Problem 1.

a.

The set of all natural numbers q for which $F(q) \geq 1.6^{q-2}$ is

$$\{ q \in \mathbb{N} \mid q \geq 1 \}.$$

Let $P(q)$ be the statement: $F(q) \geq 1.6^{q-2}$.

Theorem: For all $q \geq 1$, $q \in \mathbb{N}$, $P(q)$ is true. Proof by strong mathematical induction.

Basis step:

- $P(1)$ holds: $1 \geq 1.6^{-1}$.
- $P(2)$ holds: $1 \geq 1$.

Inductive step: [Show that for all natural numbers $k \geq 2$, if $P(i)$ is true for all integers i such that $1 \leq i \leq k$, then $P(k+1)$ is also true.] Let $k \geq 2$ be an integer such that $F(i) \geq 1.6^{i-2}$ for all $1 \leq i \leq k$.

By definition of F, we have

$$F(k+1) = F(k) + F(k-1).$$

Therefore, by the inductive hypothesis,

$$\begin{aligned}
F(k+1) &\geq 1.6^{k-2} + 1.6^{k-3} \\
&= 1.6 \cdot 1.6^{k-3} + 1.6^{k-3} \\
&= 1.6^{k-3}(1.6 + 1) \\
&= 2.6 \cdot 1.6^{k-3} \\
&= 2.6 \cdot \frac{1.6^2}{1.6^2} \cdot 1.6^{k-3} \\
&= \frac{2.6}{1.6^2} \cdot 1.6^{k-1} \\
&= \frac{2.6}{2.56} \cdot 1.6^{k-1} \\
&\geq 1.6^{k-1}.
\end{aligned}$$

We have shown $F(k+1) \geq 1.6^{k-1}$ and therefore $P(k+1)$ holds. Now the conclusion follows from the basis step, the inductive step, and the principle of strong mathematical induction.

b.

Yes. We can choose $c = 1.6^{-2}$ and $n_0 = 1$. Notice with this choice of c that the expression $c \cdot 1.6^n$ becomes

$$\begin{aligned}&(1.6^{-2})1.6^n \\ &= 1.6^{n-2}.\end{aligned}$$

We showed in the previous part that $F(n) \geq 1.6^{n-2}$ for all $n \geq 1$. Hence we conclude the assertion is implied by the previous part.

c.

Let $P(n)$ be the statement: `fib`(n) is correct.

Theorem: $P(n)$ is true for all $n \in \mathbb{N}$. Proof by strong mathematical induction.

Basis step:

- $P(0)$ holds: $F(0)$ returns 0 which is correct.
- $P(1)$ holds: $F(1)$ returns 1 which is correct.
- $P(2)$ holds: $F(2)$ returns 1 which is correct.

Inductive step: [Show that for all natural numbers $k \geq 2$, if $P(i)$ is true for all natural numbers n such that $n \leq k$, then $P(k+1)$ is also true.] Let $k \geq 2$ be an integer such that `fib`(n) is correct for all $n \leq k$.

For `fib`$(k+1)$: by the inductive hypothesis, $k+1 \geq 3$, so we enter the `else` case meaning we have `fib`$(k+1)$ = `fib`(k) + `fib`$(k-1)$. By the inductive hypothesis, `fib`(k) will return the correct result and `fib`$(k-1)$ will return the correct result. By definition, `fib`$(k+1)$ = `fib`(k) + `fib`$(k-1)$ so we have that `fib`$(k+1)$ returns the correct result.

Therefore $P(k+1)$ holds. Now the conclusion follows from the basis step, the inductive step, and the principle of strong mathematical induction.

Problem 2.

a.

- **Claim:** For any array A, and natural number j such that (i) A has (at least) $j+1$ cells, and (ii) the subarray $A[1..j]$ is sorted, when `PutInPlace`(A, j, x) terminates, the first $j+1$ cells of A contain all the elements that were originally in $A[1..j]$ plus x in sorted order. Proof by induction.

  Basis step: The base case is when $j = 0$, for which the claim is true as `PutInPlace` just puts x in the first cell of A.

  Inductive step: Let $j \geq 1$ be an arbitrary integer and assuming the claim holds for $j-1$ we show it also holds for j. If x is greater than $A[j]$, then x is greater than all elements in $A[1..j]$, so by putting x in the $j+1$-th cell the array $A[1..j+1]$ satisfies the required: sorted, and contain all the required elements.

