Due Friday 7 April, *before* 1:00pm

Note: solutions may be incomplete, and meant to be used as guidelines only. We encourage you to ask follow-up questions on the course forum or during office hours.

1. [7 marks] Nested loops.

Consider the following algorithm. (It doesn't **return** anything useful, but it sure spends a bunch of time before returning!)

```python
def loop_de_loop_de_loop(n: int) -> None:
    """ Precondition: n > 0 """
    i = 0
    s = 1
    while i < n:              # Loop 1
        i = i + s
        s = s + 2
        j = 0
        while j < i:          # Loop 2
            k = n
            while k > 1:      # Loop 3
                k = k // 2
            j = j + 3
```

(a) **[3 marks]** Let s_t be the value of variable `s` and i_t be the value of variable `i` immediately *after* t iterations of Loop 1 have occurred. Determine a general formula for each of s_t and i_t and then find the exact number of iterations of Loop 1 for a given value of `n`.

Solution

Let's start by constructing a trace table that shows the values of `t`, $\texttt{i}_\texttt{t}$ and $\texttt{s}_\texttt{t}$ after `t` iterations of Loop 1.

We have:

t	$\texttt{i}_\texttt{t}$	$\texttt{s}_\texttt{t}$
0	0	1
1	1	3
2	4	5
3	9	7
4	16	9
5	25	11

and so on.

In the code, we see that `s` starts at `1` and is incremented by `2` on each iteration of Loop 1. We can write $\texttt{s}_\texttt{t} = 2\texttt{t} + 1$. That is, after `t` iterations, $\texttt{s}_\texttt{t}$ is the $\texttt{(t+1)}^{\text{st}}$ odd natural number. This is consistent with the table given above.

In the code, we also see that `i` starts at `0` and is incremented by $\texttt{s}_\texttt{t-1}$ on iteration `t` of Loop 1. We can write

$$\begin{aligned}\mathtt{i_t} &= \sum_{a=0}^{t-1} \mathtt{s_a} \\ &= \sum_{a=0}^{t-1} \big(2\mathtt{a}+1\big).\end{aligned}$$

That is, after $\mathtt{t}$ iterations, $\mathtt{i_t}$ is the sum of the first $\mathtt{t}$ odd natural numbers. We know from Example 3.11 of the course notes (pgs. 74 - 75), that this sum is equal to $\mathtt{t}^2$, and so can write $\mathtt{i_t} = \mathtt{t}^2$. This is consistent with the table given above.

Loop 1 repeats until $\mathtt{i_t} \geq \mathtt{n}$. This first happens when $\mathtt{t}^2 \geq \mathtt{n}$ or when $\mathtt{t} \geq \sqrt{\mathtt{n}}$. We can conclude that there will be $\lceil \sqrt{\mathtt{n}} \rceil$ iterations of Loop 1.

(b) [**4 marks**] Give a Theta bound on the running time function $RT(n)$ for this algorithm. Show your work. (That is, explain how you obtained your answer and show your calculations).

Solution

Note: there may be some off-by-one errors in the counting due to crossing-up of $\lfloor\rfloor$ and $\lceil\rceil$, but that won't affect the final conclusion.

Starting with the inner-most loop, Loop 3, let $\mathtt{k_b}$ be the value of variable $\mathtt{k}$ after $\mathtt{b}$ iterations of the loop. Since $\mathtt{k}$ starts at $\mathtt{n}$, we have $\mathtt{k_0} = \mathtt{n}$. Since $\mathtt{k}$ is divided by 2 using integer division each iteration, we have $\mathtt{k_{b+1}} = \lfloor \mathtt{k_b}/2 \rfloor$. You can show that repeated flooring of floors can be combined and that $\mathtt{k_b} = \left\lfloor \frac{n}{2^b} \right\rfloor$. Loop 3 repeats until $\mathtt{k_b} \leq \mathtt{1}$. This first happens when $\left\lfloor \frac{n}{2^b} \right\rfloor \leq \mathtt{1}$ or when $\mathtt{b} \geq \lfloor \log_2 \mathtt{n} \rfloor$. We can conclude that there will be $\lfloor \log_2 \mathtt{n} \rfloor$ iterations of Loop 3. Since the loop body takes 1 step, the running time of Loop 3 is $\lfloor \log_2 \mathtt{n} \rfloor$ steps.

