new virus. The good news is that the new virus has only infected exactly two citizens in the population of n people. The bad news is that the test for the new virus fails if we mix the blood of both the infected parties. Formally, the $test(S : subset)$ for a subset of citizens S returns YES if S contains exactly one infected citizen, but returns NO if S contains zero or two infected citizens.

Describe an algorithm to identify the two infected citizens using at most $O(\log n)$ tests. (try to use 6 sentences or less)

Solution: There are a few possible solutions:

(a) We test the following subsets for $k = 1, ..., n$

$$S_k = \{a_i : \text{the } k\text{th bit of the binary string expansion is } 1\}$$

We are guaranteed that at least one of them returns YES. If all return NO then the two infected people appear together in every subset where one of them appears. But this means that the binary string that specifies their location is the same, so this is impossible. Once we find a subset S_k that returns YES, we can use binary search by testing half of the array at a time until we converge on a single answer and then test the citizens not in S_k to find the other infected person. It takes $\log n$ tests to test all the binary string sets and $2 \log n$ tests to perform binary search on each of the halves. Thus the amount of tests is $\Theta(\log n)$.

(b) We divide the set in half and test both halves. If both return YES then we can use binary search to find both citizens. Otherwise, we know one half has no infected citizens and the other half has two. Then, we pair up corresponding citizens in the set and split the pairs arbitrarily in half. We test these halves and run the same procedure as above. We are guaranteed that this process terminates in at most $\log n$ steps because no bit strings agree in more than $\log n$ positions. Thus it takes $\log n$ tests to terminate and $2 \log n$ tests to perform binary search. Thus the amount of tests is $\Theta(\log n)$.