COMP 2804 — Solutions Assignment 2

Question 1:

- Write your name and student number.

Solution:

- Name: Johan Cruijff
- Student number: 14

Question 2: The function $f : \mathbb{N} \to \mathbb{N}$ is defined by

$$\begin{aligned} f(0) &= 1, \\ f(n) &= 7 \cdot f(n-1) + (2n-1) \cdot 7^{n-1} \quad \text{if } n \geq 1. \end{aligned}$$

- Prove that for every integer $n \geq 0$,

$$f(n) = \left(n^2 + 7\right) \cdot 7^{n-1}.$$

Solution: The proof is by induction on n. The base case is when $n = 0$. For this case, the left-hand side is $f(0)$, which is 1, and the right-hand side is $(0^2 + 7) \cdot 7^{0-1}$, which is also 1. Thus, the base case holds.

For the induction step, let $n \geq 1$ be an integer and assume the claim is true for $n - 1$. In other words, we assume that

$$f(n-1) = \left((n-1)^2 + 7\right) \cdot 7^{n-2}.$$

We now prove that the claim is also true for n. For this, we use the recurrence, the assumption, and some algebra:

$$\begin{aligned} f(n) &= 7 \cdot f(n-1) + (2n-1) \cdot 7^{n-1} \\ &= 7\left(\left((n-1)^2 + 7\right) \cdot 7^{n-2}\right) + (2n-1) \cdot 7^{n-1} \\ &= \left((n-1)^2 + 7 + (2n-1)\right) \cdot 7^{n-1} \\ &= \left(n^2 - 2n + 1 + 7 + 2n - 1\right) \cdot 7^{n-1} \\ &= \left(n^2 + 7\right) \cdot 7^{n-1}. \end{aligned}$$

This proves the induction step.

Question 3: The function $f : \mathbb{N} \to \mathbb{N}$ is defined by

$$\begin{aligned} f(0) &= 0, \\ f(n) &= f(n-1) + 3 \cdot (f(n-1))^{2/3} + 3 \cdot (f(n-1))^{1/3} + 1 \quad \text{if } n \geq 1. \end{aligned}$$

- Solve this recurrence, i.e., express $f(n)$ in terms of n only.

Solution: The solution will be of the type "guess the answer and verify by induction".

There are two methods to guess the answer:

1. Use the recurrence to compute $f(n)$ for some small values of n: $f(0) = 0$, $f(1) = 1$, $f(2) = 8$, $f(3) = 27$. From this, it looks like $f(n) = n^3$.

2. The recurrence looks very similar to Question 6 in Assignment 1. Based on this, we guess that $f(n) = n^3$.

Now that we think that for all $n \geq 0$,

$$f(n) = n^3,$$

we are going to verify this by induction.

The base case is when $n = 0$. For this case, the left-hand side is $f(0)$, which is 0, and the right-hand side is 0^3, which is also 0. Thus, the base case holds.

For the induction step, let $n \geq 1$ be an integer and assume the claim is true for $n - 1$. In other words, we assume that

$$f(n-1) = (n-1)^3.$$

We now prove that the claim is also true for n. For this, we use the recurrence, the assumption, and some algebra:

$$\begin{aligned} f(n) &= f(n-1) + 3 \cdot (f(n-1))^{2/3} + 3 \cdot (f(n-1))^{1/3} + 1 \\ &= (n-1)^3 + 3 \cdot \left((n-1)^3\right)^{2/3} + 3 \cdot \left((n-1)^3\right)^{1/3} + 1 \\ &= (n-1)^3 + 3 \cdot (n-1)^2 + 3 \cdot (n-1) + 1. \end{aligned}$$

We have seen in Question 6 in Assignment 1 that the right-hand side is equal to n^3. Therefore, we have $f(n) = n^3$. This proves the induction step.

Question 4: Let $n \geq 1$ be an integer and consider the set $S = \{1, 2, \ldots, n\}$.

- Assume we arrange the elements of S in sorted order on a horizontal line. Let B_n be the number of subsets of S that do not contain any two elements that are neighbors on this line. For example, if $n = 4$, then both subsets $\{1, 3\}$ and $\{1, 4\}$ are counted in B_4, but neither of the subsets $\{2, 3\}$ and $\{2, 3, 4\}$ is counted.

  For each integer $n \geq 1$, express B_n in terms of numbers that we have seen in class.