Moving on to Loop 2, we can see that on a fixed iteration of Loop 2, the body will take $1 + \lfloor \log_2 \mathtt{n} \rfloor$ steps. If we let $\mathtt{j_c}$ be the value of variable $\mathtt{j}$ after $\mathtt{c}$ iterations of the loop, we have $\mathtt{j_c} = 3\mathtt{c}$, since $\mathtt{j}$ starts at $\mathtt{0}$ and is incremented by $\mathtt{3}$ each iteration. Loop 2 repeats until $\mathtt{j_c} \geq \mathtt{i}$. This first happens when $3c \geq \mathtt{i}$ and so there are $\left\lceil \frac{\mathtt{i}}{3} \right\rceil$ iterations of Loop 2.

The total running time of Loops 2 and 3 together is then $\left\lceil \frac{\mathtt{i}}{3} \right\rceil (1 + \lfloor \log_2 \mathtt{n} \rfloor)$ steps.

Moving on to Loop 1, we can see that on iteration $\mathtt{t}$ of Loop 1, the body will take $1 + \left\lceil \frac{\mathtt{i_t}}{3} \right\rceil (1 + \lfloor \log_2 \mathtt{n} \rfloor)$ steps. (We can use $\mathtt{i_t}$ since $\mathtt{i}$ is incremennted before the Loops 2 and 3 are performed.) This happens for $\mathtt{i_t}$ running from $\mathtt{t} = 1$ to $\lceil \sqrt{\mathtt{n}} \rceil$.

The total number of steps is then

$$\begin{aligned}
&1 + \sum_{\mathtt{t}=1}^{\lceil\sqrt{\mathtt{n}}\rceil} \left(1 + \left\lceil \frac{\mathtt{i_t}}{3} \right\rceil (1 + \lfloor \log_2 \mathtt{n} \rfloor)\right) \\
=&1 + \lceil\sqrt{\mathtt{n}}\rceil + (1 + \lfloor \log_2 \mathtt{n} \rfloor) \sum_{\mathtt{t}=1}^{\lceil\sqrt{\mathtt{n}}\rceil} \left(\left\lceil \frac{\mathtt{i_t}}{3} \right\rceil \right) \\
=&1 + \lceil\sqrt{\mathtt{n}}\rceil + (1 + \lfloor \log_2 \mathtt{n} \rfloor) \sum_{\mathtt{t}=1}^{\lceil\sqrt{\mathtt{n}}\rceil} \left(\left\lceil \frac{\mathtt{t}^2}{3} \right\rceil \right)
\end{aligned}$$

Recalling the summation formula $\sum_{i=1}^{m} i^2 = \frac{m(m+1)(2m+1)}{6}$ and applying it with $m = \lceil\sqrt{\mathtt{n}}\rceil$, we can argue the running time is $\Theta\big((\mathtt{n})^{\frac{3}{2}} \log_2 \mathtt{n}\big)$.

(An argument that uses $\mathcal{O}$ (ignore the division by 3) and Ω (pull the division by 3 out of the ceiling term) would be more precise and end with the same conclusion.)

2. [8 marks] Worst-case analysis.
Consider the following algorithm.

```python
def algo(lst: list[int]) -> None:
    """ Precondition: len(lst) > 0 """
    n = len(lst)
    i = 0
    j = 0
    while i < n:
        if lst[i] % 3 == 0:
            j = j + i
            while j > 0:
                j = j // 2
            i = i + 2
        else:
            j = j + n
            i = i + 1
```

(a) [**4 marks**] Find, with proof, an **upper bound** on the **worst-case** running time of this algorithm. Show your work. For full marks, your upper bound must match the lower bound determined in the next part.

