SDET UNIVERSITY

Algorithms

THINK LIKE A COMPUTER

THINK LIKE A COMPUTER

… make your computer think like a human analyzes

*Algorithms: a systematic approach to **methodically**, **efficiently** solve problems using repetition and calculation within complex data structures.*

It is the job of **COMPUTER SCIENTISTS** to handle complex algorithms and data structures efficiently.

COMMON ALGORITHMS

Linear Search – *finds element within a collection*
Binary Search – *finds element within a sorted collection*
Bubble Sort – *basic algorithm to sort an array*
Insertion Sort – *enhanced algorithm to sort an array*
Array List – *create a dynamic array*
Linked List – *node-reference value collection*
Stack – *LIFO collection (last in – first out)*
Queue – *FIFO collection (first in – first out)*

ALGORITHM EFFICIENCY

Time - *a measure of amount of time for an algorithm to execute.*

Space - *a measure of the amount of memory needed for an algorithm to execute.*

Complexity - *a study of algorithm performance*

"BIG O" NOTATION

O(1) - *describes an algorithm that will always execute in the same time regardless of the size of the input data set.*

O(N) - *describes an algorithm whose performance will grow linearly and in direct proportion to the size of the input data set.*

$O(N^2)$ - *represents an algorithm whose performance is directly proportional to the square of the size of the input data set.*

	Example

O(1)	`String state = “Florida”;`
	or
	`int number = 50/5;`

O(n)	`String[] names = { …. }`
	
	`for (String name : names) {`
	`    System.out.println(name);`
	`}`

$O(n^2)$	`Collection<String[]> users;`
	
	`for (String[] user : users) {`
	`   for (String field : user) {`
	`      System.out.println(“field”);`
	`   }`
	`}`

Linear Search

Tests each value of the collection for a match

Problem: Find the value / position of 4

15	62	24	4	...	13	75	71

Linear Search

Tests each value of the collection for a match

Problem: Find the value / position of 4

15	62	24	4	…	13	75	71
15 == 4?	62 == 4?	24 == 4?	4 == 4?				
			return 3 *or* return true				

Linear Search

```
void linearSearch(dataSet, target) {
  for (int n=0; n<dataSet.length; n++ ) {
    if (dataSet[n] == target} {
      return n;
      break;
    }
  }
}
```

Max Runtime:

Length of Data Structure

Big O Notation:

O(n)

Binary Search

Tests each value of a sorted collection for a match

Problem: Find the value / position of 15

4	13	15	24	62	71	75

Binary Search

Tests each value of a sorted collection for a match

Problem: Find the value / position of 15

4	13	15	24	62	71	75

Midpoint

Binary Search

Tests each value of a sorted collection for a match

Problem: Find the value / position of 15

4	13	15	24	62	71	75

15 > 24?

Binary Search

Tests each value of a sorted collection for a match

Problem: Find the value / position of 15

4	13	15	24	62	71	75

15 > 13?

Binary Search

Tests each value of a sorted collection for a match

Problem: Find the value / position of 15

4	13	15	24	62	71	75

15 > 15?
15 < 15?

FOUND
(15=15)

Binary Search

1. Assumes a sorted collection
2. “Divide and Conquer” Strategy
3. Implements a Recursive algorithm

Max Runtime:

Half the Length of
Data Structure

Big O Notation:

O(log n)

Bubble Sort

Sort the values in a collection

15 > 62?

15	62	24	4	...	13	75	71

Bubble Sort

Sort the values in a collection

62 > 24

15 62 24 4 ... 13 75 71

SWAP

Bubble Sort

Sort the values in a collection

62 > 4

15	24	62	4	...	13	75	71

SWAP

Bubble Sort

Sort the values in a collection

62 > .. ?

15	24	4	62	...	13	75	71

Bubble Sort

```
void bubbleSort(dataSet) {
boolean swapped=false;
  do {
    for (int n=0; n<data.length; n++ ) {
      if (dataSet[n] > dataSet[n+1]) {
        var temp=dataSet[i];
        dataSet[i] = dataSet[i+1];
        dataSet[i+1] = temp;
        swapped=true;
      }
    }
  } while(swapped)
}
```

Max Runtime:

Twice the Length of Data Structure

Big O Notation:

$O(n^2)$

ArrayList

Create a dynamic array

`array = new LinkedList()` `array.add(15)`	15		
`array.add(2)`	15	2	
`array.add(37)`	15	2	37

ArrayList

```
class ArrayList {
  private int[] array = new int[0];
  public void add(int value) { … }
  public void remove(int value) { … }
  private void expand() { … }
  public int get() { … }
  public int size() { … }
}
```