- Assume we arrange the elements of S in sorted order along a circle. Let C_n be the number of subsets of S that do not contain any two elements that are neighbors on this circle. For example, if $n = 4$, then the subset $\{1, 3\}$ is counted in C_4, but neither of the subsets $\{2, 3\}$ and $\{1, 4\}$ is counted.

  For each integer $n \geq 4$, express C_n in terms of numbers that we have seen in class.

Solution:

First part:

1. We have seen in class that the number of 00-free bitstrings of length n is equal to f_{n+2}, which is the Fibonacci number with index $n+2$.
2. By swapping the roles of 0 and 1, we see that the number of 11-free bitstrings of length n is also equal to f_{n+2}.
3. We have seen in class that any subset of S can be encoded as a bitstring of length n.
4. The condition that the subset cannot contain two elements that are neighbors on the line means that the bitstring is 11-free.
5. We conclude that $B_n = f_{n+2}$.

Second part:

1. As above, we are going to encode subsets as binary strings of length n.
2. The condition that the subset cannot contain two elements that are neighbors on the circle means that the bitstring is 11-free and, additionally, the first and last bits cannot both be 1.
3. We divide all these bitstrings into two groups:
   (a) Group 1: The first bit is 0.
   By removing this first bit, we get all 11-free bitstrings of length $n-1$. The number of strings in this group is f_{n+1}.
   (b) Group 2: The first bit is 1.
   The second bit must be 0 and the last bit must be 0 as well. By removing the first two bits and the last bit, we get all 11-free bitstrings of length $n-3$. The number of strings in this group is f_{n-1}.
4. We conclude that $C_n = f_{n+1} + f_{n-1}$.

Question 5: In class, we have seen algorithm EUCLID(a, b), which takes as input two integers a and b with $a \geq b \geq 1$, and returns their greatest common divisior.

- Assume we run this algorithm with two input integers a and b that satisfy $b > a \geq 1$. What is the output of this algorithm? As always, justify your answer.

Solution: As we have seen in class, the algorithm is as follows:

Algorithm EUCLID(a, b)**:**

// a and b are integers with $a \geq b \geq 1$
$r = a \bmod b$;
if $r = 0$
then return b
else EUCLID(b, r)
    // observe that $b > r \geq 1$
endif

Assume that $b > a \geq 1$. Let us see what happens if we run EUCLID(a, b):

1. The first line computes $a \bmod b$ and stores the result in r. Since $a < b$, we have $a \bmod b = a$. Thus, $r = a$.
2. Since $a \geq 1$, we have $r \neq 0$.
3. Thus, there is a recursive call EUCLID(b, r), which is the same as EUCLID(b, a).
4. Since $b > a \geq 1$, we have seen in class that EUCLID(b, a) returns $gcd(b, a)$, which is equal to $gcd(a, b)$.
5. We conclude that EUCLID(a, b) returns $gcd(a, b)$.

Question 6: The Fibonacci numbers are defined by

$$\begin{aligned} f_0 &= 0, \\ f_1 &= 1, \\ f_n &= f_{n-1} + f_{n-2}, \text{ if } n \geq 2. \end{aligned}$$

The goal of this exercise is to prove that there exists a Fibonacci number whose 2018 rightmost digits (when written in decimal notation) are all zero.

In the rest of this exercise, N denotes the number 10^{4036}. For any integer $n \geq 0$, define

$$g_n = f_n \bmod 10^{2018}.$$

- Consider the ordered pairs (g_n, g_{n+1}), for $n = 0, 1, 2, \ldots, N$. Use the Pigeonhole Principle to prove that these ordered pairs cannot all be distinct. That is, prove that there exist integers $m \geq 0$, $p \geq 1$, such that $m + p \leq N$ and

$$(g_m, g_{m+1}) = (g_{m+p}, g_{m+p+1}).$$

- Prove that $(g_{m-1}, g_m) = (g_{m+p-1}, g_{m+p})$.
- Prove that $(g_0, g_1) = (g_p, g_{p+1})$.

- Consider the decimal representation of f_p. Prove that the 2018 rightmost digits of f_p are all zero.