Solution

Let $n \in \mathbb{N}$, and consider running `algo` on an (arbitrary) input `lst` of length n.

In this algorithm, we first perform a constant-time operation (the 3 assignment statements) and then enter the outer-loop.

We claim that on each iteration of the outer loop i increases by at least 1.

This is true because whenever `lst[i]` is divisible by 3, i increases by exactly 2. When `lst[i]` is not divisible by 3, i increases by exactly 1. Since $2 > 1$, we can conclude that i increases by at least 1 each iteration.

Also, since i starts at 0 and ends when $i \geq n$, we know that at the start of each iteration of the outer-loop, i is at most n. The outer-loop will iterate at most n times.

In each iteration of the outer-loop body, we either take the else-branch and perform a constant-time operation or take the if-branch and perform an constant-time operation and the inner-loop.

In each iteration of the outer-loop body, j either increases by i or by n. We also know that when j is incremented, i is at most n. Since j starts at 0, and the outer-loop runs at most n times, we know that j is at most $n \times n$ or n^2.

In the inner-most loop, a constant-time operation is repeated. Loop variable j is divided by 2 on each iteration until j reaches zero. The number of iterations of the inner loop is at most $\lceil \log_2 j - 1 \rceil$. Since j is at most n^2, the running time of the inner-most loop (for a fixed iteration of the outer loop) is at most $\left\lceil \log_2 n^2 - 1 \right\rceil$ or $\lceil 2 \log_2 n \rceil$ steps.

Altogether then, the running-time is at most $1 + n\Big(\big(1 + \lceil 2 \log_2 n \rceil\big) + 1\Big) = n \lceil 2 \log_2 n \rceil + 2n + 1$ steps. We can conclude that $WC_{\texttt{algo}}(n) \in \mathcal{O}(n \log_2 n)$.

(b) [**4 marks**] Find, with proof, a **lower bound** on the **worst-case** running time of this algorithm. Show your work. For full marks, your lower bound must match the upper bound determined in the previous part.

Solution

We wish to prove that $WC_{\texttt{algo}}(n) \in \Omega(n \log_2 n)$.
To prove that $WC_{\texttt{algo}}(n) \in \Omega(n \log_2 n)$, we need to find an input family whose running time is $\Omega(n \log_2 n)$.
Let $n \in \mathbb{N}$.
The running time will be in $\Omega(n \log_2 n)$ if we can get the inner loop over j to be performed a number of times that depends on n, and have it start each time with a value of j that depends on n.
Let $\texttt{k} \in \mathbb{N}$ and start with $\texttt{k} = 0$.
Assuming that item `lst[k]` (for a valid non-negative index `k`) is not divisible by 3, `j` would be incremented by n and i would be incremented by 1. If `lst[k+1]` **were** divisible by 3, the inner loop **would** then be run starting with a value of `j` that is at least `n` (and so will perform at least $\lfloor \log_2 n \rfloor$ steps). Variable `j` would be reset to 0.
After running the inner loop, variable `i` is incremented by 2. If `lst[k+1+2]` is **not** divisible by 3, `j` would once again be incremented by n and i would be incremented by 1. If `lst[k+1+2+1]` **were** divisible by 3, the inner loop would once again perform at least $\lfloor \log_2 n \rfloor$ steps, and `j` would be reset to 0.
Repeating in this manner, the inner loop would perform at least $\lfloor \log_2 n \rfloor$ steps. The number of such repetitions would be at least $\left\lfloor \frac{n}{3} \right\rfloor$, since `i` is incremented by 1 and then 2 each time.
An input family of the form

$$[1, 3, 2, 1, 3, 2, 1, 3, 2, \ldots]$$

would produce the behaviour described above.
That is, the n^{th} member of the family ($n \in \mathbb{N}$), is a list `lst` where, `len(lst) == n` and for `i in range(n)`,

$$\begin{cases} \texttt{lst[i] = 1}, & \text{when } \texttt{i \% 3 = 0}, \\ \texttt{lst[i] = 3}, & \text{when } \texttt{i \% 3 = 1}, \\ \texttt{lst[i] = 2}, & \text{when } \texttt{i \% 3 = 2}. \end{cases}$$

(The entry `2` is arbitrary as the value does not affect the steps performed by the algorithm.)
Altogether then, for this input, the number steps performed is at least $\left\lfloor \frac{n}{3} \right\rfloor \lfloor \log_2 n \rfloor$ steps. We can conclude that $WC_{\texttt{algo}}(n) \in \Omega(n \log_2 n)$.

