# CS 70 Discrete Mathematics and Probability Theory
# Spring 2015 Vazirani HW 7

**Reminder:** Deadline for HW6 self-grade is Thursday, March 5 at noon.

*Reading:* Note 8 and Note 9.

## Due Monday March 9

1. **Polynomial Interpolation Virtual Lab**

   Follow the virtual lab for this homework (provided on the course website). Include screenshot(s) in your homework submission. Also provide answers to the questions from the virtual lab in your homework submission.

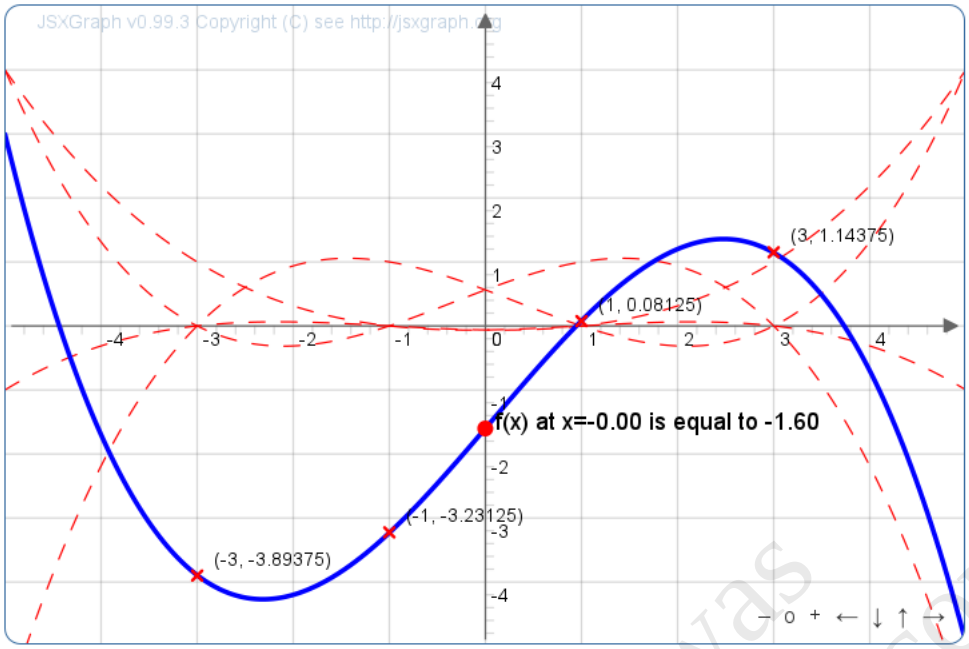
   **Answer:**

   **Drag the red points around and watch how the polynomials change. How can you drag a point such that only one scaled delta polynomial (the dashed ones) change?**

   Dragging the point vertically will change only one scaled delta polynomial, since all the $x_i$'s stay the same, and only one $y_i$ changes.

   **Once you successfully recovered the secret, take a screenshot of the plots and your secret, and include it in your writeup.**

   The following screenshot shows the correct values of the delta polynomial. The secret assigned to you is random and the value you get will vary.




Compute the 4 delta polynomials that passes the 4 points given, then click "Plot" to plot them, as well as $P(x) = y_1\Delta_{x_1} + y_2\Delta_{x_2} + y_3\Delta_{x_3} + y_4\Delta_{x_4}$, which should be a polynomial passing through all 4 points given. You can type in expressions such as $(x + 10) * (x - 12)/(-3)$ into the text boxes.

$\Delta_{x_1}$ (x+1)*(x-1)*(x-3)/(-48)

$\Delta_{x_2}$ (x+3)*(x-1)*(x-3)/16

$\Delta_{x_3}$ (x+3)*(x+1)*(x-3)/(-16)

$\Delta_{x_4}$ (x+3)*(x+1)*(x-1)/48

Now read off the secret, which is the value of the polynomial at $x = 0$.

Plot

Paste the secret here (to 1 decimal place): -1.6 Check secret

You've got the secret!

**With this information and the interactive graph above to help you, can you give a rough range for the value of the secret?**

Sliding the last point between -5 and 5, we see that the secret's value at $P(0)$ ranges between approximately -2 and 0. We see that we have narrowed the secret's possible value from all reals to a smaller range.

