2 Werewolves

(a) To test if a player x is a villager, we ask the other $n-1$ players what x s identity is.
Claim: x is a villager if and only if at least half of the other players say x is a villager.To see this, notice that if x is a villager, at least half of the remaining players are also villagers, and so regardless of what the werewolves do at least half of the players will say x is a villager. On the other hand, if x is a werewolf, then strictly more than half of the remaining players are villagers, and so strictly less than half the players can falsely claim that x is a villager.

(b) The divide and conquer algorithm to find a villager proceeds by splitting the group of friends into two (roughly) equal sets A and B, and recursively calling the algorithm on A and B: $x = villager(A)$ and $y = villager(B)$, and checking x or y using the procedure in part (a) and returning one who is a villager. If there is only one player,return that player.

Proof. We will prove that the algorithm returns a citizen if given a group of n people of which a majority are villagers. By strong induction on n:
Base Case:
If $n = 1$, there is only one person in the group who is a villager and the algorithm is trivially correct.Induction hypothesis The claim holds for $k < n$.
Induction step:
After partitioning the group into two groups A and B, at least one of the two groups has more villagers than werewolves. By the induction hypothesis the algorithm correctly returns a villager from that group, and so when the procedure from part (a) is invoked on x and y at least one of the two is identified as a villager. □

Runtime Analysis. *Running time analysis Two calls to problems of size $n/2$, and then linear time to compare the two people returned to each of the friends in the input group: $T(n) = 2T(\frac{n}{2}) + \mathcal{O}(n) = \mathcal{O}(n \log n)$.*

(c) Split up the friends into pairs and if either says the other is a werewolf,discard both friends. Otherwise discard any one and keep the other friend. If n was odd, use part (a) to test whether the odd man out is a villager. If yes, you are done, else recurse on the remaining at most $n/2$ friends.

Proof. After each pass through the algorithm, villagers remain in the majority if they were in the majority before the pass. To see this, let $n1$, $n2$ and $n3$ be the pairs of friends with both villagers, both werewolves and one of each respectively.Then the fact that villagers are in the majority means that $n1 > n2$. Note that all then 3 pairs of the third kind get discarded, and one friend is retained from each of the $n1$ pairs of the first kind. So we are left with at most $n1 + n2$ friends of whom a majority $n1$ are villagers. It is straightforward to now turn this into a formal proof of correctness by strong induction on n. □

Runtime Analysis. *In a single run of the algorithm on an input set of size n, we do $\mathcal{O}(n)$ work to check whether $f1$ is a villager in the case that n is odd and to pair up the remaining friends and pruning the candidate set to at most $n/2$ people. Therefore,the runtime is given by the following recursion: $T(n) = T(\frac{n}{2}) + \mathcal{O}(n) = \mathcal{O}(n)$.*

3 the Resistance

We partition the n players into $2k$ groups, and send each group on a mission. For the missions that succeed, we remove those groups, and split the remaining groups in half to form a new set of groups. We repeat this procedure until each group has one person left, at which point we know they are a spy.

Runtime Analysis. *In the first iteration of this procedure we have $2k$ groups going on missions. After each group goes on a mission, since there are k spies, at most k of the missions fail, which means after splitting the groups that failed, we have at most $2k$ groups again.In each iteration, at least half of the players are removed from groups.*
So we identify the spies after $\mathcal{O}(\log(n/k))$ iterations. So the total number of missions needed is $\mathcal{O}(k \log(n/k))$.

4 Modular Fourier Transform

(a) 1. Show that $\{1, 2, 3, 4\}$ are the 4th roots of unity (modulo 5).

Solution. $z = 1 : z^4 \mod 5 \equiv 1 \mod 5 \equiv 1$
$z = 2 : z^4 \mod 5 \equiv 16 \mod 5 \equiv 1$
$z = 3 : z^4 \mod 5 \equiv 81 \mod 5 \equiv 1$
$z = 4 : z^4 \mod 5 \equiv 256 \mod 5 \equiv 1$

2. show that $1 + w + w^2 + w^3 = 0 \mod 5$ for $w = 2$.

Solution. *We calculate that* $\sum_{i=1}^{4} w^{i-1} = \frac{w^4-1}{w-1} = 15$, *resulted in* $1 + w + w^2 + w^3 = 15 \mod 5 \equiv 0$.

(b) Using the FFT, produce the transform of the sequence $(0, 2, 3, 0)$ modulo 5; that is, evaluate the polynomial $2x + 3x^2$ at $\{1, 2, 4, 3\}$ using the recursive FFT algorithm defined in class, but with $w = 2$ and in modulo 5 instead of with $w = i$ in the complex numbers.All calculations should be performed modulo 5.

