Question 9.

Algorithms Assignment 1

Faaiq Bilal
23100104

1 Part 1

We can divide the algorithm into each of its steps, analyze their running time complexity to get bounds for our algorithm's complexity.

The outermost *for* loop will run over all the elements of the array of size n. This step will have a complexity of $O(n)$. The following functions are nested inside this *for* loop, which means that they will be repeated n times as well

The nested *for* loop will run for a maximum of $n - 2$ times, which is still bounded by $O(n)$. As noted above, this *for* loop will be repeated $O(n)$ times as well.

As the final piece of this equation, the integers inside the Array A have to be summed up. The maximum number of computations for a specific $B[ij]$ is equal to the largest difference between j and i. This is bounded by $O(n)$, as i ranges $1 \rightarrow n$ and j ranges $i + 1 \rightarrow n$.

It should be noted that this summation is nested inside the above for loop, hence it will be repeated $O(n^2)$ times.

As each computation is of order $O(n)$, and is repeated $O(n^2)$ times, the overall computational complexity of the algorithm will have an upper bound of $O(n^3)$.

2 Part 2

We can use the breakdown from Part 1 to help make a point here as well.

The outermost for loop will run n times, the inner for loop will run from $1 \rightarrow n - 2$ times as well. The innermost summation sums up terms $1 \rightarrow n$ times as well. It should be noted that these ranges only specify the bounds for the number of iterations.

While the exact number of steps can be computed, in our case it suffices to say the following: the first step will take n iterations. The second step will take

$f(n)$ iterations, where f is some function of the order n. The third step will take $g(n)$ iterations, where g is also some function of order n.

The overall number of iterations will be $n \cdot f(n) \cdot g(n)$. As the largest term of this function will be of the order n^3, we can safely say that there exists a constant c such that $c \cdot n^3 \leq n \cdot f(n) \cdot g(n)$.

This shows that Ω of our algorithm is also $\Omega(n^3)$. Which means that the tightest bound can be represented as: $\Theta(n^3)$.

3 Part 3

We can design an algorithm to get rid of the repeated summations in the third part of our algorithm, and instead of summing up every term from the start, we can save the result of the summation so far in a variable. Technically, we could make our algorithm even simpler by not using a temporary variable at all, and could simply use the previous index of B, however then we would require conditions for the first entry of every row of B. The algorithm is as follows:

```
for i=1 to n-1 do:
    sum ← A[i]
    for j=i+1 to n do:
        sum ← sum + A[j]
        B[i,j] ← sum
return B
```

For this algorithm, we only loop through twice. The first loop iterates over the array n times, and the inner loop then iterates $1 \to n-2$ times (only given as a bound on the number of iterations). The innermost summation is now just a single sum. The total number of iterations will now be equal to the size of upper right corner of the matrix B. As the total size of the matrix B is n^2, we can say that the number of iterations are less than n^2. The running time of the algorithm must be of the order $O(n^2)$.

$$\lim_{n\to\infty} \frac{g(n)}{f(n)}$$

$$\lim_{n\to\infty} \frac{n^2}{n^3} = 0$$