**Would this be a problem if we switched to finite fields? Why or why not?**

No, this would not be a problem. Suppose we are working in a finite field $GF(p)$ and we are given access to $d$ points. Observe that the secret comes from $\{0, 1, \ldots, p-1\}$, and for each of these values there exists a polynomial of degree at most $d$ that is consistent with the $d$ given points and the secret. Therefore we cannot gain any information about the secret.

*(Note: when we were working with reals, there was leakage of information because there had to be some bound on the y-coordinate of each point. There would be no leakage of information if we were allowed to work with unbounded values, but that is an impractical assumption.)*

**Why do you think most entries are green when you start?**

It is because the left-hand side of the equations, given by $Q(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$, depends only on $x$. Since we used the same $x$-values of $1,2,3,4,5,6$ for both messages, the left-hand side stays unchanged.

2. **RSA vulnerability**

When working with RSA, it is not uncommon to use $e = 3$ in the public key. Suppose that Alice has sent Bob, Carol, and Dorothy the same message indicating the time she is having her birthday party. Eve who is not invited wants to decrypt the message and show up uninvited to the party.

Bob, Carol, and Dorothy have public keys $(N_1,e_1),(N_2,e_2),(N_3,e_3)$ where $e_1 = e_2 = e_3 = 3$. Of course, $N_1,N_2,N_3$ are all different, as are their prime factors. Alice has chosen a number $0 \leq x < N_1,N_2,N_3$ which indicates the time her party starts and encodes it and sends it to each of her three friends as $E_i(x) = x^3 \pmod{N}_i$. Eve has been able to intercept all three encoded messages, but is not very optimistic about decrypting any of them given the security of RSA. Can you show her an efficient method of obtaining $x$ given the information she has.

**Answer:**

Let the three encoded messages be $y_1,y_2,y_3$. Since the messages are encoded by RSA with public keys $(N_1,3)$, $(N_2,3)$, and $(N_3,3)$, we have:

$$x^3 \equiv y_1 \bmod N_1$$
$$x^3 \equiv y_2 \bmod N_2$$
$$x^3 \equiv y_3 \bmod N_3$$

Since $N_1,N_2,N_3$ do not share any prime factor, by using the Chinese Remainder Theorem, we can solve the above system of congruence equations. Let the solution be

$$x^3 \equiv x_0 \bmod N_1N_2N_3$$

with $0 \leq x_0 < N_1N_2N_3$. Since $x < N_1,N_2,N_3$, $x^3 < N_1N_2N_3$ and thus $x^3 = x_0$. We can take the cube root of $x_0$ and recover the original message $x = x_0^{1/3}$. In this problem, the trick is that we were able to convert a problem of finding cube-roots mod a prime (which is hard) into finding cube-roots in the integers (which is easy).

*(Note: in fact, the assumption that $N_i$'s all have different prime factors is unnecessary. We can easily test if they share any prime factor using Euclid's algorithm, and if they do, we have already broken the security of RSA, as we have factored at least one of the $N_i$'s.)*

3. **Hierarchical secret sharing**

Suppose that Google's most precious asset was the simple algorithm at the core of Google search[1]. This secret is closely guarded, and indeed no single individual at Google knows the algorithm. In fact, no individual at Google knows anything about the algorithm, they just oversee the machines that run it.

In the case of an emergency, Google would like to be able to recover this core algorithm. Google is run by its president and board of 9 directors, and they would like a group including the president and a majority of the members of the board to be able to recover the algorithm. But they would also like to ensure that no other group can learn *anything* about the algorithm.

[1] Disclaimer: there is no simple algorithm at the core of Google search.

Describe a scheme which has the desired property. You should specify some data to give to the president and to each member of the board, such that the president and 5 members of the board can recover the secret algorithm but such that any group which doesn't contain the president, or any group which has only 4 members of the board, can't learn anything about the algorithm.

**Answer:** The solution is a simple hierarchical scheme that uses two polynomials; we first encode the given secret as $P(0)$ of a degree-1 polynomial $P$ and give $P(1)$ to the president. Instead of giving out $P(2)$ to the board members directly, we further encode $P(2)$ as $Q(0)$ of a degree-4 polynomial $Q$ and distribute $Q(1), Q(2), \ldots, Q(9)$ to each board member.