3. [10 marks] Average-case analysis.

Consider the following algorithm.

```python
def has_even(numbers: list[int]) -> bool:
    """Return whether numbers contains an even number.
    Precondition: len(lst) > 0
    """
    for number in numbers:
        if number % 2 == 0:
            return True
    return False
```

(a) [**1 mark**] Write a set of allowable inputs $\mathcal{I}_{\texttt{has\_even},n}$ for which $Avg_{\texttt{has\_even}}(n)$ is $\Theta(1)$. State briefly the reasoning used to arrive at your answer.

Solution

We note that the number of iterations of the `for`-loop depends on the location of the first even number in `numbers`. The running time of `has_even` will be constant independent of n when the first even number in `numbers` occurs at a fixed index i that is independent of n. When this is the case, the `for`-loop will always run exactly $i+1$ times.

To simplify, let's suppose that `numbers[0]` is even, and consider the set F_n ($n \in \mathbb{Z}^+$) of input lists `numbers` to `has_even` where `len(numbers) == n` and

$$\big(\forall \texttt{i} \in \texttt{range}(n), 1 \leq \texttt{numbers[i]} \leq n\big) \land \texttt{numbers[0] \% 2 == 0}.$$

For the reasons given above, $Avg_{\texttt{has\_even}}(n)$ is $\Theta(1)$ when $\mathcal{I}_{\texttt{has\_even},n} = F_n$.

Note that we need to limit the items in `numbers` and not allow all `int`s so that F_n is a finite set.

(b) [**1 mark**] Write a set of allowable inputs $\mathcal{I}_{\texttt{has\_even},n}$ for which $Avg_{\texttt{has\_even}}(n)$ is $\Theta(n)$. State briefly the reasoning used to arrive at your answer.

Solution

The running time of `has_even` will be depend on n when the first even number in `numbers` occurs at an index i that is depends on n. When this is the case, the `for`-loop will always run exactly $i+1$ times.

Let's suppose that `numbers[n-1]` is even, and consider the set L_n ($n \in \mathbb{Z}^+$) of input lists `numbers` to `has_even` where `len(numbers) == n` and

$$\begin{aligned} & \big(\forall \texttt{i} \in \texttt{range}(n), 1 \leq \texttt{numbers[i]} \leq n\big) \\ \land \quad & \big(\forall \texttt{i} \in \texttt{range}(n-1), \texttt{numbers[i]}\%2 == 1\big) \\ \land \quad & \texttt{numbers[n-1] \% 2 == 0}. \end{aligned}$$

For the reasons given above, $Avg_{\texttt{has\_even}}(n)$ is $\Theta(n)$ when $\mathcal{I}_{\texttt{has\_even},n} = L_n$.

(c) [**2 marks**] Define

$$S_n = \{\text{input lists } \texttt{numbers} \text{ to } \texttt{has\_even} \mid \forall \texttt{i} \in \texttt{range}(n), 1 \leq \texttt{numbers[i]} \leq n\}.$$

Note that an `int` value is allowed to be repeated in an element of S_n. For example,

$$S_2 = \{[1,1],[1,2],[2,1],[2,2]\}.$$

Consider the set of allowable inputs $\mathcal{I}_{\texttt{has\_even},n} = S_n$. Give an expression for $|\mathcal{I}_{\texttt{has\_even},n}|$. State briefly the reasoning used to arrive at your answer.

Solution

Each element of S_n is a distinct length n list of `int`s where each item is an `int` between 1 and n inclusive.

