Growable Array-based Stack

- In a push operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one.
- How large should the new array be?
  - Incremental strategy: increase the size by a constant $c$.
  - Doubling strategy: double the size.

**Algorithm** $\text{push}(o)$

```
if $t = S.length - 1$ then
    $A \leftarrow$ new array of size ...
    for $i \leftarrow 0$ to $t$ do
        $A[i] \leftarrow S[i]$
        $S \leftarrow A$
        $t \leftarrow t + 1$
    $S[t] \leftarrow o$
```

Comparison of the Strategies

- We compare the incremental strategy and the doubling strategy by analyzing the total time $T(n)$ needed to perform a series of $n$ push operations.
- We assume that we start with an empty stack represented by an array of size 1.
- We call amortized time of a push operation the average time taken by a push over the series of operations, i.e., $T(n)/n$. 
Incremental Strategy Analysis

- We replace the array \( k = n/c \) times.
- The total time \( T(n) \) of a series of \( n \) push operations is proportional to

\[
 n + c + 2c + 3c + 4c + \ldots + kc = \\
 n + c(1 + 2 + 3 + \ldots + k) = \\
 n + ck(k + 1)/2
\]

- Since \( c \) is a constant, \( T(n) \) is \( O(n + k^2) \), i.e., \( O(n^2) \).
- The amortized time of a push operation is \( O(n) \).
Doubling Strategy Analysis

- We replace the array $k = \log_2 n$ times.
- The total time $T(n)$ of a series of $n$ push operations is proportional to
  
  $n + 1 + 2 + 4 + 8 + \ldots + 2^k = n + 2^{k+1} - 1 = 2n - 1$

- $T(n)$ is $O(n)$
- The amortized time of a push operation is $O(1)$