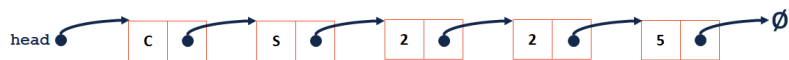


```

List.hpp
103 template <typename T>
104 T List<T>::remove(unsigned index) {
105
106
107
108
109 }

```



List Implementation #2: _____

```

List.h
1 #pragma once
2
3 template <typename T>
4 class List {
5     public:
6     /* ... */
28     private:
29
30
31
32 };

```

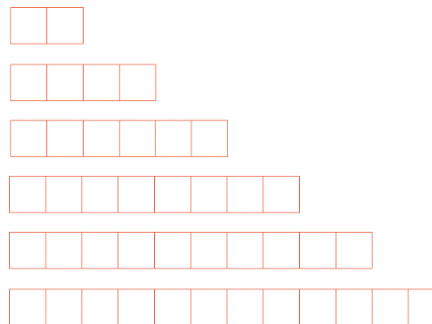
Implementation Details and Analysis:

What is the running time of insertFront()?



→ What is our resize strategy?

Array Resize Strategy #1:



...total copies across all resizes: _____

...total number of insert operations: _____

...average (amortized) cost of copies per insert: _____

Array Resize Strategy #2:



...total copies across all resizes: _____

...total number of insert operations: _____

...average (amortized) cost of copies per insert: _____

Running Time:

	Singly Linked List	Array
Insert/Remove at front		
Insert after a given element		
Remove after a given element		
Insert at arbitrary location		
Remove at arbitrary location		

A List implementation in std

- `std::vector` implements a list with dynamic growth
- `#include <vector>` to use it!
- Documentation widely available, including on CBTF exams
-

Stack ADT

Function Name	Purpose

Queue ADT

Function Name	Purpose

Stack and Queue Implementations

```
Stack.h
1 #pragma once
2
3 #include <vector>
4
5 template <typename T>
6 class Stack {
7     public:
8         void push(const T & d);
9         T pop();
10        bool isEmpty();
11
12    private:
13        std::vector<T> list_;
14 };
15
16 #include "Stack.hpp"
```

```
Stack.hpp
3 template <typename T>
4 void Stack<T>::push(const T & d) {
5     list_.push_back(d);
6 }
7
8 template <typename T>
9 T Stack<T>::pop() {
10    T data = list_.back();
11    list_.pop_back();
12    return data;
13 }
```

CS 225 – Things To Be Doing:

1. mp_stickers due Monday;
2. Daily POTDs