  Otherwise (if x is less than or equal to $A[j]$), we put the element at $A[j]$ into the $j+1$-th cell and call `PutInPlace`$(A, j-1, x)$. Now, the largest element of subarray $A[1..j+1]$ is at position $j+1$ since we know the subarray $A[1..j-1]$ was sorted by the inductive hypothesis and hence the element at position $j+1$ is the largest element in the subarray $A[1..j+1]$. Furthermore, the inductive hypothesis states that when `PutInPlace`$(A, j-1, x)$ terminates, the subarray $A[1..j]$ is sorted and contains all elements for $A[1..j]$. Hence when, `PutInPlace`(A, j, x) terminates, subarray $A[1..j+1]$ is sorted and contains all required elements and hence the claim holds for j. Now the conclusion follows from the basis step, the inductive step, and the principle of induction.

- **Claim:** For any array A and natural number n such that $n \geq 0$ and array A has n cells, when `InsertionSort`(A, n) terminates, array A contains all the elements that were originally in A in sorted order. Proof by induction.

  Basis step: The base cases are:

  - $n = 0$: the claim is true as `InsertionSort`(A, n) terminates right away and the array A contains all original elements and is sorted since it is empty.
  - $n = 1$: the claim is true as `InsertionSort`(A, n) terminates right away and the array A contains all original elements and is sorted since it only contains one element.

  Inductive step: Let $n > 1$ be an arbitrary integer and assuming the claim holds for $n-1$ we show it also holds for n.

  Since $n > 1$ by the inductive hypothesis, we enter the branch. By the inductive hypothesis, when the call to `InsertionSort`$(A, n-1)$ terminates, the subarray $A[1..n-1]$ is sorted and contains all original elements up to $n-1$. We then store the element at cell n into variable

x. We then call `PutInPlace`$(A, n-1, x)$. As we proved previously, subarray $A[1..n]$ will be sorted and contain all original elements when `PutInPlace`$(A, n-1, x)$ terminates. After this, `InsertionSort`(A, n) terminates and $A[1..n]$ will be sorted and contain all original elements. Hence the claim holds for n. Now the conclusion follows from the basis step, the inductive step, and the principle of induction.

b.

During the sort, the following intermediate results occur:

1. $[4, 7, 2, 8, 6, 5]$
2. $[2, 4, 7, 8, 6, 5]$
3. $[2, 4, 6, 7, 8, 5]$
4. $[2, 4, 5, 6, 7, 8]$

Hence, the following occur: ii, iv, and vii.

Problem 3.

i

The procedure returns 26.

ii

At the start of each iteration i ($1 \leq i \leq n$), we have

- when $i = 1$, the value of p is $(A[1] + 1)$;
- when $i = 2$, the value of p is $(A[1] + 1) + (A[2] + 2)$;
- when $i = 3$, the value of p is $(A[1] + 1) + (A[2] + 2) + (A[3] + 3)$;
- $\ldots$
- when $i = k$ for any k such that $1 \leq k < n$, the value of p is $(A[1] + 1) + \ldots + (A[k] + k)$;
- when $i = n$, the loop terminates.

So, the loop invariant is: at the beginning of each iteration,

$$p = \sum_{k=1}^{i} (A[k] + k).$$

iii

- Initially: before the loop begins, $p = A[1] + 1 = \sum_{k=1}^{1} (A[k] + k)$.
- Maintenance: suppose that at the beginning of iteration i, $p = \sum_{k=1}^{i} (A[k] + k)$. Then, at the beginning of iteration $i + 1$,

$$\begin{aligned} p^{\text{after}} &= p^{\text{before}} + (A[j] + j) \\ &= p^{\text{before}} + (A[i+1] + i + 1) \quad (\text{since } j = i + 1) \\ &= \sum_{k=1}^{i} (A[k] + k) + (A[i+1] + i + 1) \quad (\text{the LI}) \\ &= \sum_{k=1}^{i+1} (A[k] + k) \end{aligned}$$

$$= \sum_{k=1}^{i^{\text{after}}} (A[k] + k).$$

- Termination #1: the loop terminates as we only increment i, so eventually we will have $i \geq n$.
- Termination #2: when the loop terminates, $i = n$, in which case the loop invariant implies $p = \sum_{k=1}^{i}(A[k] + k)$. The procedure sums up each element and the indices of each element and returns the result. This is clearly implied by the loop invariant: for each index in the array, the loop adds the index and the element at that index to a running total.

