List ADT

Implementations

List Fixed Array Based

List Resize Array

Theoretical Algorithm Analysis

Count # of operations

Express # of crit. operations as a function of the size of the input (n = input size)

Average Case analysis: Best Case, Worst Case, Average Case
BC, WC, AC

Divide by # of existing cases

- Space Complexity (Amount of memory used)
- Time Complexity (efficiency)

List ADT
Implementations

Step 1: Choice of data structure

Step 2: choice of specific implementation using the chosen data structure.

Fixed Size Array Based (FAB)

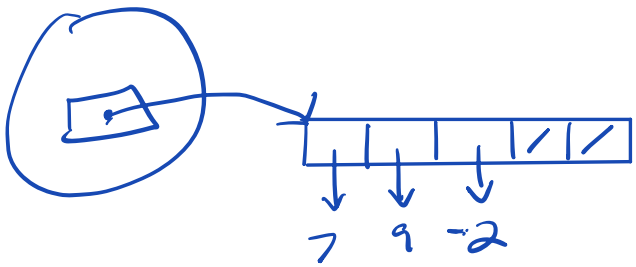
pros - Simple
DIA for get operation

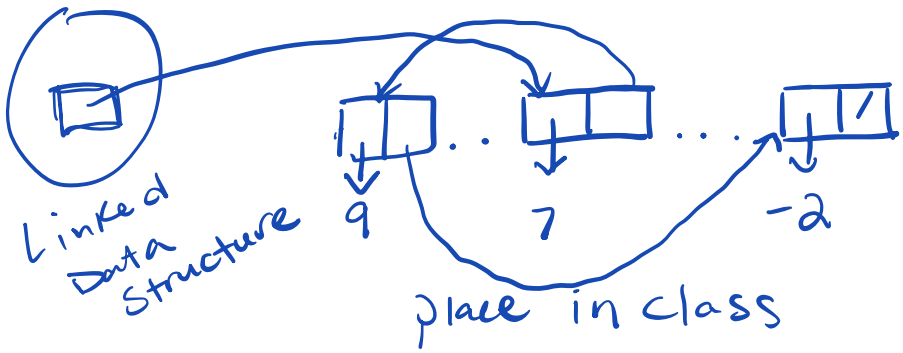
cons- Fixed size
Shifting (array entry copying) upon add/remove
Preallocation

Address Cons:

Preallocation:
Solution - change data structure

Fixed Size-
Sol: resize for larger storage





place in class

```
public class node
{
    private  Obj item;
    private  Node next;

    public Node (Object item)
    { this.item = item;
      next = null;
    }
```

```
public Node(Object item, Node next)
{
    this.item = item;
    this.next = next;
}
```

... getters & Setters

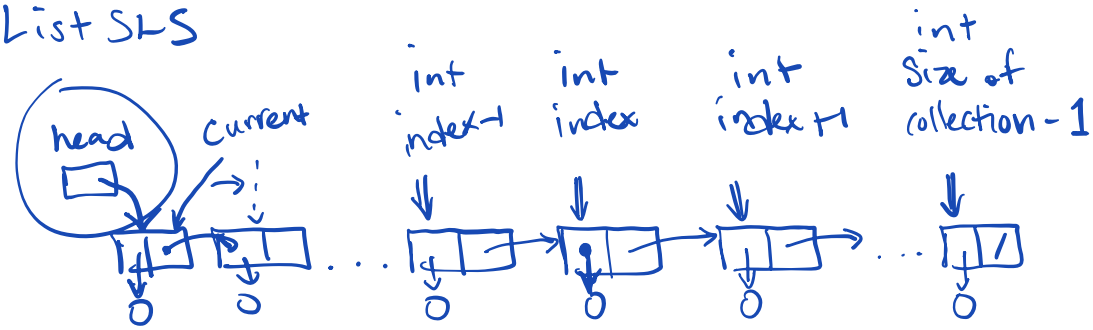
getItem, getNext, setItem (this.item), setNext

```
Public String toString()
{
    return item.toString();
}
```

}

Simply Linked Structure = SLS

List SLS



public obj get(int index)

remove

add

to get to next one..

```
Private Node find(int index)
 {Node current = head;
    for(int i=0; i<index; i++)
      {current = current.getNext(); }
    return current;
 }

  Public Object get(int index)
    { Object result;
      if...
   result = find(index).getItem();
      else...
      return result;
    }
```