Solution. *1.* $w = 8$*:* $\{1\}$
group1: $index = 1 : 0 \mod 5 \equiv 0$
group2: $index = 2 : 2x \mod 5 \equiv 2$
group3: $index = 4 : 3x^2 \mod 5 \equiv 3$
group4: $index = 3 : 0 \mod 5 \equiv 0$
2. $w = 4$*:* $\{1, 4\}$
group1:
$index = 1 : 0 + 1 * 3 \mod 5 \equiv 3$
$index = 4 : 0 + 4 * 3 \mod 5 \equiv 2$
group2:
$index = 2 : 2 + 1 * 0 \mod 5 \equiv 2$
$index = 3 : 2 + 4 * 0 \mod 5 \equiv 2$
3. $w = 2$*:* $\{1, 2, 4, 3\}$
group1:
$index = 1 : 3 + 1 * 2 \mod 5 \equiv 0$
$index = 2 : 2 + 2 * 2 \mod 5 \equiv 1$
$index = 4 : 3 + 4 * 2 \mod 5 \equiv 1$
$index = 3 : 2 + 3 * 2 \mod 5 \equiv 3$

(c) Now perform the inverse FFT on the sequence (0, 1, 4, 0), also using the recursive algorithm. Recall that the inverse FFT is the same as the forward FFT, but using w^{-1} instead of w, and with an extra multiplication by 4^{-1} for normalization.

Solution. *1.* $w^{-3} = 27$*:* $\{1\}$
group1: $index = 1 : 0 \mod 5 \equiv 0$
group2: $index = 3 : x \mod 5 \equiv 1$
group3: $index = 4 : 4x^2 \mod 5 \equiv 4$
group4: $index = 2 : 0 \mod 5 \equiv 0$
2. $w^{-2} = 9$*:* $\{1, 4\}$
group1:
$index = 1 : 0 + 1 * 4 \mod 5 \equiv 4$
$index = 4 : 0 + 4 * 4 \mod 5 \equiv 1$
group2:
$index = 3 : 1 + 1 * 0 \mod 5 \equiv 1$
$index = 2 : 1 + 4 * 0 \mod 5 \equiv 1$
3. $w^{-1} = 2^{-1} = 3$*:* $\{1, 3, 4, 2\}$
group1:
$index = 1 : 4 + 1 * 1 \mod 5 \equiv 0$
$index = 3 : 1 + 2 * 1 \mod 5 \equiv 3$
$index = 4 : 4 + 4 * 1 \mod 5 \equiv 3$
$index = 2 : 1 + 3 * 1 \mod 5 \equiv 4$
4.normalization: $(0, 3, 3, 4) \Rightarrow (0, 3/4, 3/4, 1)$

(d) Now show how to multiply the polynomials $3x + 2x^2$ and $3 - x$ using the FFT modulo 5. You may use the fact that the FFT of (3, 4, 0, 0) modulo 5 is (2, 1, 4, 0) without doing your own calculation.

Step 1. *exert FFT on the polynomial* $3x + 2x^2$
1. $w = 8$*:* $\{1\}$
group1: $index = 1 : 0 \mod 5 \equiv 0$
group2: $index = 2 : 3x \mod 5 \equiv 3$
group3: $index = 4 : 2x^2 \mod 5 \equiv 2$
group4: $index = 3 : 0 \mod 5 \equiv 0$
2. $w = 4$*:* $\{1, 4\}$
group1:
$index = 1 : 0 + 1 * 2 \mod 5 \equiv 2$
$index = 4 : 0 + 4 * 2 \mod 5 \equiv 3$
group2:
$index = 2 : 3 + 1 * 0 \mod 5 \equiv 3$
$index = 3 : 3 + 4 * 0 \mod 5 \equiv 3$
3. $w = 2$*:* $\{1, 2, 4, 3\}$
group1:
$index = 1 : 2 + 1 * 3 \mod 5 \equiv 0$
$index = 2 : 3 + 2 * 3 \mod 5 \equiv 4$
$index = 4 : 2 + 4 * 3 \mod 5 \equiv 4$
$index = 3 : 3 + 3 * 3 \mod 5 \equiv 2$

