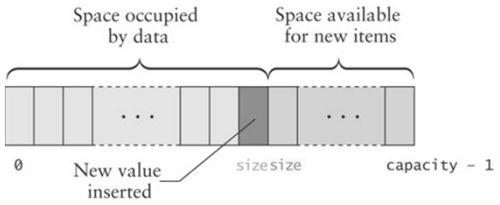
CSIS 3103

Chapter 2
ArrayList Implementation
Intro to Algorithm Analysis

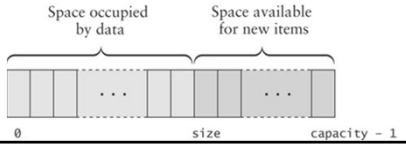
ArrayList.add(E anEntry)



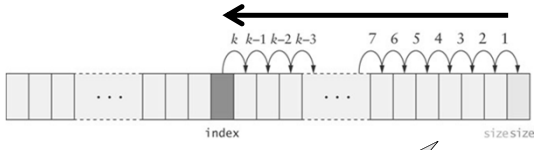
KWArrayList

A simple implementation of an `ArrayList` class

- Physical size of array indicated by data field `capacity`
- Number of data items indicated by the data field `size`

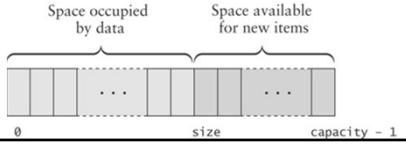


ArrayList.add(int index,E anEntry)

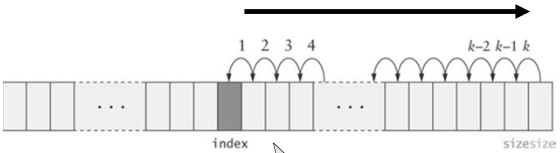


Shift values to the right of the insertion point to make room for new entry

```
public class KWArrayList <E> {
   /** The default initial capacity */
    private static final int INITIAL_CAPACITY = 10;

    /** The underlying data array */
    private E[] theData;

    /** The current size */
    private int size = 0;

    /** The current capacity */
    private int capacity = 0;
```



remove(int index)



Shift items on the right of the removal point to eliminate gap

Algorithm Analysis

A Big Problem?

A simple array and sequential search is never going to be good enough for this…

Swapping two values

```
// Swap contents of int variables
// x and y

if (x != y) {
   int temp = x;
   x = y;
   y = temp;
}
```

Worst case	
Lookups:	5
Assignments:	3
Comparisons:	1

How does this swap algorithm compare with others in terms of total work required?

The Questions

Is the program fast enough?
- Depends on purpose and expectations
- Depends on the **size of input data**

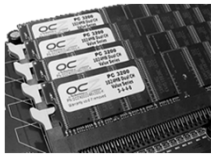
How much storage is needed?
- Depends on the **size of input data**

Will the solution scale as input gets *really* big?

Performance Analysis

What to Measure?

The time it takes an algorithm to execute



The space requirements of the data structure

Processors are fast, memory and disks are cheap, but…

Consider This

Scan a really big text file to find and print the 20 most frequently used words, together with counts of how often they occur.

Solution 1 (Donald Knuth): Uses heavy-duty data structures: Hash Trie implementation, randomized placement, pointers, several pages of code.

Now Consider This

Solution 2 (Doug McIlroy): UNIX shell script:

```
tr -c -s '[:alpha:]' '[\n*]' < FILE | \
sort | \
uniq -c | \
sort -n -r -k 1,1 | \
sed 20q
```

Which is better?
- Solution 1 is much faster
- But Solution 2 took 5 minutes to write and processes 20MB in 1 minute.

In many cases, anything will do, so *Keep It Simple*

Linear Growth Rate

```
public static int search(int[] x, int target) {
  for (int i=0; i<x.length; i++) {
    if (x[i]==target)
      return i;
  }
  return -1; // target not found
}
```

Measuring Execution Time

Experimental Study
- Write a program that implements the algorithm
- Run the program with various data sets
- Use `System.currentTimeMillis()` to get a measure of the actual running time

Advantages: Easy to measure, obvious meaning, appropriate where actual time is critical

n x m Growth Rate

```
public static boolean areDifferent(
                              int[] x, int[] y) {
  for (int i=0; i<x.length; i++) {
    if (search(y, x[i]) != -1)
      return false;
  }
  return true;
}
```

Limitation of Experimental Studies

- It is necessary to implement and test the algorithm to determine its running time.
- Experiments apply only to specific data sets and may not be indicative of the running time for other inputs
- Comparing two algorithms should be done in the same hardware and software environments

Quadratic Growth Rate

```
public static boolean areUnique(int[] x) {
  for (int i=0; i<x.length; i++) {
    for(int j=0; j<x.length; j++) {
      if (i != j && x[i] == j[i])
        return false;
    }
  }
  return true;
}
```