Problem 4.

The algorithm psuedocode is provided below:

```
procedure FindMinMax(A, n)
    min ← A[1]
    max ← A[1]
    i ← 2
    if (n%2 = 0) then
        if (A[1] < A[2]) then
            max = A[2]
        else
            min = A[1]
        end if
        i ← 3
    end if
    while (i < n) do
        if (A[i] < A[i + 1]) then
            if (A[i] < min) then
                min = A[i]
            end if
            if (A[i + 1] > max) then
                max = A[i + 1]
            end if
        else
            if (A[i] > max) then
                max = A[i]
            end if
            if (A[i + 1] < min) then
                min = A[i + 1]
            end if
        end if
        i ← i + 2
    return min, max
```

The algorithm always uses less than $3n/2$ key comparisons. We have two cases:

- When n is even, we enter the first `if then` statement where we perform one key comparison between $A[1]$ and $A[2]$ thus handling the first two elements of A. We then enter the `while` loop where each iteration handles a pair of elements. Since we handled the first pair of elements in the `if then` statement before the `while` loop, we have to handle $(n/2-1)$ more pairs. As we can see, there are three key comparisons in all scenarios for each pair of elements, so the number of key comparisons completed inside the `while` loop is $3 \cdot (n/2-1)$. As such, the total number of key comparisons when n is even is $1 + 3 \cdot (n/2-1) = 3n/2-2$.

- When n is odd, we skip the `if then` statement and proceed straight to the `while` loop. Notice we handle the first element in A, leaving us with an even number of remaining elements that can be paired up. Hence, inside the `while` loop, we will perform comparisons of $(n-1)/2$ pairs of elements. As with the case where n is even, each pair of elements requires three key comparisons to process. As such, the number of key comparisons when n is odd is $3 \cdot (n-1)/2$.

Hence, for all cases, there are less than $3n/2$ key comparisons.

We next derive a loop invariant. We define the loop invariant to be: at the beginning of each iteration i $(2 \leq i < n)$, $min = \min\{A[1], A[2], \ldots, A[i-1]\}$ and $max = \max\{A[1], A[2], \ldots, A[i-1]\}$. In other words, min and max contain the minimum and maximum values of subarray $A[1..i-1]$.

- Initially: before the loop begins:
  - If n is even, we enter the `if else` statement and set min to $\min\{A[1], A[2]\}$ and max to $\max\{A[1], A[2]\}$. Initially, $i = 3$ in this case. We can see that min and max contain the minimum and maximum values of subarray $A[1..i-1] = A[1..2]$.
  - If n is odd, we simply have that $min = A[1]$ and $max = A[1]$. Initially, $i = 2$ in this case. We can see that min and max contain the minimum and maximum values of subarray $A[1..i-1] = A[1..1]$.
- Maintenance: assume the LI holds at the start of iteration i for each $i \geq 2$. After the code in the loop is executed, min holds the smallest value between min, $A[i]$, and $A[i+1]$. Furthermore, max holds the largest value between max, $A[i]$, and $A[i+1]$.

  By the assumption that min holds the smallest of the first $i-1$ elements at the beginning of iteration i, at the start of the next iteration, $min = \min\{A[1], A[2], \ldots, A[i^{\text{new}}-1]\}$ where $i^{\text{new}} = i+2$. Furthermore, by the assumption that max holds the largest of the first $i-1$ elements at the beginning of iteration i, at the start of the next iteration, $max = \max\{A[1], A[2], \ldots, A[i^{\text{new}}-1]\}$ where $i^{\text{new}} = i+2$. This shows that the LI holds again.
- Termination #1: the loop terminates as we only increment i, so eventually we will have $i \geq n$.
- Termination #2: when the loop terminates, whether n is even or odd, $i = n+1$ and the LI implies that $min = \min\{A[1], A[2], \ldots, A[i-1]\} = \min\{A[1], A[2], \ldots, A[n]\}$ and $max = \max\{A[1], A[2], \ldots, A[i-1]\} = \max\{A[1], A[2], \ldots, A[n]\}$.