Step 2. *exert FFT on the polynomial* $3 - x$
1. $w = 8$*:* $\{1\}$
group1: $index = 1 : 3 \mod 5 \equiv 3$
group2: $index = 2 : -x \mod 5 \equiv 4$
group3: $index = 4 : 0 \mod 5 \equiv 0$
group4: $index = 3 : 0 \mod 5 \equiv 0$
2. $w = 4$*:* $\{1, 4\}$
group1:
$index = 1 : 3 + 1 * 0 \mod 5 \equiv 3$
$index = 4 : 3 + 4 * 0 \mod 5 \equiv 3$
group2:
$index = 2 : 4 + 1 * 0 \mod 5 \equiv 4$
$index = 3 : 4 + 4 * 0 \mod 5 \equiv 4$
3. $w = 2$*:* $\{1, 2, 4, 3\}$
group1:
$index = 1 : 3 + 1 * 4 \mod 5 \equiv 2$
$index = 2 : 3 + 2 * 4 \mod 5 \equiv 1$
$index = 4 : 3 + 4 * 4 \mod 5 \equiv 4$
$index = 3 : 3 + 3 * 4 \mod 5 \equiv 0$

Step 3. *polynomial multiplication*
$p1(0, 4, 4, 2) * p2(2, 1, 4, 0) \mod 5 = p(0, 4, 1, 0)$

Using the fact the problem given, we attain the answer (2, 1, 4, 0).

5 Patern Matching

(a)

Solution. *How many possible n-length substrings are there? at most* $m - 1$*. Note that we may check each substring against our reference substring* g *in* $\mathcal{O}(n)$ *time by comparing each index. Therefore our total runtime is* $\mathcal{O}((m-1)n) = \mathcal{O}(mn)$*.*

(b)

Solution. *Set the matching function*

$$match(i, j) = (g(i) - s(j))^2 \tag{1}$$

If $match(i, j) = 0$*, then position* i *of* g *and position* j *of* s *are successfully matched, else* $match(i, j) = 1$*.*
The exact match function

$$P(x) = \sum_{i=0}^{n-1} match(i, x - n + 1 + i) \tag{2}$$

which means the end of x in s, push forward n lengths, and perform matching function operations one by one. The theoretical basis for this prove is that if $P(x) = k$, the match is in $n - k$ positions.
Then define the reverse function to flip g:

$$reverse(x) = g(n - x - 1) \tag{3}$$

Let $g(x) = reverse(n - x - 1)$, then

$$\begin{aligned} P(x) &= \sum_{i=0}^{n-1} match(x) \\ &= \sum_{i=0}^{n-1} [reverse(n-i-1) - s(x-n+1+i)]^2 \\ &= \sum_{i=0}^{n-1} reverse(n-i-1)^2 + \sum_{i=0}^{n-1} s(x-n+i+1)^2 - 2\sum_{i=0}^{n-1} reverse(n-i-1)s(x-n+i+1) \end{aligned}$$

$\sum_{i=0}^{n-1} reverse(n-i-1)s(x-n+i+1)$ *is typical convolutional form, easily solved with FFT.*

Runtime Analysis. *Coumpting the formula*

$$\sum_{i=0}^{n-1} reverse(n-i-1)^2 + \sum_{i=0}^{n-1} s(x-n+i+1)^2$$

can be done in $\mathcal{O}(n)$ time, doing the multiplication

$$\sum_{i=0}^{n-1} reverse(n-i-1)s(x-n+i+1)$$

can be done in $\mathcal{O}(m \log m)$ time leveraging the FFT, and therefore final runtime: $\mathcal{O}(m \log m)$.

(c)

Solution. *Construct a length 4n string g' from g by the following rule: replace (A) with (0, 0, 0, 1), replace (C) with (0, 0, 1, 0), replace (G) with (0, 1, 0, 0), replace (T) with (1, 0, 0, 0). Construct a length 4m string s' from s following the same rule. Construct m - n strings of length 4m by initializing 4m zeroes, then placing g at index 0, then 4, then 8, then 12, etc. Place these strings into a matrix as rows, sorted in increasing order by the index that we placed g'. Multiply this constructed matrix by s . This multiplication yields a length $m - k$ vector: one index for each possible starting index of g. Accept each index which is within k of n, these are your solutions.*

$$\begin{bmatrix} g' & 0 & \dots & 0 \\ 0 & g' & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & g' \end{bmatrix} \begin{bmatrix} s' \end{bmatrix} = answer\ goodness\ (accept\ high\ values)$$

Proof. Note that for each row, every correct index will add 1 to the output at the location where the string started. Every incorrect index will add 0 to the output at the location where the strings started. Therefore, a row (corresponding to an index) that is correct in all but k positions will result in an output row with value m − k, and we accept if within bounds because this index matches our conditions. □

Runtime Analysis. *Generating the matrix can be done in $\mathcal{O}(m)$ time by shifting, doing the multiplication can be done in $\mathcal{O}(4m \log(4m)) = \mathcal{O}(m \log m)$ time leveraging the FFT. Interpreting the result can be done in $\mathcal{O}(m)$ time, and therefore final runtime: $\mathcal{O}(m \log m)$.*