Characteristics:

The magic of the dynamic array list is the algorithm in add() and remove()

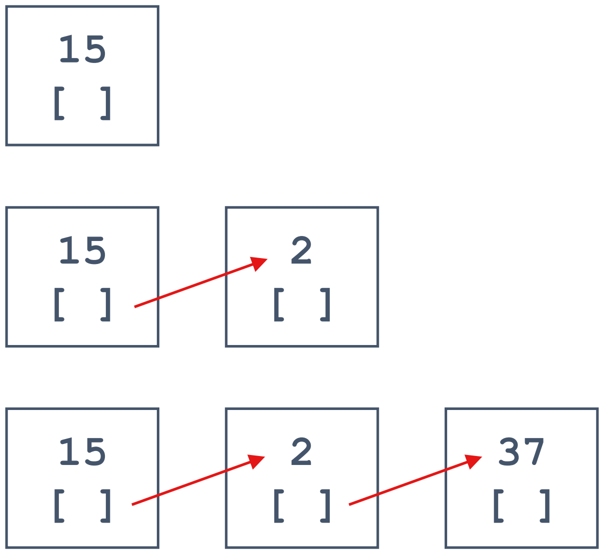
LinkedList

Create a dynamic list with node references

```
array = new LinkedList()
array.add(15)

array.add(2)

array.add(37)
```



LinkedList

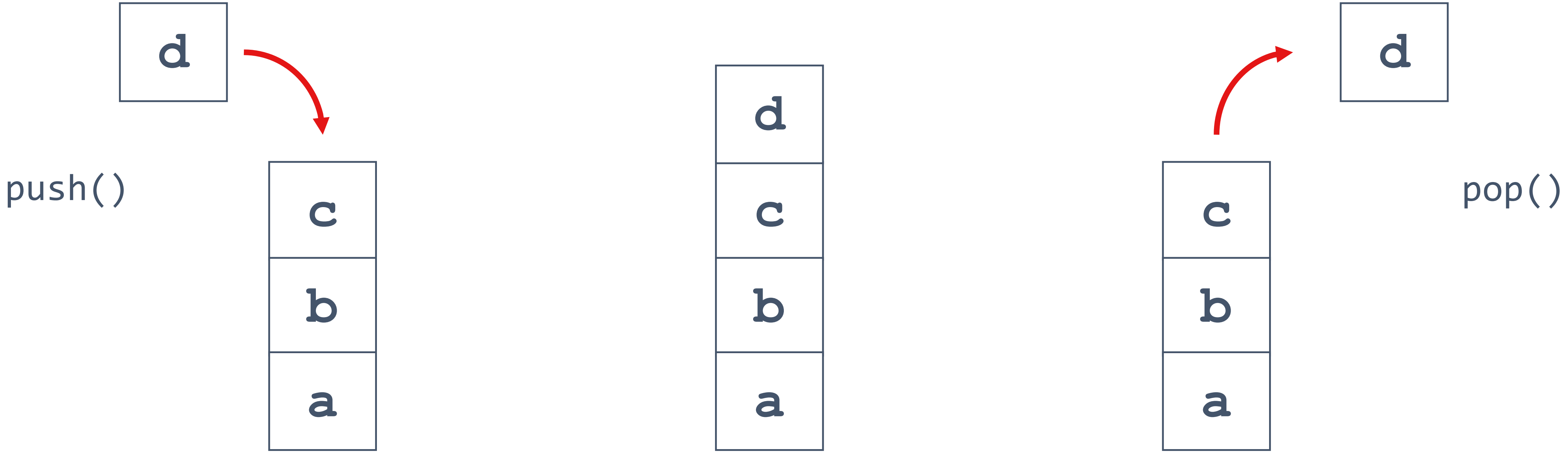
```
class LinkedList {
  public add(int value) {
  Node node = new Node(value)
class Node {
  int value;
  Node node;
}
```

Characteristics:

The magic of the dynamic array list is the algorithm in add() and remove()

Stack

A list-based collection whose accessible elements are at the end (elements are added and removed at the end).



Stack

```
class Stack {
  public void push(int value) { … }
  public int pop() { … }
  public int peek() { … }
}
```

Characteristics:

The Stack implements a **LIFO** Last-In First-Out data structure

Queue

A list-based collection whose elements are added at the beginning and removed at the end.

Queue

```
class Queue {
  public void enqueue(int value) { … }
  public void dequeue() { … }
  public int examine(int index) { … }
  public int size() { … }
}
```

Characteristics:

The Queue implements a **FIFO** First-In First-Out data structure