1. Find a tight bound on the worst-case running time of the following algorithm.

```
# Precondition: L is a list that contains n > 0 real numbers.
1. max ← 0
2. for i ← 0, 1, . . . , n − 1:
3.   for j ← i, i + 1, . . . , n − 1:
4.     sum ← 0
5.     for k ← i, i + 1, . . . , j:
6.       sum ← sum + L[k]
7.     if sum > max:
8.       max ← sum
```

Intuitively, \( T(n) \in \mathcal{O}(n^3) \) because of the three nested loops, each one of which iterates no more than \( n \) times. We want to prove this formally, and also show that the bound is tight (i.e., \( T(n) \in \Omega(n^3) \)).

\[ T(n) \in \mathcal{O}(n^3): \]

**Proof Structure:**

- Let \( c' = \ldots \) and \( B' = \ldots \)
- Then \( c' \in \mathbb{R}^+ \) and \( B' \in \mathbb{N} \).
- Assume \( n \in \mathbb{N} \) and \( n \geq B' \) and \( L \) is a list of \( n \) real numbers.
- \( \ldots \) show \( t(L) \leq c'n^3 \ldots \) (\( t(L) \) is the number of steps taken by the algorithm on input \( L \))
- Then \( \forall n \in \mathbb{N}, n \geq B' \Rightarrow \forall L \in \{ \text{all lists of real numbers} \}, \text{len}(L) = n \Rightarrow t(L) \leq c'n^3 \).
- Then \( \exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow T(n) \leq cn^3 \).

**Scratch Work:** To find values of \( c \) and \( B \) that work, we over-estimate the number of steps taken by the algorithm. This simplifies the computation: we don’t have to find the exact number of steps carried out, just a value that is clearly greater than or equal to the number of steps.

In this case, working inside-out, we get that:

- line 6 takes 1 step;
- the loop on lines 5–6 iterates at most \( n \) times (because \( i \in \{0, 1, \ldots, n - 1\} \) and \( j \in \{i, i + 1, \ldots, n - 1\} \)), so the number of steps is \( \leq n \cdot 1 = n \);
- lines 4–8 add at most 3 steps to this (counting each line separately);
- the loop on lines 3–8 iterates at most \( n \) times, so the number of steps is \( \leq n \cdot (n + 3) \leq n \cdot (n + n) = 2n^2 \) (if \( n \geq 3 \)) — we do this to keep the expression as simple as possible;
- the loop on lines 2–8 iterates exactly \( n \) times, so the number of steps is \( \leq n \cdot 2n^2 = 2n^3 \);
- line 1 adds 1 step to this, so the number of steps is \( \leq 2n^3 + 1 \leq 2n^3 + n^3 = 3n^3 \) (if \( n \geq 1 \)).

**Complete Proof:**

Assume \( n \in \mathbb{N} \) and \( n \geq 3 \) and \( L \) is a list of \( n \) real numbers.

Then the first line takes \( 1 < n < n^3 \) steps.

Also, the loop over \( i \) iterates exactly \( n \) times, and for each iteration...

- The loop over \( j \) iterates at most \( n \) times, and for each iteration...
  - The loop over \( k \) iterates at most \( n \) times, and each iteration takes 1 step, for a total of at most \( n \) steps.
  - The other statements in the loop body for \( j \) take at most 3 steps.
  - So the loop body for \( j \) takes at most \( n + 3 \leq 2n \) steps.
  - So the loop over \( j \) takes at most \( 2n^2 \) steps.

...so the loop over \( i \) takes at most \( 2n^3 \) steps.

The entire algorithm therefore takes at most \( n^3 + 2n^3 = 3n^3 \) steps.

Then, \( \forall n \in \mathbb{N}, n \geq 3 \Rightarrow \forall L \in \{ \text{all lists of real numbers} \}, \text{len}(L) = n \Rightarrow t(L) \leq 3n^3 \).

Hence, \( T(n) \in \mathcal{O}(n^3) \).
Proof Structure:

Let \( c' = \ldots \) and \( B' = \ldots \)
Then \( c' \in \mathbb{R}^+ \) and \( B' \in \mathbb{N} \).
Assume \( n \in \mathbb{N} \) and \( n \geq B' \).

Let \( L = \ldots \)
Then \( L \) is a list of \( n \) real numbers.
\[ T(n) \in \Omega(n^3) \]

Then show that \( t(L) \geq c'n^3 \ldots \)
Then \( \forall n \in \mathbb{N}, n \geq B' \Rightarrow \exists L \in \{ \text{all lists of real numbers} \}, \text{len}(L) = n \land t(L) \geq c'n^3. \)

Then \( \exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow T(n) \geq cn^3. \)

Scratch Work: Note that the running time of the algorithm does not depend on the contents of \( L \): it is the same for every list of length \( n \). This means all we have to argue is that the algorithm always carries out at least some fraction of \( n^3 \) many steps.

