Tutorial 7 for CSCI2100
Outline

- Dynamic array vs. linked list
- Dynamic array: choose expansion coefficient
- An application of the stack
Dynamic array v.s. Linked list

- A linked list ensures $O(1)$ insertion cost. A dynamic array guarantees $O(1)$ insertion cost after amortization.

- A dynamic array provides constant-time access to any position, which a linked list cannot achieve.
Dynamic array v.s. Linked list

Design a data structure of $O(n)$ space to store a set $S$ of $n$ integers to satisfy the following requirements:

- An integer can be inserted in $O(1)$ time.
- We can enumerate all integers in $O(n)$ time.

**Answer:** Linked list.
Dynamic array v.s. Linked list

Design a data structure of $O(n)$ space to store a set $S$ of $n$ integers to satisfy the following requirements:

- An integer can be inserted in $O(1)$ amortized time.
- We can enumerate all integers in $O(n)$ time.
- For an arbitrary integer $i \in [1, n]$, we can access the $i^{th}$ inserted integer in $O(1)$ time.

**Answer:** Dynamic array.
Outline

- Dynamic array vs. linked list
- Dynamic array: choose expansion coefficient
- An application of the stack
Choose expansion coefficient

In the lecture, we expand the array from size $n$ to $2n$ when it is full.

Think: what if we expand the array size to $3n$ instead?
Choose expansion coefficient

- Suppose, initially, the array has size 3. Define \( s_1 = 3 \).
- At the first expansion, the size of the array increases from \( s_1 \) to \( s_2 = 3s_1 = 9 \).
- At the second expansion, the size of the array increases from \( s_2 \) to \( s_3 = 3s_2 = 27 \).
- \( \ldots \)

- At the \( i \)-th expansion, the size of the array increases from \( s_i \) to \( s_{i+1} = 3s_i = 3^{i+1} \).

We have \( s_{i+1} = 3^{i+1} \). So the cost of the \( i \)-th expansion is \( O(3^{i+1}) = O(3^i) \).
Choose expansion coefficient

Consider we insert $n$ integers to the array. The cost of putting the integers into the array is $O(n)$.

Suppose there are $h$ expansions. The cost of all the expansions is bounded by $\sum_{i=1}^{i=h} O(3^i) = O(3^h)$. So the total insertion cost is $O(n + 3^h)$.

As before the $h$-th expansion, the size of the array is $3^h < n$, so we have the total cost is $O(n + 3^h) = O(n)$. 
Choose expansion coefficient

Now, consider the general case where we expand the array size to $\alpha n$ for some integer $\alpha \geq 2$. Which $\alpha$ is the best?
Choose expansion coefficient

- Suppose, initially, the array has size $\alpha$. Define $s_1 = \alpha$.
- At the first expansion, the size of the array increases from $s_1$ to $s_2 = \alpha s_1 = \alpha^2$.
- At the second expansion, the size of the array increases from $s_2$ to $s_3 = \alpha s_2 = \alpha^3$.
- ...
- At the $i$-th expansion, the size of the array increases from $s_i$ to $s_{i+1} = \alpha s_i = \alpha^{i+1}$

So $s_i = \alpha^i$, and the cost of the $i$-th expansion is $\alpha^{i+1}$. 
Choose expansion coefficient

Consider inserting \( n \) integers into the array.

Suppose there are \( h \) expansions. The cost of all the expansions is asymptotically bounded by \( \sum_{i=1}^{i=h} \alpha^{h+1} = O\left(\frac{\alpha^{h+2}}{\alpha-1}\right) \). So the total insertion cost is \( O(n + \frac{\alpha^{h+2}}{\alpha-1}) \).

Before the \( h \)-th expansion, the size of the array is \( \alpha^h < n \). Hence, the total cost is \( O(n + \frac{\alpha^2}{\alpha-1} n) \), so the amortized cost is \( O(1 + \frac{\alpha^2}{\alpha-1}) \).

Function \( \frac{\alpha^2}{\alpha-1} \) achieves its minimum when \( \alpha = 2 \).
Outline

- Dynamic array vs. linked list
- Dynamic array: choose expansion coefficient
- An application of the stack
An application of the stack

Input: A sentence stored in a sequence of \( n \) cells. Each cell contains a word or one of the following pairing characters:

```
"", (,), {, }, <, >
```

Please design an algorithm to determine whether the pairing characters have been matched correctly (in the way we are used to in English). The following input is a correct sentence:

```
I say "I like ( red ) apple "
```

And the following input is not a correct sentence:

```
I say "I like ( red " ) apple
```

Your algorithm should finish in \( O(n) \) time.
An application of the stack

The key idea is to use a stack to check whether all the ", (, {, < are closed properly. We will discuss the idea on the following two examples:

```
{ < < " " } > ( ) > }
```

```
{ < < " { } > ( ) > }
```
An application of the stack

Solution:
Sequentially scan the input sentences.

- At reading a ‘‘, (, <, {, push it into the stack.
- At reading a ′′, ), >, }, check whether the top of the stack matches the character just read. If so, pop the stack and continue; otherwise, report ”incorrect”.
- After reading all the cells, check whether the stack is empty. If so, report “correct”; otherwise, report “incorrect”.

The time complexity is $O(n)$. 