Lec. 3

Examples for Computing Time & Space Complexity

Ex. 1:
Find the result of \( \sum_{j=1}^{n} A_j \)

Algorithm \textit{sum}(A,n):
\begin{align*}
\text{Input:} & \quad \text{a positive integer } n \text{ and an array } A \text{ indexed from } 1 \text{ to } n. \\
\text{Output:} & \quad S, \text{ the sum of the numbers in } A.
\end{align*}
\begin{enumerate}
\item \( S \leftarrow 0 \)
\item \textbf{for } \( j \leftarrow 1 \) \textit{ to } \( n \)
\item \( S \leftarrow S + A[j] \)
\item \textbf{end for}
\item \textbf{return } S
\end{enumerate}

This problem is characterized with \( n \)

Space Complexity: This algorithm requires two cells, for \( S \) and \( j \) variables, this is constant storage and independent with problem characteristics, thus

\( S_{\text{sum}}(n) = 0 \)

Time Complexity with Operation counts: With selection addition operation for array elements (A)

\( T_{\text{sum}}(n) = n \)

Time Complexity with Step counts:
\[
\begin{array}{l}
1. \ldots \ldots \ 1 \\
2. \ldots \ldots \ n+1 \\
3. \ldots \ldots \ n \\
4. \ldots \ldots \ 0 \\
5. \ldots \ldots \ 1 \\
\end{array}
\]

\( T_{\text{sum}}(n) = 2n+3 \)

Ex. 2: Find the summation of the elements of two array:
\( C(mn) = A(mn) + B(mn) \)

Algorithm \textit{add} (A, B, m, n):
\begin{align*}
\text{Input:} & \quad \text{a positive integers } m, n \text{ and two-dimensional arrays of numbers } A \text{ and } B \text{ each of which has its rows indexed from } 1 \text{ to } m \text{ and columns from } 1 \text{ to } n. \\
\text{Output:} & \quad \text{a two-dimensional array of numbers } C, \text{ containing addition of } A \text{ and } B.
\end{align*}
\begin{enumerate}
\item \textbf{for } \( i \leftarrow 1 \) \textit{ to } \( m \)
\item \textbf{for } \( j \leftarrow 1 \) \textit{ to } \( n \)
\item \( C[i, j] \leftarrow A[i, j] + B[i, j] \)
\item \textbf{end for}
\item \textbf{end for}
\item \textbf{return} array \( C \)
This problem is characterized with \( m \) and \( n \)

Space Complexity:

\[
S_{\text{add}}(m,n) = mn + 2 \quad \text{(where \( mn \) is array size, 2 for \( i, j \)).}
\]

Time Complexity with Operation counts: With selection addition operation between the elements of (A), (B) arrays

\[
T_{\text{add}}(m,n) = mn
\]

Time Complexity with Step counts:

1. \( \ldots \ldots \) \( m+1 \)
2. \( \ldots \ldots \) \( m(n+1) \)
3. \( \ldots \ldots \) \( m(2n+1) \)
6. \( \ldots \ldots \) \( mn \)

\[
T_{\text{add}}(m,n) = 3mn + 2m + 1
\]

**Note:**

This expression is accepted when \( n \geq m \), but when \( m > n \), two for statements are replaced to decrease time complexity, it becomes:

\[
T_{\text{add}}(m,n) = 3mn + 2n + 1
\]

**H.W.:**

Compute space and time complexity of a problem which finds the summation of the elements of two array:

\[
Z(n) = X(n) + Y(n)
\]

**Ex. 3:** Fibonacci Numbers.

It is starting as:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …………

Where as every new number will be getting through adding two previous numbers, if first number in Fibonacci is \( F_0 \), then \( F_0 = 0 \) and \( F_1 = 1 \), in general:

\[
F_n = F_{n-1} + F_{n-2} \quad , \quad n \geq 2
\]

```plaintext
Algorithm fibonacci (n):
Input: a nonnegative integer n.
Output: fib, the nth term of the fibonacci sequence.
1. if \( n \leq 1 \) then
2. fib \( \leftarrow n \)
3. else
4. fnm1 \( \leftarrow 0 \)
5. fnm2 \( \leftarrow 1 \)
6. for i \( \leftarrow 2 \) to n
7. fib \( \leftarrow fnm1 + fnm2 \)
8. fnm1 \( \leftarrow fnm2 \)
9. fnm2 \( \leftarrow fib \)
10. end for
11. end if
12. return fib
```
This problem is characterized with \( n \).