The main idea for the solution comes from the simple observation that if the board consisted of only one single member, then the problem would be easy. It would be an instance of secret sharing with two people who both need to provide their shares to figure out the secret. To do that one would need a polynomial of degree at most 1, $P(x)$, where $P$ encodes the secret algorithm somehow (for example at $P(0)$), and then $P(1)$ would be given to the president and $P(2)$ would be given to the board. Now, knowing either $P(1)$ or $P(2)$ would not give us any information about $P(0)$, because $P(0)$ can be anything. But knowing both $P(1)$ and $P(2)$ would give us the exact polynomial $P$ and we could recover the secret algorithm from it. (Assume that we are working mod some prime number $p$ that is large enough that the secret algorithm can be encoded as a number mod $p$.)

When the board of directors consists of more than one member however this scheme does not work anymore, because we can only specify a minimum required number of people with this scheme; there is no room for a constraint such as the president's share being present.

What we can do instead, is to do a hierarchical scheme with two polynomials. Suppose that we use the same polynomial $P$ as before (with degree at most 1), but now $P(2)$ which is the board's share is not given to any individual member of the board; rather it is also subject to a secret sharing scheme. So we take $P(2)$ and encode it in a polynomial $Q$ of degree at most 4 (for the sake of concreteness, assume that we encode it as $Q(0)$, but there are other methods as well). Now we give $Q(1), Q(2), \ldots, Q(9)$ to each of the individual board members. If less than 5 board members come together, their combined shares would not give us any information about $Q(0)$, and thus we would have no information about $P(2)$, and therefore no information about $P(0)$ which is the secret. However if we have at least 5 board members, they together can recover $Q$ and therefore $Q(0)$ which is $P(2)$. That by itself is not enough to learn anything about $P(0)$, but if it is combined with the president's share which is $P(1)$, we can recover $P(x)$ and therefore $P(0)$ exactly.

4. **Adversarial Secret Sharing**

Suppose that I want to break up a secret into shares such that any set of $k$ people can recover the secret, but I'm also worried that some people might be dishonest and may lie about the secrets they have (in order to prevent the other people from recovering the correct secret). Show that in the ordinary secret sharing scheme based on polynomials, any group of $k+2$ people can recover the secret even if one of them is dishonest.

**Answer:** This follows from making a simple observation that secret sharing and error-correcting codes not only both use polynomials, but also in almost identical manner. Indeed we can simply use Berlekamp-Welch algorithm to recover the secret even when one of the $k+2$ people is lying, thereby considering the lying person as an error.

In the given secret sharing scheme, we are using a polynomial $P(x)$ of degree at most $k-1$ and are giving the value of $P$ at different points to the people. If we have the exact value of $P$ at $k$ points, then we can of course recover $P$ by interpolation and therefore recover the secret.

But note that error-correcting codes fall into a very similar scenario. In particular, if we know the value of a polynomial $P(x)$ of degree at most $d-1$ at $d+2r$ different points with $r$ of those values being erroneous, then the Berlekamp-Welch algorithm would be able to recover $P(x)$ exactly.

Here we have a polynomial of degree at most $k-1$ and we know its value at $k+2$ points (the $k+2$ people who give us their shares). We know that at most one of these values is erroneous (the one person who is dishonest). This is identical to the situation described with $r=1$. Therefore if we just run the Berlekamp-Welch algorithm, we would be able to recover $P(x)$ from the $k+2$ values that contain at most 1 error.

5. **Sharing RSA**

Complaints received by the human resources department of a company are very sensitive, since they might invoke retaliation of all kinds. The HR department is trying to build a system where employees can securely send messages to the department. The following properties are desired:

- The messages should be encrypted; if someone (e.g. from the IT department) intercepts the messages sent over the network, they should not be able to read them.
- No single person from the HR department should be able to read or understand anything about the encrypted messages (the complaints might be about them). But any two members of the HR team should be able to come together and read all the messages they have received.

How should the HR department implement its system? Describe your method and prove that it has the desired properties.

**Answer:**

Without the requirement that at least two members of the HR department be present, this would be an easy problem, because simple public-key encryption schemes such as RSA satisfy the requirement. The HR department would have a private key (consisting of $p$, $q$, and $d$) and a public key consisting of ($N=pq$ and $e$). Any employee wanting to send a message to the HR department would use the public key to encrypt the message they want to send, and even if someone intercepts that message it would be hard for them to decrypt it without knowing the secret key (which only the HR department has).