In other words, we have to show that the loop over \( k \) iterates at least some fraction of \( n \) times, for at least a fraction of \( n \) many values of \( j \), for at least a fraction of \( n \) many values of \( i \).
To keep things simple, let’s split up the range \([0, \ldots, n-1]\) into thirds, roughly: \([0, \ldots, n/3] \), \([n/3, \ldots, 2n/3] \), \([2n/3, \ldots, n-1] \) (we’ll add appropriate floors and/or ceilings later on, to ensure every value is an integer). There are many other ways we could have done this! The important thing is to come up with a collection of pairs \((i, j)\) that contains at least \( n^2 \) many pairs (within a constant factor) and for which the difference \( j - i \) is at least some constant fraction of \( n \). In this case:

- \( i \) iterates over at least the \( n/3 \) values \( \{0, 1, \ldots, n/3 - 1\} \) (more than that actually);
- for each of those values of \( i, j \) iterates over at least the \( n/3 \) values \( \{2n/3, \ldots, n-1\} \) (more than that actually);
- for each of these \( n^2/9 \) many pairs \((i, j)\), \( k \) iterates over every value \( \{i, \ldots, j\} \), and there are at least \( n/3 \) many values in that range (more than that actually).

This means the algorithm always executes line 6 at least \( n^3/27 \) many times.

To formalize this, a bit of trial and error shows that

- The range \( \{0, \ldots, [n/3]\} \) contains \( [n/3] + 1 > n/3 \) values.
- The range \( \{\lfloor 2n/3 \rfloor, \ldots, n-1\} \) contains \( n - 1 - \lfloor 2n/3 \rfloor + 1 \geq n - 2n/3 = n/3 \) values (because \( \lfloor 2n/3 \rfloor \leq 2n/3 \Rightarrow -\lfloor 2n/3 \rfloor \geq -2n/3 \)).
- The range \( \{\lfloor n/3 \rfloor, \ldots, \lfloor 2n/3 \rfloor\} \) contains \( \lfloor 2n/3 \rfloor - \lfloor n/3 \rfloor + 1 \geq 2n/3 - n/3 = n/3 \) values (because \( \lfloor 2n/3 \rfloor + 1 > 2n/3 \)).

Complete Proof:
Assume \( n \in \mathbb{N} \) and \( n \geq 1 \).
Let \( L = [1, 2, \ldots, n] \).
Then for each value of \( i \in \{0, \ldots, [n/3]\} \ldots \)
For each value of \( j \in \{\lfloor 2n/3 \rfloor, \ldots, n-1\} \ldots \)
The loop for \( k \) iterates over every value in \( \{i, \ldots, j\} \), and executes 1 step at each iteration.
So the loop for \( k \) takes at least \( n/3 \) steps (since there are at least \( \lfloor 2n/3 \rfloor - \lfloor n/3 \rfloor + 1 \geq n/3 \) values for \( k \)).
\( \ldots \) so the loop for \( j \) takes at least \( n^2/9 \) steps (since there are at least \( n - \lfloor 2n/3 \rfloor \geq n/3 \) values for \( j \)).
\( \ldots \) so the loop for \( i \) takes at least \( n^3/27 \) steps (since there are at least \( \lfloor n/3 \rfloor + 1 > n/3 \) values for \( i \)).
Then \( \forall n \in \mathbb{N}, n \geq 1 \Rightarrow \exists L \in \{ \text{all lists of real numbers} \}, \text{len}(L) = n \land t(L) \geq n^3/27. \)
Hence, \( T(n) \in \Omega(n^3) \).
2. Prove that $T_{BFT}(n) \in \Theta(n^2)$, where BFT is the algorithm below.

BFT$(E, n)$:
1. $i \leftarrow n - 1$
2. while $i > 0$
3. \hspace{1em} $P[i] \leftarrow -1$
4. \hspace{1em} $Q[i] \leftarrow -1$
5. \hspace{1em} $i \leftarrow i - 1$
6. $P[0] \leftarrow n$
7. $Q[0] \leftarrow 0$
8. $t \leftarrow 0$
9. $h \leftarrow 0$
10. while $h \leq t$
11. \hspace{1em} $i \leftarrow 0$
12. \hspace{1em} while $i < n$
13. \hspace{2em} if $E[Q[h]][i] \neq 0$ and $P[i] < 0$
14. \hspace{3em} $P[i] \leftarrow Q[h]$
15. \hspace{3em} $t \leftarrow t + 1$
16. \hspace{3em} $Q[t] \leftarrow i$
17. \hspace{2em} $i \leftarrow i + 1$
18. \hspace{1em} $h \leftarrow h + 1$

(Although this is not directly relevant to the question, this algorithm carries out a breadth-first traversal of the graph on $n$ vertices whose adjacency matrix is stored in $E$.)