**Space Complexity:** This algorithm requires four cells, for storing the values of \( \text{fib}, \text{fmn1}, \text{fmn2} \) and \( i \), this is constant storage and independent with problem characteristics, thus

\[ S_{\text{fibonacci}}(n) = 0 \]

**Time Complexity with Operation counts:** With selection assignment operation between the elements of Fibonacci numbers:

7. \( \ldots \quad \ldots \quad n-1 \)
8. \( \ldots \quad \ldots \quad n-1 \)
9. \( \ldots \quad \ldots \quad n-1 \)

\[ T_{\text{fibonacci}}(n) = 3(n-1) = 3n-3 \]

**Time Complexity with Step counts:**

Two cases must be considered in this method:

First case: when \( n=0 \) or \( n=1 \), number of steps is 3.

Second case: when \( n>1 \), number of steps is \( 4n+1 \).  

\[ T_{\text{fibonacci}}(n) = \begin{cases} 3 & \text{if } n \in \{0,1\} \\ 4n+1 & \text{if } n > 1 \end{cases} \]

There is a relationship between time complexity and problem's characteristics, where as increasing \( n \) leads to linear increasing in time complexity, which is better than squared increasing.

**Note:**

In most problems there is a balance between amount of time and algorithm's space, more storage leads to increase running speed, and the reverse of this statement is true.

**Ex. 4:** Reform Fibonacci algorithm.

1. \( \text{comment: f[0..n] is an auxiliary array.} \)
2. \( f[0] \leftarrow 0; \)
3. \( \text{if } n > 0 \text{ then} \)
4. \( f[1] \leftarrow 1 \)
5. \( \text{for } i \leftarrow 2 \text{ to } n \)
6. \( f[i] \leftarrow f[i-1] + f[i-2]; \)
7. \( \text{end for} \)
8. \( \text{return } f[n]; \)

**Space Complexity:**

\[ S_{\text{fibonacci}}(n) = n+2 \]

**Time Complexity with Step counts:**
Ex. 5: Compute Prefix averages for set of numbers.

Problem concept is there is a certain array lets $X$ of $n$ integer numbers, the required is computing another array lets $A$, where an element $A[i]$ is the average of elements from $X[1]$ to $X[i]$ for $i=1 \ldots n$, it is meaning:

$$A_i = \frac{\sum_{j=1}^{i} X_j}{i}$$

Algorithm \textit{prefixAverages}($X$):

\begin{enumerate}
\item for $i \leftarrow 1$ to $n$
\item $a \leftarrow 0$
\item for $j \leftarrow 1$ to $i$
\item $a \leftarrow a + X[j]$
\item end for
\item $A[i] \leftarrow a / i$
\item end for
\item return array $A$
\end{enumerate}

This problem is characterized with $n$.

Time Complexity with Step counts:

1. \ldots $n+1$
2. \ldots $n$
3. \ldots $\sum_{i=1}^{n}(i+1)$ \quad \Rightarrow \quad \sum_{i=1}^{n}(i+1) \sum_{i=1}^{n} \Rightarrow \quad T_{\text{prefixAverages}}(n) = n^2 + 6n + 1$
4. \ldots $\sum_{i=1}^{n}$
5. \ldots $n$
6. \ldots $n$
8. \ldots $n$

The function of time complexity of prefix average problem (\textbf{with the above algorithm}) is \textbf{square}. 