# #5: Array List ______________________________

## List Implementation #2: ______________

```
Alternate List.h
#pragma once

template <typename T>
class List {
  public:
     /* ... */
  private:





};
```

## Array - Implementation Details:

| C | S | 2 | 2 | 5 |
|---|---|---|---|---|
| [0] | [1] | [2] | [3] | [4] |

1. What is the running time of []? How?

2. What is the running time of `insertFront()`? Why?

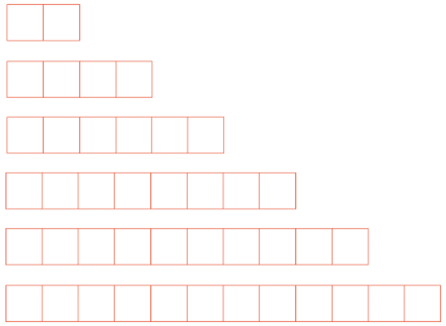
3. What is the running time of `insert` ()? Why?

4. What is the running time of `remove` ()? Why?

## Implementation Details and Analysis:

➔ What is our resize strategy?

Array Resize Strategy #1:



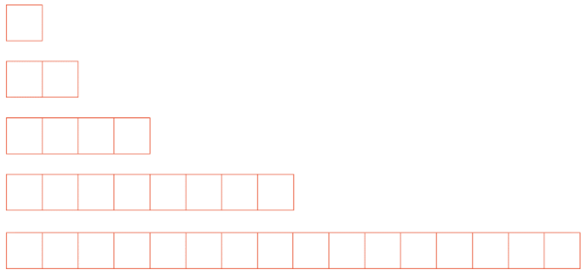
...total copies across all resizes: ___________

...total number of insert operations: ___________

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

What is our Big O runtime and amortized runtime?

Array Resize Strategy #2:



...total copies across all resizes: ___________

...total number of insert operations: ___________

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

What is our Big O runtime and amortized runtime?

**Running Time:**

| | Singly Linked List | Array |
|---|---|---|
| Look up **arbitrary** location | | |
| Insert after a **given** element | | |
| Remove after a **given** element | | |
| Insert at **arbitrary** location | | |
| Remove at **arbitrary** location | | |
| Search for an input **value** | | |

**Consider tradeoffs between data structures when deciding what to use! Can you think of some ways to improve some of the data structures seen today? What are the consequences?**