Solution:

First part:

1. The number of ordered pairs (g_n, g_{n+1}) is equal to $N + 1$.
2. For each such pair, the first "coordinate" is an element of $\{0, 1, 2, , \ldots, \sqrt{N}-1\}$. Thus, there are $\sqrt{N}$ possible values for the first coordinate.
3. For each such pair, the second "coordinate" is an element of $\{0, 1, 2, , \ldots, \sqrt{N} - 1\}$. Thus, there are $\sqrt{N}$ possible values for the second coordinate.
4. The total number of different pairs is equal to $\sqrt{N} \times \sqrt{N} = N$.
5. We conclude that there are more pairs than posssible different pairs. By the Pigeonhole Principle, the sequence (g_n, g_{n+1}), for $n = 0, 1, 2, \ldots, N$, contains duplicates.
6. If we use m and $m + p$ to denote the indices of two duplicates, then $(g_m, g_{m+1}) = (g_{m+p}, g_{m+p+1})$.

Second part:

1. We first look at the Fibonacci numbers:
   (a) If we know f_{n-1} and f_{n-2}, then we can obtain the value f_n: It is equal to $f_{n-1} + f_{n-2}$. In other words, this allows us to obtain the *next* number in the sequence.
   (b) If we know f_n and f_{n-1}, then we can obtain the value f_{n-2}: It is equal to $f_n - f_{n-1}$. In other words, this allows us to obtain the *previous* number in the sequence.
   (c) Since $g_n = f_n \bmod \sqrt{N}$, the same is true for the numbers in the sequence $g_0, g_1, g_2, \ldots$
2. We know from the first part that
$$g_m = g_{m+p}$$
and
$$g_{m+1} = g_{m+p+1}.$$
It follows that (all arithmetic is done modulo $\sqrt{N}$):
$$\begin{aligned} g_{m-1} &= g_{m+1} - g_m \\ &= g_{m+p+1} - g_{m+p} \\ &= g_{m+p-1}. \end{aligned}$$
3. Since $g_{m-1} = g_{m+p-1}$ and $g_m = g_{m+p}$, we conclude that $(g_{m-1}, g_m) = (g_{m+p-1}, g_{m+p})$.

Third part:

1. In the second part, we have shown the following:

   $$\text{Since } (g_m, g_{m+1}) = (g_{m+p}, g_{m+p+1}), \text{ we have } (g_{m-1}, g_m) = (g_{m+p-1}, g_{m+p}).$$

2. Of course, we can repeat this: We know that

   $$(g_{m-1}, g_m) = (g_{m+p-1}, g_{m+p}),$$

   from which we get

   $$(g_{m-2}, g_{m-1}) = (g_{m+p-2}, g_{m+p-1}),$$

   from which we get

   $$(g_{m-3}, g_{m-2}) = (g_{m+p-3}, g_{m+p-2}),$$

   from which we get

   $$(g_{m-4}, g_{m-3}) = (g_{m+p-4}, g_{m+p-3}),$$

   etc., etc. At the end, we conclude that

   $$(g_{m-m}, g_{m-m+1}) = (g_{m+p-m}, g_{m+p-m+1}),$$

   which is the same as

   $$(g_0, g_1) = (g_p, g_{p+1}).$$

Fourth part:

1. We have seen above that $g_p = g_0$.
2. We know that $g_0 = f_0 \bmod 10^{2018}$. Since $f_0 = 0$, we have

   $$g_0 = f_0 \bmod 10^{2018} = 0 \bmod 10^{2018} = 0.$$

3. We now know that $g_p = g_0 = 0$, which means that $f_p \bmod 10^{2018} = 0$, which means that f_p is divisible by 10^{2018}, which means that the 2018 rightmost digits of f_p are all zero.

Remark: Of course there is nothing magic about the number 2018: For every integer $k \geq 1$, there is a Fibonacci number whose k rightmost digits are all 0. To prove this, repeat this exercise with $g_n = f_n \bmod 10^k$ and $N = 10^{2k}$.

Question 7: In this exercise, we consider strings of characters, where each character is an element of $\{a, b, c\}$. Such a string is called *aa-free*, if it does not contain two consecutive a's. For any integer $n \geq 1$, let F_n be the number of *aa*-free strings of length n.

- Determine F_1, F_2, and F_3.

- Let $n \geq 3$ be an integer. Express F_n in terms of F_{n-1} and F_{n-2}.
- Prove that for every integer $n \geq 1$,

$$F_n = \left(\frac{1}{2} + \frac{1}{\sqrt{3}}\right)\left(1+\sqrt{3}\right)^n + \left(\frac{1}{2} - \frac{1}{\sqrt{3}}\right)\left(1-\sqrt{3}\right)^n.$$

Hint: What are the solutions of the equation $x^2 = 2x + 2$? Using these solutions will simplify the proof.