We show that $T_{BFT}(n) \in \Theta(n^2)$ by proving $T_{BFT}(n) \in O(n^2)$ and $T_{BFT}(n) \in \Omega(n^2)$. 
$T_{\text{BFT}}(n) \in \mathcal{O}(n^2)$:

Let $c = 16$ and $B = 1$. Then, $c \in \mathbb{R}^+$ and $B \in \mathbb{N}$.

Assume $n \in \mathbb{N}$, $n \geq B = 1$, and $E$ is an arbitrary input of size $n$.

One of the tricky features of this algorithm is that the main loop depends on the values of $h$ and $t$, but the algorithm does not explicitly bound either value. To prove an upper bound on $T_{\text{BFT}}(n)$, we start by proving a bound on the value of $t$. Namely, we show that at any point during the execution of the algorithm, $t \leq n$.

From lines 1–9, when the main loop (lines 10–18) begins execution, $h = t = 0$, $P[0] = n$, $Q[0] = 0$, and $P[i] = Q[i] = -1$ for $i = 1, 2, \ldots, n - 1$.

Note that the value of $t$ is changed only on line 15, and this line is executed only when $P[i] < 0$ (among other conditions).

Moreover, each time $t$ is incremented, the value of $Q[i]$ is set to a natural number (on line 16), so that at any point during the execution of the algorithm, $Q[0 \ldots t] \in \mathbb{N}$ and $Q[t + 1 \ldots n - 1] = -1$. Since $h \leq t$ (from line 10), this means that $Q[h] \geq 0$ is always true inside the main loop.

Hence, on line 14, the assignment $P[i] = Q[h]$ guarantees that $P[i] \geq 0$ from that point on. This means that the value of $t$ can increase at most once for each value of $i = 0, 1, \ldots, n - 1$ (it increases only when $P[i] < 0$, at which point $P[i]$ is set to a natural number), i.e., $t \leq n$.

From the algorithm,

- lines 2–5 take 4 steps for one iteration, and are executed exactly $n - 1$ times (once for each value of $i = n - 1, n - 2, \ldots, 1$), plus 1 more step for the last execution of line 2, for a total of $4(n - 1) + 1 = 4n - 3$ steps;
- lines 6–9 take 4 steps;
- lines 12–17 take at most 6 steps for one iteration (if the condition of the if statement is true at every iteration), and are executed exactly $n$ times (once for each value of $i = 0, 1, \ldots, n - 1$), plus 1 more step for the last execution of line 12, for a total of at most $6n + 1$ steps;
- lines 10–18 take at most $6n + 1 + 3 = 6n + 4$ steps for one iteration, and are executed at most $n$ times (since $t \leq n$, as shown above), for a total of at most $6n^2 + 4n$ steps;
- so in total, the algorithm takes at most $1 + 4n - 3 + 4 + 6n^2 + 4n = 6n^2 + 8n + 2$ steps.

Since $n \geq 1$, this means that the number of steps executed by the algorithm on input $(E, n)$ is $\leq 6n^2 + 8n + 2 \leq 6n^2 + 8n^2 + 2n^2 = 16n^2$.

Since $(E, n)$ was arbitrary, $\forall n \in \mathbb{N}, n \geq 1 \Rightarrow T_{\text{BFT}}(n) \leq 16n^2$.

Therefore, $T_{\text{BFT}}(n) \in \mathcal{O}(n^2)$.

$T_{\text{BFT}}(n) \in \Omega(n^2)$:

Let $c = 1$ and $B = 1$. Then, $c \in \mathbb{R}^+$ and $B \in \mathbb{N}$.

Assume $n \in \mathbb{N}$ and $n \geq B = 1$.

Consider an input $(E, n)$ such that $E[i][j] = 1$ for all indices $0 \leq i < n, 0 \leq j < n$.

The first time that lines 12–17 are executed, the condition of the if statement will be true for all values of $i = 0, 1, \ldots, n - 1$ so at the end of the loop, $t$ will have value at least $n$ (since $t$ starts at 0 and gets incremented $n$ times). Since lines 12–17 always get executed exactly $n$ times (once for each value of $i = 0, 1, \ldots, n - 1$), they take at least $n$ steps.

This means that lines 10–18 will get executed for every value of $h = 0, 1, \ldots, n - 1$ (at least), and take at least $n$ steps at each iteration, for a total of at least $n^2$ steps.

So the number of steps on input $(E, n)$ is $\geq n^2$.

Hence, $\forall n \in \mathbb{N}, n \geq 1 \Rightarrow T_{\text{BFT}}(n) \geq n^2$.

Therefore, $T_{\text{BFT}}(n) \in \Omega(n^2)$. 