Worked with: Prin Pulkes, Kelsey Chen, Jayden Personnat

Problem 1

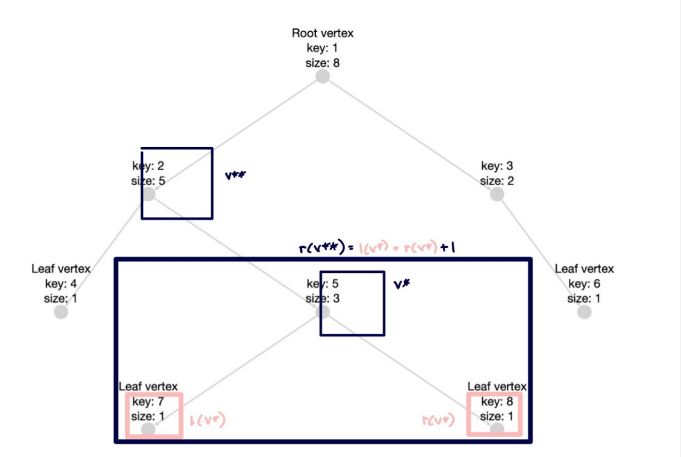
a. The program runs in $\mathcal{O}(n)$ because it recursively calls the function *calculate_sizes* on each vertex. Thus, there are n calls of *calculate_sizes* and each call performs basic operations. This produces a run time of $\mathcal{O}(n)$.

b. We first begin by defining two definitions: the potential function ϕ and v^*

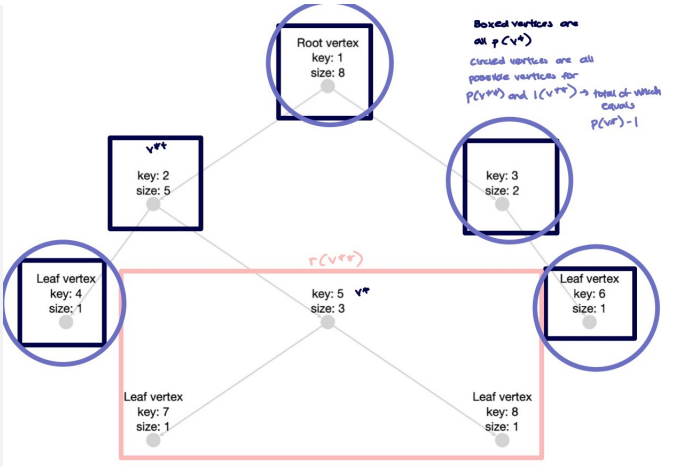
Definition 0.1 (Potential Function ϕ)**.** We define a potential function called ϕ on the vertices of the tree such that ϕ(v) equals the size of the largest subtree created by removing the vertex v.

Definition 0.2 (v*)**.** We define v* as the vertex that minimizes the value of ϕ such that ϕ(v*) $\leq \phi$(v) for all other vertices v.

Proof. We want to prove that ϕ(v*) $\leq$ n/2. Thus, we prove by contradiction: $\phi(v^*) > \frac{n}{2}$. By this statement, there is at least one subtree that has a size $> \frac{n}{2}$. Say we move in the direction of the largest subtree formed. Let's call v^{**} the child or parent of v^* formed by moving in the direction of v^*'s largest subtree (if the largest subtree is rooted at $T.root$, then v^{**} is the parent of v^*. If the largest subtree is rooted at $left(v^*)$ or $right(v^*)$ then v^{**} is a child of v^*). The combined size of the two smaller subtrees of v^* must be no greater than $\frac{n}{2} - 2$ because the smallest possible size of the largest subtree is $\frac{n}{2} + 1$ because $\phi(v^*) > \frac{n}{2}$. With v^{**}, we know that one of v^{**}'s subtrees becomes the combined size of the smaller subtrees (can be of size 0) of v^* plus one because we add v^* to one of v^{**}'s subtrees. As the combined size of the smaller subtrees of v^* must be no greater than $\frac{n}{2} - 2$, the size of one of v^{**}'s subtrees must be no greater than $\frac{n}{2} - 1$. Thus, the size of one of v^{**}'s subtrees is less than $\phi(v^*)$. Reference Figure 1 for visualization. Figure 1 uses the case where the parent subtree is the largest subtree and that v^* is the right child of v^{**}. $p(v)$, $r(v)$, $l(v)$ demarcates the parent subtree rooted at $T.root$, the right subtree, and the left subtree.



The other two subtrees of v^{**} must take from the remaining vertices left in the tree. The number of remaining vertices left in the tree equals $\phi(v*) - 1$. This is because $\phi(v*)$ defines the number of vertices in the largest subtree. Since we move from v^* to v^{**} in the direction of the largest subtree, the size of the largest subtree must decrease by 1, and since we are enclosing on the largest subtree, the subtree either divides into one or two subtrees, of which their combined size is $\phi(v^*) - 1$. Therefore, each of the one or two subtrees formed from enclosing on v^*'s largest subtree must have a size less than $\phi(v^*)$. Reference Figure 2 for visualization. Figure 2 uses the case where the parent subtree is the largest subtree and that v^* is the right child of v^{**}.