Solution:

First part:

1. To determine F_1: There are 3 possible strings of length 1; all of them are aa-free. Therefore, $F_1 = 3$.
2. To determine F_2: There are $3^2 = 9$ possible strings of length 2. Among these, the string aa is not aa-free. Therefore, $F_2 = 9 - 1 = 8$.
3. To determine F_3: There are $3^3 = 27$ possible strings of length 2. Here are the 5 strings of length 3 that are not aa-free:

   $$aaa, aab, aac, baa, caa.$$

   Therefore, $F_3 = 27 - 5 = 22$.

Second part: Let $n \geq 3$. By definition, the number of aa-free strings of length n is equal to F_n. We are going to divide all these strings into three groups, based on the first character:

1. Group 1: Strings that start with b.

   If we remove the first character from all these strings, then we obtain all aa-free strings of length $n-1$. Therefore, this group consists of F_{n-1} many strings.
2. Group 2: Strings that start with c.

   If we remove the first character from all these strings, then we obtain all aa-free strings of length $n-1$. Therefore, this group consists of F_{n-1} many strings.
3. Group 3: Strings that start with a.

   We subdivide this group into two subgroups, based on the second character (this second character cannot be a):

   (a) Group 3.1: The second character is b.

   If we remove the first two characters from all these strings, then we obtain all aa-free strings of length $n-2$. Therefore, this group consists of F_{n-2} many strings.

(b) Group 3.2: The second character is c.

If we remove the first two characters from all these strings, then we obtain all aa-free strings of length $n-2$. Therefore, this group consists of F_{n-2} many strings.

Based on this way of counting, the number of aa-free strings of length n is equal to

$$2 \cdot F_{n-1} + 2 \cdot F_{n-2}.$$

We conclude that $F_n = 2 \cdot F_{n-1} + 2 \cdot F_{n-2}$.

As a sanity check, we have seen that $F_1 = 3$, $F_2 = 8$, and $F_3 = 22$. We see that $F_3 = 2 \cdot F_2 + 2 \cdot F_1$. (If this were not the case, we would have made a mistake!)

Third part: We have obtained the following recurrence:

$$\begin{aligned} F_1 &= 3, \\ F_2 &= 8, \\ F_n &= 2 \cdot F_{n-1} + 2 \cdot F_{n-2}, \text{ if } n \geq 3. \end{aligned}$$

You learned in high school that the equation $x^2 = 2x + 2$ has two solutions: $\alpha = 1 + \sqrt{3}$ and $\beta = 1 - \sqrt{3}$. Thus, we have to prove that for all $n \geq 1$,

$$F_n = \left(\frac{1}{2} + \frac{1}{\sqrt{3}}\right) \alpha^n + \left(\frac{1}{2} - \frac{1}{\sqrt{3}}\right) \beta^n.$$

We are going to prove this by induction.

The first base case is when $n = 1$. The left-hand side is F_1, which is 3. The right-hand side is

$$\left(\frac{1}{2} + \frac{1}{\sqrt{3}}\right) \alpha + \left(\frac{1}{2} - \frac{1}{\sqrt{3}}\right) \beta,$$

which is also 3. This proves the first base case.

The second base case is when $n = 2$. The left-hand side is F_2, which is 8. The right-hand side is

$$\left(\frac{1}{2} + \frac{1}{\sqrt{3}}\right) \alpha^2 + \left(\frac{1}{2} - \frac{1}{\sqrt{3}}\right) \beta^2,$$

which is also 8. This proves the second base case.

For the induction step, let $n \geq 3$ and assume that the claim is true for $n-1$ and $n-2$. Thus, we assume that

$$F_{n-1} = \left(\frac{1}{2} + \frac{1}{\sqrt{3}}\right) \alpha^{n-1} + \left(\frac{1}{2} - \frac{1}{\sqrt{3}}\right) \beta^{n-1}$$

and

$$F_{n-2} = \left(\frac{1}{2} + \frac{1}{\sqrt{3}}\right) \alpha^{n-2} + \left(\frac{1}{2} - \frac{1}{\sqrt{3}}\right) \beta^{n-2}.$$

