**Topic: Algorithm Analysis**

1. **Intuition:**

First we observe that the number of steps executed depends only on the length of the list A. It does not depend on the contents of the list. Also, the number of iterations of the loop for any input A depends on the length of the list A, which we will call \( n \).

The variable index increases by the value of variable step near the end of each iteration, and the value of variable step increases by 1 at the end of each iteration.

Variable step takes on the values:

\[ \text{step} = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow \ldots \]

and variable index takes on the values:

\[ \text{index} = 0 \rightarrow 1 \rightarrow 3 \rightarrow 6 \rightarrow 10 \rightarrow 15 \rightarrow \ldots . \]

Let \( k \) represent the total number of iterations of the loop body performed for a list of length \( n \). When the loop has completed, the value of variable index will be \( \sum_{i=1}^{k} \text{step}_i \), where \( \text{step}_i \) is the value of variable step at the start of the \( i \)th iteration of the loop. But, since \( \text{step}_i = i \), we have \( \text{index} = \sum_{i=1}^{k} i \).

But we know from class that \( \sum_{i=1}^{k} i = \frac{k(k+1)}{2} \). Since the loop exits when index \( \geq n \), we know that \( k \) will be the smallest integer such that \( \frac{k(k+1)}{2} \geq n \). This means that \( k(k + 1) \geq 2n \), or \( k^2 + k - 2n \geq 0 \).

Solving the quadratic equation \( k^2 + k - 2n = 0 \) gives \( k = \frac{-1 + \sqrt{1 + 8n}}{2} \). Since \( k \geq 0 \), we want the positive solution. Also, since \( k \) is integer, we know that the number of iterations of the loop body is \( \lceil \frac{-1 + \sqrt{1 + 8n}}{2} \rceil \). So, the number of iterations of the loop varies with the square-root of \( n \). This tells us that our runtime bound will involve \( \sqrt{n} \).

Tracing the program we can generate the following table of observations:

<table>
<thead>
<tr>
<th>( n )</th>
<th>final value of index</th>
<th>number of iterations</th>
<th>( \lceil \frac{-1 + \sqrt{1 + 8n}}{2} \rceil )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

1
We can note that the value generated by the formula that we derived is correct (as of course it should be if we have not made a mistake!).

Since there is a constant number of steps in loop, the total number of steps will not be more than a constant times the number of iterations of the loop, $\lceil -\frac{1+\sqrt{1+8n}}{2} \rceil$. For this reason, our intuition tells us that the worst-case running time is $O(\sqrt{n})$.

To determine a lower-bound on the worst-case running time, we need to think about what sort of input produces a worst-case time. But, for this algorithm, the runtime does not depend on the contents of the list, only its length. Since each loop-iteration will take at least one step, our intuition tells us that the worst-case running time is $\Omega(\sqrt{n})$.

Altogether, then, the worst-case running time is $\Theta(\sqrt{n})$.

To prove this conclusion in a detailed way, we need to put the above arguments in a formal proof that follows the usual structure.

Before presenting the formal proof, let us note that

$$\lceil -\frac{1+\sqrt{1+8n}}{2} \rceil = \left\lceil -\frac{1}{2} + \frac{\sqrt{1+8n}}{2} \right\rceil = \left\lceil -\frac{1}{2} + \sqrt{\frac{1}{4} + 2n} \right\rceil \leq \left\lceil \sqrt{\frac{1}{4} + 2n} \right\rceil \leq \sqrt{4n} = 2\sqrt{n}.$$  

(We increase the argument in the square-root so that we can remove the use of the ceiling function. And we increase it in such a way that we can pull the constant out of the square-root function.)

Also,

$$\lceil -\frac{1+\sqrt{1+8n}}{2} \rceil = \left\lceil -\frac{1}{2} + \sqrt{\frac{1}{4} + 2n} \right\rceil \geq \sqrt{n}.$$  

(Here we can remove the use of the ceiling function since we are trying to underestimate the quantity. And we can decrease the argument in the square-root so that we can remove the messy constants. Our goal is to get a simple function that grows in the same way as the number of loop iterations.)