Because the parent, left, and right subtree of v^{**} all have sizes less than $\phi(v^*)$, we come to the conclusion that $\phi(v^{**}) < \phi(v^*)$ and have reached a contradiction in that v^* is not the vertex that minimizes the value of ϕ such that $\phi(v^*) \leq \phi(\text{v})$ for all other vertices v. Thus, by proof of contradiction, $\phi(v^*) \leq \frac{n}{2}$. □

c. Since we are starting from the root and traversing downwards to the child, the program runs $\mathcal{O}(h)$ time because the program only traverses one node per level (height) since it chooses which one, left or right subtree, is the larger one to iterate on.

Problem 2

a. We first begin by the given definition.

Definition 0.3 (f(n))**.** We define $f(n)$ to be the number of ways of matching donors to patients, under the given circumstances, so that each donor donates exactly one kidney to a compatible patient and each patient receives exactly one kidney from a compatible donor.

Proof. We first prove $f(1) = 0$ and $f(2) = 1$. When there is only one donor and one patient, then there are no matches possible because the one patient available is incompatible to the one donor available and vice versa. When there are two donors and two patients, then only one match is possible because the donor matches with the one patient it is not incompatible with and vice versa. Thus $f(1) = 0$ and $f(2) = 1$ holds.

We now look to cases where $n > 2$. Let d represent the first donor. Let there be n donors and n patients, where each donor can only donate to one patient. There are $n - 1$ patients to which d can donate to because d can donate to all patients but $incomp(d)$. Let d^* represent the second donor. d^*'s pairings depend on whether or not d pairs with $incomp(d^*)$

Case 0.1. Say d is paired with $incomp(d^*)$. As a result, d^* has $n - 1$ pairing possibilities. There are $f(n - 1)$ ways to match the $n - 1$ patients left. Because these pairings are dependent of each other, we invoke the multiplication rule so that if d was paired with $incomp(d^*)$, we get:

$$f(n) = (n - 1) \times f(n - 1) \tag{1}$$

Case 0.2. Say d is paired with a patient other than $incomp(d^*)$. Then d^* would have $n - 2$ pairing possibilities because it wouldn't be able to match with $incomp(d^*)$ as well as the patient d was matched to. There are $f(n - 2)$ ways to match the $n - 2$ patients left. Because these

pairings are dependent of each other, we invoke the multiplication rule so that if d is not paired with $incomp(d^*)$, we get:

$$f(n) = (n-1) \times f(n-2) \tag{2}$$

We add (1) and (2) together because combined they represent the total number of ways to match donors to patients, providing us with the equation:

$$f(n) = (n-1) \times f(n-1) + (n-1) \times f(n-2) \tag{3}$$

which can be simplified to:

$$(n-1) \times (f(n-1) + f(n-2)) \tag{4}$$

□

b. *Proof.* We prove by strong induction.

Definition 0.4 (P(n))**.** Let $P(n)$ be the predicate for the inequality for n such that $\frac{n!}{3} \leq f(n) \leq \frac{n!}{2}$ where $f(1) = 0$, $f(2) = 1$ and for all n ≥ 3, $f(n) = (n-1) * (f(n-1) + f(n-2))$.

Base Case 0.1. $P(2)$ is true because $f(2) = 1$ and the comparison $\frac{2!}{3} \leq 1 \leq \frac{2!}{2}$ holds true. $P(3)$ is true because $f(3) = 2$ and the comparison $\frac{3!}{3} \leq 2 \leq \frac{3!}{2}$ holds true.

Inductive Hypothesis 0.1. Assume that the predicate P(n) is true for all n greater than or equal to 2 such that P(2), P(3), P(4),...,P(n) are all true.

Inductive Step 0.1. To show that $P(n+1)$ is true, we show that $\frac{(n+1)!}{3} \leq f(n+1) \leq \frac{(n+1)!}{2}$ is true.

First, we know that $f(n+1) = n \times (f(n) + f(n-1))$. Through the Inductive Hypothesis, we know that $\frac{n!}{3} \leq f(n) \leq \frac{n!}{2}$ and $\frac{(n-1)!}{3} \leq f(n-1) \leq \frac{(n-1)!}{2}$. Therefore, we can combine both inequalities to give:

$$\frac{n!}{3} + \frac{(n-1)!}{3} \leq (f(n) + f(n-1)) \leq \frac{n!}{2} + \frac{(n-1)!}{2} \tag{5}$$

We then multiply the above equation by n on both sides to get:

$$n \times (\frac{n!}{3} + \frac{(n-1)!}{3}) \leq n \times (f(n) + f(n-1)) \leq n \times (\frac{n!}{2} + \frac{(n-1)!}{2}) \tag{6}$$

$$\frac{n!n}{3} + \frac{n!}{3} \leq n \times (f(n) + f(n-1)) \leq \frac{n!n}{2} + \frac{n!}{2} \tag{7}$$

$$\frac{(n+1)!}{3} \leq n \times (f(n) + f(n-1)) \leq \frac{(n+1)!}{2} \tag{8}$$

$$\frac{(n+1)!}{3} \leq P(n+1) \leq \frac{(n+1)!}{2} \tag{9}$$

We have thus shown that $P(n+1)$ is true and therefore, by the principle of mathematical induction, the inequality $P(n)$ holds true such that:

$$\frac{n!}{3} \leq f(n) \leq \frac{n!}{2} \text{ for all n} \geq 3$$

□