Using the recurrence, these two assumptions, and the facts that $\alpha^2 = 2\alpha+2$ and $\beta^2 = 2\beta+2$, we get

$$\begin{aligned}
F_n &= 2 \cdot F_{n-1} + 2 \cdot F_{n-2} \\
&= 2 \cdot \left(\left(\frac{1}{2} + \frac{1}{\sqrt{3}} \right) \alpha^{n-1} \right) + 2 \cdot \left(\left(\frac{1}{2} - \frac{1}{\sqrt{3}} \right) \beta^{n-1} \right) + \\
&\quad 2 \cdot \left(\left(\frac{1}{2} + \frac{1}{\sqrt{3}} \right) \alpha^{n-2} \right) + 2 \cdot \left(\left(\frac{1}{2} - \frac{1}{\sqrt{3}} \right) \beta^{n-2} \right) \\
&= \left(\frac{1}{2} + \frac{1}{\sqrt{3}} \right) \alpha^{n-2} (2\alpha + 2) + \left(\frac{1}{2} - \frac{1}{\sqrt{3}} \right) \beta^{n-2} (2\beta + 2) \\
&= \left(\frac{1}{2} + \frac{1}{\sqrt{3}} \right) \alpha^{n-2} \cdot \alpha^2 + \left(\frac{1}{2} - \frac{1}{\sqrt{3}} \right) \beta^{n-2} \cdot \beta^2 \\
&= \left(\frac{1}{2} + \frac{1}{\sqrt{3}} \right) \alpha^{n} + \left(\frac{1}{2} - \frac{1}{\sqrt{3}} \right) \beta^{n}.
\end{aligned}$$

This proves the induction step.

Question 8: Let $m \geq 1$ and $n \geq 1$ be integers and consider an $m \times n$ matrix A. The rows of this matrix are numbered $1, 2, \ldots, m$, and its columns are numbered $1, 2, \ldots, n$. Each entry of A stores one number and, for each row, all numbers in this row are pairwise distinct. For each $i = 1, 2, \ldots, m$, define

$$g(i) = \text{ the position (i.e., column number) of the smallest number in row } i.$$

We say that the matrix A is *awesome*, if

$$g(1) \leq g(2) \leq g(3) \leq \ldots \leq g(m).$$

In the matrix below, the smallest number in each row is in boldface. For this example, we have $m = 4$, $n = 10$, $g(1) = 3$, $g(2) = 3$, $g(3) = 5$, and $g(4) = 8$. Thus, this matrix is awesome.

$$A = \begin{pmatrix}
13 & 12 & \mathbf{5} & 8 & 6 & 9 & 15 & 20 & 19 & 7 \\
3 & 4 & \mathbf{1} & 17 & 6 & 13 & 7 & 10 & 2 & 5 \\
19 & 5 & 12 & 7 & \mathbf{2} & 4 & 11 & 13 & 6 & 3 \\
7 & 4 & 17 & 10 & 5 & 14 & 12 & \mathbf{3} & 20 & 6
\end{pmatrix}.$$

From now on, we assume that the $m \times n$ matrix A is awesome.

- Let i be an integer with $1 \leq i \leq m$. Describe, in plain English and a few sentences, an algorithm that computes $g(i)$ in $O(n)$ time.
- Describe, in plain English and a few sentences, an algorithm that computes all values $g(1), g(2), \ldots, g(m)$ in $O(mn)$ total time.

In the rest of this exercise, you will show that all values $g(1), g(2), \ldots, g(m)$ can be computed in $O(m + n \log m)$ total time.

- Assume that m is even and assume that you are given the values

$$g(2), g(4), g(6), g(8), \ldots, g(m).$$

Describe, in plain English and using one or more figures, an algorithm that computes the values

$$g(1), g(3), g(5), g(7), \ldots, g(m-1)$$

in $O(m+n)$ total time.

- Assume that $m = 2^k$, i.e., m is a power of two. Describe a recursive algorithm FindMinima that has the following specification:

Algorithm FindMinima(A, i)**:**
Input: An $m \times n$ awesome matrix A and an integer i with $0 \leq i \leq k$.
Output: The values $g\left(j \cdot m/2^i\right)$ for $j = 1, 2, 3, \ldots, 2^i$.

For each i with $0 \leq i \leq k$, let $T(i)$ denote the running time of algorithm FindMinima(A, i). The running time of your algorithm must satisfy the recurrence

$$\begin{aligned} T(0) &= O(n), \\ T(i) &= T(i-1) + O\left(2^i + n\right), \text{ if } 1 \leq i \leq k. \end{aligned}$$

You may use plain English and figures to describe your algorithm, but it must be clear how you use recursion.