But the problem is that the members of the HR department have access to the private key, and so it is easy enough for any single one of them to decrypt the messages. But we know how to handle this problem. The solution is secret sharing. Suppose that we use a polynomial $P(x)$ of degree 1 mod some large prime number $r$. Then we can encode RSA's private key in $P(x)$; there are multiple ways to do this. For one example, we can append $p$, $q$, $d$ together to get a large number, and we just need to make sure $r$ is larger than this, and then we can let $P(0)$ be that number. Now, we give $P(1), P(2), \ldots$ to different employees of the HR department. Any single one of them has no information about $P(0)$, because for any possible value of $P(0)$, there is exactly one polynomial that is compatible with their share and has that $P(0)$. Therefore any member of the HR department would know nothing more about the private key than the information that can be already recovered from the public key. So it would be as hard for them as any other person (such as the IT employee who intercepts messages) to decrypt the messages. But if any two of the HR department employees come together, they can recover $P$ exactly and thus $P(0)$ which is the private key for RSA. Using the private key they can decrypt all of the messages sent to them.

6. **Polynomial GCD**

Suppose that we have two polynomials with rational coefficients $P(x), Q(x)$. Then by polynomial long division, we can extract two polynomials $D(x), R(x)$ such that $P(x) = Q(x)D(x) + R(x)$ and where the degree of $R(x)$ is less than the degree of $Q(x)$. If $R(x) = 0$ we say that $Q(x)$ divides $P(x)$.

(a) Do the polynomial long division for the polynomials $P(x) = x^4 + 3x^3 + 1$ and $Q(x) = x^2 + 1$. What is $D(x)$ and what is $R(x)$?

(b) The GCD of two polynomials $P(x), Q(x)$ is a polynomial $C(x)$ that divides both $P(x)$ and $Q(x)$, and such that for any polynomial $T(x)$ that divides both $P(x)$ and $Q(x)$, it must be true that $T(x)$ divides $C(x)$.

Prove that if GCD of $P(x), Q(x)$ is the polynomial $C(x)$, then $C(x)$ must have the largest degree among polynomials that divide both $P(x)$ and $Q(x)$.

(c) Prove that the set of common polynomial divisors of $P(x), Q(x)$ is the same as the set of common polynomial divisors of $Q(x), R(x)$.

(d) The last part lets us compute the GCD of two polynomials by an algorithm that looks just like Euclid's GCD algorithm. Write such an algorithm for polynomial GCD.

(e) Your algorithm will runs until one of the two polynomials becomes 0. At that point the other polynomial is the GCD. Prove using induction that this must happen at some point.

(f) Run your algorithm for the two polynomials $P(x) = x^4 + x^3 + 2x^2 + x + 1$ and $Q(x) = x^4 + 3x^2 + 2$ to compute their GCD. Show your work.

**Answer:**

(a) Here you can see the results of the long division.

$$
\begin{array}{r}
x^2+3x-1 \\
x^2+1\,\overline{)\;x^4+3x^3\qquad\quad+1} \\
\underline{-x^4\qquad-x^2} \\
3x^3-x^2 \\
\underline{-3x^3\qquad-3x} \\
-x^2-3x+1 \\
\underline{x^2\qquad+1} \\
-3x+2
\end{array}
$$

The resulting remainder (or $R(x)$) is $-3x+2$ and the the resulting $D(x)$ is simply $x^2+3x-1$. In other words we have

$$x^4 + 3x^3 + 1 = (x^2+1)(x^2+3x-1) + (-3x+2)$$

(b) Let $T(x)$ be any common divisor of $P(x)$ and $Q(x)$. Then by definition it must be true that $T(x)$ divides $C(x)$, i.e. $C(x) = T(x)F(x)$. Since $C(x)$ is nonzero, $T(x)$ and $F(x)$ must also be nonzero, so $\deg(C) = \deg(T) + \deg(F)$. Therefore, $\deg(C) \geq \deg(T)$, as desired.

(c) If $T(x)$ is a common divisor of $P(x)$ and $Q(x)$ then it means that $P(x) = T(x)U_1(x)$ and $Q(x) = T(x)U_2(x)$. Now, note that we have

$$R(x) = P(x) - Q(x)D(x) = T(x)(U_1(x) - U_2(x)D(x)).$$

Therefore $T(x)$ divides $R(x)$ as well, and so $T(x)$ is a common divisor of $Q(x)$ and $R(x)$.

For the other direction, note that if $T(x)$ is a common divisor of $Q(x)$ and $R(x)$, then it must be that $Q(x) = T(x)V_1(x)$ and $R(x) = T(x)V_2(x)$. Now, note that

$$P(x) = Q(x)D(x) + R(x) = T(x)(V_1(x)D(x) + V_2(x)).$$

So $T(x)$ also divides $P(x)$, which means that $T(x)$ is a common divisor of $P(x)$ and $Q(x)$.