Since we have n choices for each item (an `int` between 1 and n inclusive) and n items in each list, the total number of distinct length n lists of `int`s is

$$\begin{aligned} & n \times n \times n \times \cdots \times n \qquad (n \text{ terms in the product}) \\ &= n^n. \end{aligned}$$

For this reason, $|\mathcal{I}_{\texttt{has\_even},n}| = n^n$.

(d) [**6 marks**] Calculate an *exact* expression for $Avg_{\texttt{has\_even}}(n)$ for the set of allowable inputs $\mathcal{I}_n = S_n$, where S_n is as defined in the previous part. Show your work.

Hint: You may use without proof any helpful correct formula that simplifies an expression containing a summation.

Solution

We can start with the observation that

$$\begin{aligned} Avg_{\texttt{has\_even}}(n) &= \frac{1}{|\mathcal{I}_{\texttt{has\_even},n}|} \sum_{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n}} \text{running time of } \texttt{has\_even(numbers)} \\ &= \frac{1}{n^n} \sum_{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n}} \text{running time of } \texttt{has\_even(numbers)} \end{aligned}$$

The running time of `has_even` on `numbers` is determined by the smallest index `i` for which `numbers[i]` is even.

When the first even number in `numbers[i]` has index 0, the running time is 1 step. When it has index 1, the running time is 2 steps. (Here we are simply counting the number of iterations of the loop and taking the running time of the loop body as 1 step.) When `numbers` does not contain an even number, the running time is $n + 1$ steps. (There are n iterations of the loop followed by 1 step for the return statement.)

In general, when `numbers` contains an even number and the first even is at index i, the running time is $i + 1$ steps. When `numbers` does not contain an even number, the running time is $n + 1$ steps.

We can this observation to simpify the single summation given above. We have:

$$Avg_{\texttt{has\_even}}(n) = \frac{1}{n^n}\left(\begin{array}{l} \Big(\displaystyle\sum_{i=0}^{n-1} \sum_{\substack{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n} \\ \texttt{numbers[i] \% 2 == 0}}} \text{running time of } \texttt{has\_even(numbers)}\Big) \\ + \displaystyle\sum_{\substack{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n} \\ \texttt{all items in numbers[i] are odd}}} \text{running time of } \texttt{has\_even(numbers)} \end{array} \right)$$

$$= \frac{1}{n^n}\left(\begin{array}{l} \Big(\displaystyle\sum_{i=0}^{n-1} \sum_{\substack{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n} \\ \texttt{numbers[i] \% 2 == 0}}} (i+1)\Big) \\ + \displaystyle\sum_{\substack{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n} \\ \texttt{all items in numbers[i] are odd}}} (n+1) \end{array} \right)$$

$$= \frac{1}{n^n}\left(\begin{array}{l} \Big(\displaystyle\sum_{i=0}^{n-1} (i+1) \sum_{\substack{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n} \\ \texttt{numbers[i] \% 2 == 0}}} (1)\Big) \\ +(n+1) \displaystyle\sum_{\substack{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n} \\ \texttt{all items in numbers[i] are odd}}} (1) \end{array} \right)$$

The expression

$$\sum_{\substack{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n} \\ \texttt{numbers[i] \% 2 == 0}}} (1)$$

is equal to the number of lists in S_n with first even item at index i, while the expression

$$\sum_{\substack{\texttt{numbers} \in \mathcal{I}_{\texttt{has\_even},n} \\ \texttt{all items in numbers[i] are odd}}} (1)$$

is equal to the number of lists in S_n that only contain odd numbers.
The value of each of those expressions depends on whether n is even or odd.
Let's first consider the case that n is even.
Then there are $\frac{n}{2}$ even numbers between 1 and n (inclusive), and $\frac{n}{2}$ odd numbers between 1 and n.