- Assume again that $m = 2^k$. Prove that all values $g(1), g(2), \ldots, g(m)$ can be computed in $O(m + n \log m)$ total time.

Hint: $1 + 2 + 2^2 + 2^3 + \cdots + 2^k \leq 2m$.

Solution: This exercise may look difficult. However, if you carefully read, and understand, the different parts, then you will see that there are many hints!

First part: The algorithm should be obvious: Walk along the elements in row i, and determine the smallest element in this row. The position of this smallest element is equal to $g(i)$. The running time is obviously $O(n)$.

Second part: This algorithm should also be obvious: We repeat the first part once for every $i = 1, 2, \ldots, m$, i.e., once for every row. The running time is $m \cdot O(n) = O(mn)$.

Third part: The matrix has m rows and n columns. The smallest elements in all rows form a "staircase": If you are at the smallest element in row i, and move down to row $i+1$, then the smallest element in row $i+1$ is to the right or at the same position.

We are given the positions of the smallest elements in the even rows. We want the positions of the smallest elements in the odd rows.

The smallest element in row 1 is at one of the positions $1, 2, \ldots, g(2)$. We find this smallest element by scanning the first $g(2)$ positions in row 1. Thus, we scan $g(2)$ many entries in row 1 and find $g(1)$ in time that is proportional to $g(2)$.

The smallest element in row 3 is at one of the positions $g(2), \ldots, g(4)$. We find this smallest element by scanning these positions in row 3. Thus, we scan $g(4) - g(2) + 1$ many entries in row 3 and find $g(3)$ in time that is proportional to $g(4) - g(2) + 1$.

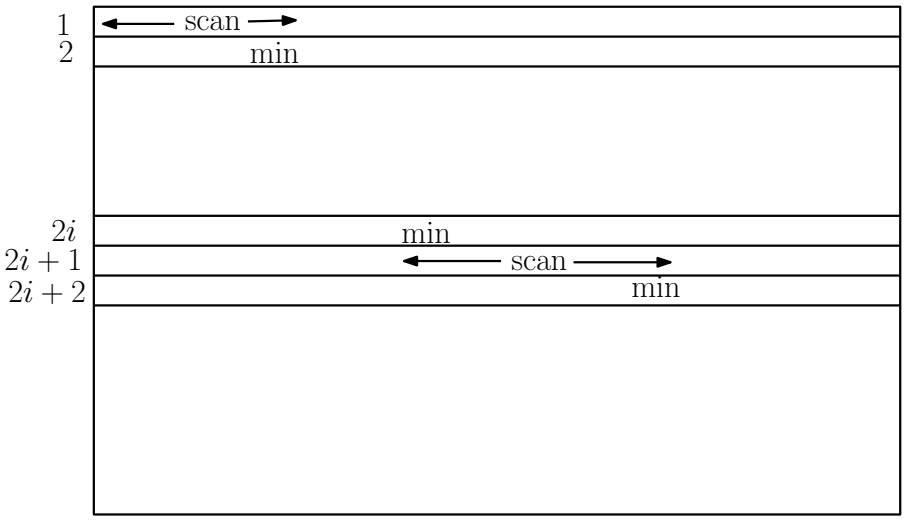
The smallest element in row 5 is at one of the positions $g(4), \ldots, g(6)$. We find this smallest element by scanning these positions in row 5. Thus, we scan $g(6) - g(4) + 1$ many entries in row 5 and find $g(5)$ in time that is proportional to $g(6) - g(4) + 1$.

Etc., etc.

Here is a more formal way of describing this process:

1. Compute $g(1)$ to be the position of the smallest element among the first $g(2)$ positions in row 1.

2. For $i = 1, 2, \ldots, \frac{m}{2} - 1$: Compute $g(2i+1)$ to be the position of the smallest element among the positions $g(2i), \ldots, g(2i+2)$ in row $2i+1$.