(d) We follow exactly the Euclidean algorithm with integer division replaced by polynomial long division.

---

**Algorithm 1**: POLYNOMIALGCD

---

**Input**: Polynomials $P(x)$ and $Q(x)$.
**Output**: The GCD of $P(x)$ and $Q(x)$.
**if** *P(x)=0* **then**
  **return** *Q(x)*
**else if** *Q(x)=0* **then**
  **return** *P(x)*
**else**
  Do the polynomial long division on $P(x)$ and $Q(x)$ and let the result be $D(x), R(x)$. i.e. $P(x) = Q(x)D(x) + R(x)$ where $\deg(R) < \deg(Q)$. Recursively call POLYNOMIALGCD with inputs $Q(x)$ and $R(x)$ and return the result.

---

Note that we proved that the set of common divisors of $P(x), Q(x)$ is the same as the set of common divisors of $Q(x), R(x)$. Therefore the GCD of $P(x), Q(x)$ is the same as the GCD of $Q(x), R(x)$. Therefore if the algorithm's recursion stops, we must have found the correct GCD. In the next part we prove that the algorithm stops.

(e) We use induction on the maximum degree of $P(x)$ and $Q(x)$.

**Base Case** When the maximum degree is at most 0, then both $P$ and $Q$ are constants. Therefore $R(x)$ would be 0 and $D(x)$ would be a constant. Therefore in the next step one of the two polynomials is zero, and the algorithm will be finished.

**Induction Hypothesis** Assume that the algorithm stops in finitely many steps when $\max(\deg(P), \deg(Q)) \leq k$.

**Induction Step** Suppose now that we have two polynomials $P(x), Q(x)$ whose maximum degree is at most $k+1$. After one step of the algorithm, $P(x)$ gets replaced by $R(x)$ whose degree is strictly less than $Q(x)$. Therefore the degree of $R(x)$ is at most $k$. At the next step, we do long division with $Q(x)$ and $R(x)$. So $Q(x)$ would be replaced by a polynomial whose degree is less than $R(x)$ which is at most $k$. Therefore after two steps the maximum degree becomes at most $k$, and by the inductive hypothesis, the algorithm ends after finitely many steps.

(f) We start with $P_0(x) = x^4 + x^3 + 2x^2 + x + 1$ and $Q_0(x) = x^4 + 3x^2 + 2$. After doing the long division we get

$$\begin{array}{r|l} & 1 \\ \hline x^4+3x^2+2 & x^4+x^3+2x^2+x+1 \\ & -x^4 \quad -3x^2 \quad -2 \\ \hline & x^3 \quad -x^2+x-1 \end{array}$$

Now we have the two polynomials $P_1(x) = Q_0(x) = x^4 + 3x^2 + 2$ and $Q_1(x) = x^3 - x^2 + x - 1$. If we do the long division here we get

$$
\begin{array}{r|l}
 & x+1 \\
\hline
x^3-x^2+x-1 & x^4 \quad +3x^2 \quad +2 \\
 & \underline{-x^4+x^3 \quad -x^2+x} \\
 & \quad x^3+2x^2+x+2 \\
 & \quad \underline{-x^3 \quad +x^2-x+1} \\
 & \quad\quad 3x^2 \quad +3
\end{array}
$$

In the next step we have $P_2(x) = Q_1(x) = x^3 - x^2 + x - 1$ and $Q_2(x) = 3x^2 + 3$. If we do the long division here we get

$$
\begin{array}{r|l}
 & \frac{1}{3}x-\frac{1}{3} \\
\hline
3x^2+3 & x^3-x^2+x-1 \\
 & \underline{-x^3 \quad -x} \\
 & \quad -x^2 \quad -1 \\
 & \quad \underline{x^2 \quad +1} \\
 & \quad\quad 0
\end{array}
$$

we get zero as the remainder. So $P_3(x) = Q_2(x) = 3x^2 + 3$ and $Q_3(x) = 0$. Therefore the GCD is simply $3x^2 + 3$.