Let $I$ be the set of possible inputs to the algorithm. That is, $I$ is the set of nonempty lists of numbers.

Let $c_0 = 9$ and $B_0 = 36$.  # Choose $B_0 \geq 36$ so $\sqrt{n} \geq 6$

Then $c_0 \in \mathbb{R}^+$ and $B_0 \in \mathbb{N}$.

Assume $A \in I$ is an arbitrary list of length $n \geq B_0$

Then we will take steps in lines 1, 2, 3, 4 and 9 (5 steps in total).

Then the algorithm executes 4 steps for each iteration of a while-loop (lines 5,6,7,8), and loops for not more than $2\sqrt{n}$ iterations.

Then, adding in the step for when the loop condition is false, not more than $4 \cdot 2\sqrt{n} + 1$ steps are executed in lines 5, 6, 7 and 8.

Then, adding together all of the steps, not more than $8\sqrt{n} + 6$ steps are executed.

Then $t_{Q1F}(A) \leq 8\sqrt{n} + 6$.

Then $t_{Q1F}(A) \leq 9\sqrt{n}$.  # Since $n \geq 36, 6 \leq \sqrt{n}$.

Then $t_{Q1F}(A) \leq c_0\sqrt{n}$.  

2
Let $c_0 = 14$ and $B_0 = 1$.

Then $c_0 \in \mathbb{R}^+$ and $B_0 \in \mathbb{N}$.

Assume $n \in \mathbb{N}$ and $n \geq B_0$

Let $A_0 = [1, 2, 3, \ldots, n]$.
Then $A_0 \in I$ and size($A_0$) = $n$.

Then the algorithm executes not fewer than $\sqrt{n}$ iterations of at least one step.
# Ignore the other steps since we are looking for a lower bound.

Then $t_{Q1F}(A_0) \geq \sqrt{n}$ steps.
Then $t_{Q1F}(A_0) \geq c_0\sqrt{n}$ steps.
Then size($A_0$) = $n \land t_{Q1F}(A_0) \geq c_0\sqrt{n}$.

Then $\exists A \in I, \text{size}(A) = n \land t_{Q1F}(A) \geq c_0\sqrt{n}$.

Then $\forall n \in \mathbb{N}, n \geq B_0 \Rightarrow \exists A \in I, \text{size}(A) = n \land t_{Q1F}(A) \geq c_0\sqrt{n}$.

Then $T(n) \in \Omega(\sqrt{n})$.

Then $T(n) \in O(\sqrt{n}) \land T(n) \in \Omega(\sqrt{n})$.
Then $T(n) \in \Theta(\sqrt{n})$.

Here, as is usual, $T(n)$ is the worst-case running time for the algorithm on a list of length $n$.

2. Note: This function performs the well-known bubble sort algorithm, but ends once it is noticed that the list is sorted (i.e., it is noticed that no swaps were performed on the previous pass).

Intuition: Since we have two, nested, for-loops, and each for loop executes a number of times that depends on $n$ (but is not more than $n$), our intuition tells us that the worst-case running time is $O(n^2)$.

To determine a lower-bound on the worst-case running time, we need to think about what sort of input produces a worst-case time. The worst-case time is triggered when the given list $A$ is sorted in reverse order, because then there will be a swap of at least one pair of elements in each pass through the list. In this case, we will do $n-1$ passes through the list, and each time could perform a number of swaps that depends on $n$. Our intuition tells us that the worst-case running time is $\Omega(n^2)$.

Altogether, then, the worst-case running time is $\Theta(n^2)$.

To prove this conclusion in a detailed way, we need to put the above arguments in a formal proof that follows the usual structure.

Before doing this, we note that in python, range($n-1$, 0, -1) generates the list of values $[n-1, n-2, n-3, \ldots, 2, 1]$. And range(passnum) generates the list of values $[0, 1, 2, \ldots, \text{passnum}-2, \text{passnum}-1]$.

Let $I$ be the set of possible inputs to the algorithm. That is, $I$ is the set of nonempty lists of numbers.