The total running time is proportional to

$$g(2) + \sum_{i=1}^{\frac{m}{2}-1} \left(g(2i+2) - g(2i) + 1 \right),$$

which is the sum of

$$g(2), g(4) - g(2) + 1, g(6) - g(4) + 1, g(8) - g(6) + 1, \ldots, g(m) - g(m-2) + 1,$$

which is

$$g(m) + \left(\frac{m}{2} - 1\right).$$

Since $g(m) \leq n$, the total running time is at most

$$n + \left(\frac{m}{2} - 1\right) \leq n + m = O(m + n).$$

Intermezzo: Before we go to the fourth part, here is some intuition of what we are going to do:

1. Our goal is to compute all values $g(1), ..., g(m)$.
2. From the third part: If we know $g(2), g(4), g(6), \ldots$, then we can compute $g(1), g(3), g(5), \ldots$, in $O(m + n)$ time.
3. How do we know $g(2), g(4), g(6), \ldots$? By the same idea: If we know $g(4), g(8), g(12), \ldots$, then we can compute $g(2), g(6), g(10), \ldots$, in total time $O(m/2 + n)$.
4. How do we know $g(4), g(8), g(12), \ldots$? By the same idea: If we know $g(4), g(12), g(20), \ldots$, then we can compute $g(8), g(16), g(24), \ldots$,, in total time $O(m/4 + n)$.
5. You will see the pattern. At the end, we get:
6. How do we know $g(m/2), g(m)$? By the same idea: If we know $g(m)$, then we can compute $g(m/2)$ in time $O(n)$.
7. How do we know $g(m)$? For this, we use the first part of the question.

The purpose of the fourth part of the question is to describe this whole process using recursion.

Fourth part:

1. The algorithm is called FindMinima(A, i) and is recursive. So there must be a base case. From the recurrence for $T(i)$ in the question, it should be clear that the base case is when $i = 0$.
2. From the specification of the algorithm, what is FindMinima$(A, 0)$ supposed to return: Since $i = 0$, it returns only one value: $g(m)$. So in the base case, we scan row m, and return the position of the smallest number in this row.
3. What if $i \geq 1$:
   (a) From the recurrence for $T(i)$, it should be clear that there is one recursive call, namely FindMinima$(A, i - 1)$.
   (b) Let us see what FindMinima(A, i) is supposed to return: I write L for $m/2^i$.

(c) FindMinima(A, i) should return the g-values for all rows that are multiples of L.

(d) If we know the g-values for all rows that are even multiples of L, then we can use the approach in the third part of the question to obtain the g-values for all rows that are odd multiples of L, in total time $O(2^i + n)$.

(e) The g-values for all rows that are even multiples of L are exactly the g-values for all rows that are multiples of $2L$. Since $2L = m/2^{i-1}$, we obtain these values by calling FindMinima$(A, i-1)$.

In summary, here is what FindMinima(A, i) does:

Algorithm FindMinima(A, i)**:**
If $i = 0$: Determine the smallest number in row m. The position of this smallest number gives us the value $g(m)$.
If $i \geq 1$: Run FindMinima$(A, i-1)$. After this has terminated, we know the g-values for all rows that are even multiples of $m/2^i$. Now use the approach of the third part to compute the g-values for all rows that are odd multiples of $m/2^i$.

Fifth part: To obtain all values $g(1), \ldots, g(m)$, we run FindMinima(A, k). The total time for this is equal to $T(k)$.

We are going to use unfolding to solve the recurrence. There are two big-O's in the recurrence. We assume that the constants in both of them are equal to 1. (Why are we allowed to do this?)

$$\begin{aligned}
T(k) &= \big(2^k + n\big) + T(k-1) \\
&= \big(2^k + n\big) + \big(2^{k-1} + n\big) + T(k-2) \\
&= \big(2^k + 2^{k-1}\big) + 2n + T(k-2) \\
&= \big(2^k + 2^{k-1}\big) + 2n + \big(2^{k-2} + n\big) + T(k-3) \\
&= \big(2^k + 2^{k-1} + 2^{k-2}\big) + 3n + T(k-3) \\
&\vdots \qquad\qquad\qquad \vdots \\
&= \big(2^k + 2^{k-1} + 2^{k-2} + ... + 2\big) + kn + T(0) \\
&= \big(2^k + 2^{k-1} + 2^{k-2} + ... + 2\big) + kn + n.
\end{aligned}$$

According to the hint, $2^k + 2^{k-1} + 2^{k-2} + ... + 2 \leq 2m$. Therefore,

$$T(k) \leq 2m + kn + n = 2m + n(1+k).$$

Since $k = \log m$, we conclude that

$$T(k) \leq 2m + n(1 + \log m) = O(m + n \log m).$$