Question 2

(a)

```
maxSumMemo(A, j T):
    if(j == 0):
        return A[0]
    if(T[j] != -infinity):
        return T[j]
    prev_max = maxSumMemo(A, j - 1, T) + A[j]
    T[j] = max(prev_max, A[j])
    return T[j]

maxSum(A):
    n = A.length
    T is a table/array of length n, whose values are all initialised to -infinity
    max = maxSumMemo(A, n - 1, T)
    for(k = n - 2; k >= 0; k--):
        curr = maxSumMemo(A, k, T)
        if(curr > max):
            max = curr
    return max
```

(b)

In maxSum(), when maxSumMemo() is called for the first time outside the for-loop, it recursively iterates through all n entries of A, filling up the table T with their maximum subarray sums. This takes n steps.

Then in maxSum(), the for-loop iterates approximately n times, calling maxSumMemo() each time. However, this time maxSumMemo() will only take a fixed number of steps, since the table T has already been filled out. So, the for-loop takes n steps.

So, the number of steps is approximately n + n. Which is a $\Theta(n)$ cost.