Let $c_0 = 14$ and $B_0 = 1$.
Then $c_0 \in \mathbb{R}^+$ and $B_0 \in \mathbb{N}$.
Assume $A \in I$ is an arbitrary list of length $n \geq B_0$
Then we will take steps in lines 1 and 12. (We add these 2 steps in at the end.)

Then, in lines 5, 6, 7, 8 and 9, we will perform at most 5 steps.
The loop in lines 4 – 9 will execute at most passnum iterations of at most 6 steps, and then 1 step when the loop condition fails.
(The 5 becomes a 6 when we count a step for the for-loop statement.)
Since passnum < n, the execution of lines 4 – 9 will take not more than 6n + 1 steps.
Then the execution of lines 3 – 11 will take not more than 1 + 6n + 1 + 2 = 6n + 4 steps.
The loop in lines 2 – 11 will execute not more than n − 1 times, which we over estimate to be not more than n times.
Then the execution of lines 2 – 11 will take not more than n(7n + 4) + 1 steps.
(The 6 becomes a 7 when we count a step for the for-loop statement, and we then add 1 step for when the loop condition fails.)
Then, in total (adding in lines 1 and 12), the algorithm requires not more than 7n^2 + 4n + 3 steps to be executed.

Then \( t_{Q2F}(A) \leq 7n^2 + 4n + 3 \).
Then \( t_{Q2F}(A) \leq 14n^2 \). # Since \( n \geq 1, 4n^2 \geq 4n \) and \( 3n^2 \geq 3 \).
Then \( t_{Q2F}(A) \leq c_0n^2 \).

Let \( c_0 = \frac{1}{4} \) and \( B_0 = 2 \).
Then \( c_0 \in \mathbb{R}^+ \) and \( B_0 \in \mathbb{N} \).
Assume \( n \in \mathbb{N} \) and \( n \geq B_0 \)
Let \( A_0 = [n, n-1, n-2, \ldots, 3, 2, 1] \).
Then \( A_0 \in I \) and \( \text{size}(A_0) = n \).
Then, in the inner loop (lines 5 – 9), at least 1 step is executed, and this inner loop goes through passnum iterations.
Then, in the outer loop (lines 2 – 11), at least passnum steps are executed each iteration.
Then, since the initial list is in reverse order, there will be at least one swap each pass through the list.
Then the outer loop is executed for passnum = n − 1 down to 1. (There is no early break from the outer loop.)

Then at least \( \sum_{\text{passnum}=n-1}^{n} \) (passnum) steps are executed.

Then \( t_{Q2F}(A_0) \geq \sum_{\text{passnum}=n-1}^{n} \) (passnum).

Then \( t_{Q2F}(A_0) \geq \sum_{\text{passnum}=1}^{n-1} \) (passnum).

Then \( t_{Q2F}(A_0) \geq \frac{(n-1)((n-1)+1)}{2} \).

Then \( t_{Q2F}(A_0) \geq \frac{1}{2}(n^2 - n) \).

Then \( t_{Q2F}(A_0) \geq \frac{1}{2}(n^2 - \frac{1}{2}n^2) \). # for \( n \geq B_0, \frac{1}{2}n^2 \geq n \).
Then $t_{Q2F}(A_0) \geq \frac{1}{4} n^2$.
Then $t_{Q2F}(A_0) \geq c_0 n^2$.
Then $t_{Q2F}(A_0) \geq c_0 n^2$.
Then $\exists A \in I, \text{size}(A) = n \wedge t_{Q2F}(A_0) \geq c_0 n^2$.

Then $\forall n \in \mathbb{N}, n \geq B_0 \Rightarrow \exists A \in I, \text{size}(A) = n \wedge t_{Q2F}(A) \geq c_0 n^2$.
Then $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B_0 \Rightarrow \exists A \in I, \text{size}(A) = n \wedge t_{Q2F}(A) \geq c_0 n^2$.
Then $T_{Q2F}(n) \in \Omega(n^2)$.

Then $T(n) \in O(n^2) \wedge T(n) \in \Omega(n^2)$.
Then $T_{Q2F}(n) \in \Theta(n^2)$.

Here, as is usual, $T_{Q2F}(n)$ is the worst-case running time for the algorithm on a list of length $n$. 