*(Note: the polynomial GCD is actually determined up to a constant, because multiplying by nonzero constants does not change anything in terms of polynomial divisibility. So $3x^2 + 3 = 3(x^2 + 1)$ means that $x^2 + 1$ is also another GCD. In general all GCDs of these two polynomials will be of the form $c(x^2 + 1)$ where $c$ is a nonzero constant.)*

7. **Horner's method**

Suppose that we are given a polynomial $P(x) = c_n x^n + \cdots + c_0$ and are asked to compute $P(a)$ where $a$ is a given number. A naive way is to compute each $a^i$ for $i = 0, \ldots, n$ by multiplying $a$ with itself enough times; then multiply $a^i$ by $c_i$, and sum the result for $i = 0, \ldots, n$.

(a) Prove using induction on $n$ that the naive method requires $0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$ multiplications.

(b) We can improve upon the naive method by observing that once we have computed $a^i$, we can compute $a^{i+1}$ by just one more multiplication (there is no need to compute it from scratch). Show that if we employ this observation, the number of multiplications reduces to $0 + 1 + 2 + 2 + 2 + \cdots + 2 = 2n - 1$.

(c) Horner's method reduces that number even further. It starts from $c_n$; from that it computes $c_n a + c_{n-1}$ by one multiplication and one addition. From that it computes $c_n a^2 + c_{n-1} a + c_{n-2}$ by one more multiplication and one more addition.

In general the method starts from $v_0 = c_n$; then it recursively computes $v_i := a v_{i-1} + c_{n-i}$. Then the method outputs $v_n$. Prove using induction that this method works (i.e. that $v_n = P(a)$) and that it uses $n$ multiplications.

**Answer:**

(a) We use induction on $n$ to prove this.

**Base Case** When $n = 0$, we do 0 multiplications since the answer is simply $c_0$. This takes $0 = \frac{0(0+1)}{2}$ multiplications.

**Induction Hypothesis** Assume that for polynomials of degree $n-1$ we take $0+\cdots+(n-1) = \frac{(n-1)n}{2}$ multiplications.

**Inductive Step** When computing the polynomial $P(x)$ of degree $n$, we first compute the $c_n x^n$ term. This term takes $n$ multiplications, because we need to multiply $n+1$ terms ($n$ items are $x$ and one term is $c_n$). After we compute that term, we need to compute $c_{n-1}x^{n-1}+\cdots+c_0$, which by the inductive hypothesis takes $0+\cdots+(n-1) = \frac{(n-1)n}{2}$ multiplications. If we add $n$, we get $0+\cdots+n = \frac{(n-1)n}{2}+n = \frac{n(n+1)}{2}$ multiplications.

(b) In order to compute the term $c_0$, we need 0 multiplications. For the second term, i.e. $c_1 x$ we need 1 multiplication. We note that for each subsequent term we need two multiplications, e.g., to compute $c_i x^i$, we first need to multiply $x^{i-1}$ by $x$ to obtain $x^i$, and then multiply $x^i$ by $c_i$. Therefore, the total number of multiplications is $0+1+2+2+2+\cdots+2 = 2n-1$.

(c) We use induction:

**Claim:** In the $i$-th round of Horner's method, we have $v_i = c_n a^i + c_{n-1}a^{i-1} + c_{n-2}a^{i-2} + \cdots + c_{n-i}$ and the number of multiplications used up to that round is $i$.

**Base Case** When $i = 0$, trivially $v_0 = c_n$ and we have used 0 multiplications.

**Induction Hypothesis** Assume that in the $(i-1)$-th round of Horner's method, we have $v_{i-1} = c_n a^{i-1} + c_{n-1}a^{i-2} + c_{n-2}a^{i-3} + \cdots + c_{n-i+1}$ and the number of multiplications used up to that round is $i-1$.

**Inductive Step** In the $i$-th round, we compute $v_i = av_{i-1} + c_{n-i}$. Since we use one additional multiplication, the number of multiplications we have used so far is $i$, as desired. Moreover, by induction hypothesis, $v_{i-1} = c_n a^{i-1} + c_{n-1}a^{i-2} + c_{n-2}a^{i-3} + \cdots + c_{n-i+1}$. Therefore,

$$\begin{aligned} v_i &= av_{i-1} + c_{n-i} \\ &= a(c_n a^{i-1} + c_{n-1}a^{i-2} + c_{n-2}a^{i-3} + \cdots + c_{n-i+1}) + c_{n-i} \\ &= c_n a^i + c_{n-1}a^{i-1} + c_{n-2}a^{i-2} + \cdots + c_{n-i+1}a + c_{n-i}, \end{aligned}$$

as desired.

Hence, after the $n$-th round of Horner's method, $v_i = P(a)$ and we have used $n$ multiplications.