The number of lists in S_n with first even item at index i is then

$$\left(\frac{n}{2}\right)^i \times \left(\frac{n}{2}\right)^1 \times \left(n\right)^{n-i-1}$$

$$= \frac{n^n}{2^{i+1}}$$

since each of the first i items can be any one of the possible odd numbers, the item at index $[i]$ can be any one of the possible even numbers and the remaining $n-i-1$ items can be any one of the numbers between 1 and n (inclusive).
The number of lists in S_n that only contain odd numbers is

$$\left(\frac{n}{2}\right)^n$$

$$= \frac{n^n}{2^n}.$$

Substituting these new expressions into our average-case running time expression gives:

$$\begin{aligned}
Avg_{\texttt{has_even}}(n) &= \frac{1}{n^n}\left(\left(\sum_{i=0}^{n-1}(i+1)\frac{n^n}{2^{i+1}}\right) + (n+1)\frac{n^n}{2^n}\right) \\
&= \frac{n^n}{n^n}\left(\left(\sum_{i=0}^{n-1}(i+1)\frac{1}{2^{i+1}}\right) + \frac{n+1}{2^n}\right) \\
&= \left(\left(\sum_{i=1}^{n}(i)\frac{1}{2^i}\right) + \frac{n+1}{2^n}\right) \\
&= \left(\left(\sum_{i=1}^{n}(i)\left(\frac{1}{2}\right)^i\right) + \frac{n+1}{2^n}\right)
\end{aligned}$$

At this point, we will find it helpful to use the following formula from the course notes (pg. 115):

$$\sum_{i=0}^{n-1} ir^i = \frac{nr^n}{r-1} + \frac{r-r^{n+1}}{(r-1)^2}$$

or the equivalent:

$$\sum_{i=1}^{n} ir^i = \frac{(n+1)r^{n+1}}{r-1} + \frac{r-r^{n+2}}{(r-1)^2}$$

We then have, using $r = \frac{1}{2}$, that

$$
\begin{aligned}
Avg_{\texttt{has\_even}}(n) &= \left(\Big(\sum_{i=1}^{n}(i)\big(\frac{1}{2}\big)^i\Big) + \frac{n+1}{2^n}\right) \\
&= \left(\frac{(n+1)(\frac{1}{2})^{n+1}}{\frac{1}{2}-1} + \frac{\frac{1}{2}-(\frac{1}{2})^{n+2}}{(\frac{1}{2}-1)^2} + \frac{n+1}{2^n}\right) \\
&= \left(-2(n+1)\Big(\frac{1}{2}\Big)^{n+1} + 2^2\Big(\frac{1}{2}-\Big(\frac{1}{2}\Big)^{n+2}\Big) + (n+1)\Big(\frac{1}{2}\Big)^n\right) \\
&= \left(-(n+1)\Big(\frac{1}{2}\Big)^n + \Big(2-\Big(\frac{1}{2}\Big)^n\Big) + (n+1)\Big(\frac{1}{2}\Big)^n\right) \\
&= 2 - \frac{1}{2^n}
\end{aligned}
$$

Now we need to repeat the same exercise for n odd!
Suppose that n is odd.
Then there are $\dfrac{n-1}{2}$ even numbers between 1 and n (inclusive), and $\dfrac{n+1}{2}$ odd numbers between 1 and n.
The number of lists in S_n with first even item at index i is then

$$
\Big(\frac{n+1}{2}\Big)^i \times \Big(\frac{n-1}{2}\Big)^1 \times \Big(n\Big)^{n-i-1}
$$

$$
\Big(\frac{1}{2}\Big)^{i+1} \times \Big(n+1\Big)^i \times \Big(n-1\Big)^1 \times \Big(n\Big)^{n-i-1}
$$

$$
\Big(\frac{1}{2}\Big)^{i+1} \times \Big(n+1\Big)^{i+1} \times \Big(\frac{n-1}{n+1}\Big) \times \Big(n\Big)^{n-i-1}
$$

since each of the first i items can be any one of the possible odd numbers, the item at index $[i]$ can be any one of the possible even numbers and the remaining $n-i-1$ items can be any one of the numbers between 1 and n (inclusive).
The number of lists in S_n that only contain odd numbers is

$$
\begin{aligned}
&\Big(\frac{n+1}{2}\Big)^n \\
= \quad &\frac{(n+1)^n}{2^n}.
\end{aligned}
$$

Substituting these new expressions into our average-case running time expression gives:

$$
\begin{aligned}
Avg_{\texttt{has\_even}}(n) &= \frac{1}{n^n}\left(\Big(\sum_{i=0}^{n-1}(i+1)\Big(\frac{1}{2}\Big)^{i+1}\Big(n+1\Big)^{i+1}\Big(\frac{n-1}{n+1}\Big)\Big(n\Big)^{n-i-1}\Big) + (n+1)\frac{(n+1)^n}{2^n}\right)\\
&= \left(\Big(\sum_{i=0}^{n-1}(i+1)\Big(\frac{1}{2}\Big)^{i+1}\frac{1}{n^{i+1}}\Big(n+1\Big)^{i+1}\Big(\frac{n-1}{n+1}\Big)\frac{1}{n^{n-i-1}}\Big(n\Big)^{n-i-1}\Big) + (n+1)\frac{1}{n^n}\frac{(n+1)^n}{2^n}\right)\\
&= \left(\Big(\sum_{i=0}^{n-1}(i+1)\Big(\frac{1}{2}\Big)^{i+1}\Big(1+\frac{1}{n}\Big)^{i+1}\Big(\frac{n-1}{n+1}\Big)\Big) + (n+1)\frac{(1+\frac{1}{n})^n}{2^n}\right)\\
&= \left(\Big(\frac{n-1}{n+1}\Big)\Big(\sum_{i=1}^{n}(i)\Big(\frac{1}{2}\Big)^{i}\Big(1+\frac{1}{n}\Big)^{i}\Big) + (n+1)\Big(\frac{1}{2}\Big)^{n}\Big(1+\frac{1}{n}\Big)^{n}\right)
\end{aligned}
$$

Now let's apply the previously mentioned summation formula with

$$r = \Big(\frac{1}{2}\Big)\Big(1+\frac{1}{n}\Big) = \frac{n+1}{2n}.$$

Note that $r-1 = \dfrac{1-n}{2n}$.
Also note that the last term is the sum above can be written as

$$(n+1)\Big(\frac{1}{2}\Big)^{n}\Big(1+\frac{1}{n}\Big)^{n} = (n+1)r^n.$$

$$
\begin{aligned}
Avg_{\texttt{has\_even}}(n) &= \left(\Big(\frac{n-1}{n+1}\Big)\Big(\sum_{i=1}^{n}(i)\Big(\frac{1}{2}\Big)^{i}\Big(1+\frac{1}{n}\Big)^{i}\Big) + (n+1)\Big(\frac{1}{2}\Big)^{n}\Big(1+\frac{1}{n}\Big)^{n}\right)\\
&= \left(\Big(\frac{n-1}{n+1}\Big)\Big(\Big(\frac{(n+1)r^{n+1}}{r-1}\Big)+\Big(\frac{r-r^{n+1}}{(r-1)^2}\Big)\Big) + (n+1)r^n\right)\\
&= \left(\Big(\frac{n-1}{n+1}\Big)\Big(\frac{(n+1)r^{n+1}}{r-1}\Big)+\Big(\frac{n-1}{n+1}\Big)\Big(\frac{r-r^{n+1}}{(r-1)^2}\Big) + (n+1)r^n\right)
\end{aligned}
$$

The first term can be rearranged to get $-2nr^{n+1}$.
The middle term can be rearranged to get $\dfrac{(1-r^n)(2n)}{(n-1)}$.
The three terms can be added to get an expression of the form:

$$2\frac{n}{n-1} - r^n \cdot A_n$$

where A_n is a positive quantity.
We have then that

$$Avg_{\texttt{has\_even}}(n) = 2\frac{n}{n-1} - B_n$$

where B_n is a quantity that goes to 0 from above as n increases.
This expression has a similar form as in the case of n even, as one would expect.
(We should have restricted the assigned discussion to even n.)