Katherine Ayers
Code Nation Masters 38

Algorithms Analysis

Algorithm Analysis with Kat

- A Brief introduction
- Sorting algorithms
- Big-O notation
- Leaning can be fun
- Q & A

Katherine Ayers
Code Nation Masters 38

Algorithms Analysis

Introduction

We are learning to create algorithms as junior developers

Algorithms produce the result we want.

Common algorithms we have created are

- **Storing algorithms**
- **Searching algorithms**
- **Sorting algorithms**

When two programs solve the same problem but look different, which one is better?

Sorting Algorithms

Sorting Algorithms

- Bubble Sort
- Selection Sort
- Insertion Sort
- Sorted() - comparison

Sorting Algorithms

Katherine Ayers
Code Nation Masters 38

BubbleSort

86	25	24	52	10	99	68	44	67	5
25	86	24	52	10	99	68	44	67	5
25	24	86	52	10	99	68	44	67	5
25	24	52	86	10	99	68	44	67	5
25	24	52	10	86	99	68	44	67	5
25	24	52	10	86	99	68	44	67	5
25	24	52	10	86	68	99	44	67	5
25	24	52	10	86	68	44	99	67	5
25	24	52	10	86	68	44	67	99	5
25	24	52	10	86	68	44	67	5	99

Sorting Algorithms

Katherine Ayers
Code Nation Masters 38

Selection Sort

86	25	24	52	10	99	68	44	67	5
86	25	24	52	10	5	68	44	67	99
67	25	24	52	10	5	68	44	86	99
67	25	24	52	10	5	44	68	86	99
44	25	24	52	10	5	67	68	86	99
44	25	24	5	10	52	67	68	86	99
10	25	24	5	44	52	67	68	86	99
10	5	24	25	44	52	67	68	86	99
10	5	24	25	44	52	67	68	86	99
5	10	24	25	44	52	67	68	86	99

Sorting Algorithms

Katherine Ayers
Code Nation Masters 38

Insertion Sort

86	25	24	52	10	99	68	44	67	5
25	86	24	52	10	99	68	44	67	5
24	25	86	52	10	99	68	44	67	5
24	25	52	86	10	99	68	44	67	5
10	24	25	52	86	99	68	44	67	5
10	24	25	52	86	99	68	44	67	5
10	24	25	52	68	86	99	44	67	5
10	24	25	44	52	68	86	99	67	5
10	24	25	44	52	67	68	86	99	5
5	10	24	25	44	52	67	68	86	99

Algorithms Analysis

Katherine Ayers
Code Nation Masters 38

Python

- I am going to use Python to demonstrate some sorting algorithms.
- I am using Python because it is the programming language I am most familiar with.
- The same logic can be applied to any algorithm.

Katherine Ayers
Code Nation Masters 38

Algorithm Efficiency

Big-O notation

When trying to characterise an algorithms efficiency in terms of execution, it is important to quantify the number of steps an algorithm will take.

An algorithms execution time changes with respect to the size of the problem.

A basic unit of computation for computing is to count the number of steps that are performed

T(n) = 1 + n
"n" is the size of the problem
T(n) is the time taken to solve the problem

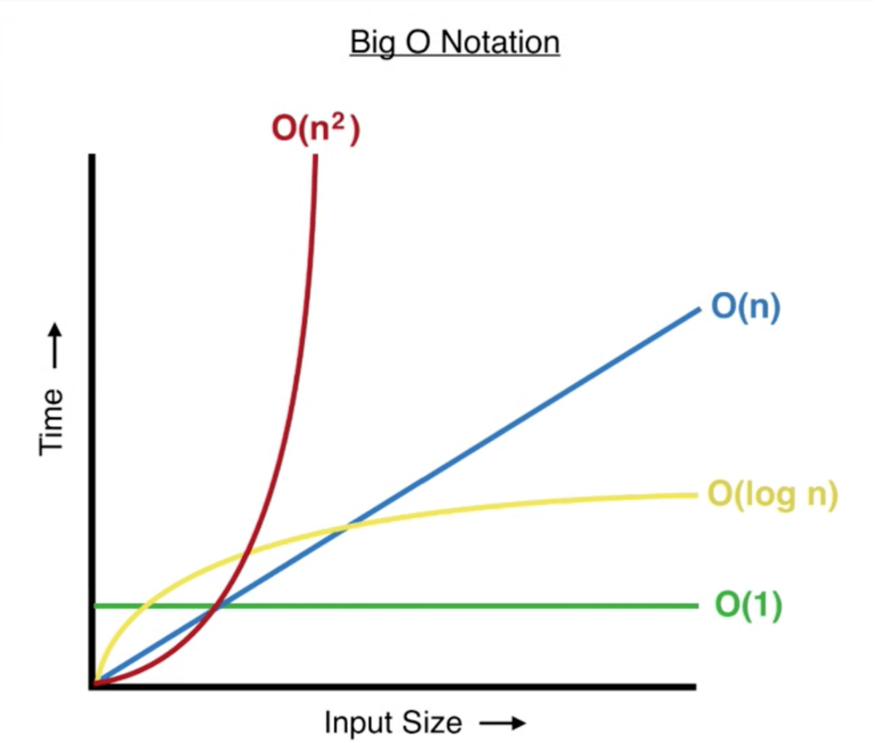
Katherine Ayers
Code Nation Masters 38

Algorithm Efficiency

Big-O notation

Order of magnitude functions "O(f(n))", often called Big-O notation ("O" for order)

f(n)	Name
1	Constant
log n	Logarithmic
n	Liniear
n log n	Log Linear
n^2	Quadratic
n^3	Cubic
2n	Exponential



Katherine Ayers
Code Nation Masters 38

Big-O notation

Big-O Bubble Sort

```
1   def bubbleSort(aList):
2       for passNumber in range (len(aList)-1,0,-1): # n(
3           for i in range(passNumber):              # n(
4               if aList[i] > aList [i + 1]:         # 1 +
5                   temp = aList[i]                  # 1 +
6                   aList[i] = aList[i +1]           # 1 +
7                   aList[i +1] = temp               # 1)
8       return aList
9
10  # T(n) = n(n(1 + 1 + 1 + 1))
11  # T(n) = n(n(4))
12  # T(n) = 4n^2
13  # O(n) = n^2
```

```
def bubbleSort(aList):
    for passNumber in range (len(aList)-1,0,-1):        n(
        for i in range(passNumber):                     n(
            if aList[i] > aList [i + 1]:                1 +
                temp = aList[i]                         1 +
                aList[i] = aList[i +1]                  1 +
                aList[i +1] = temp                       1)
    return aList
```

= n(n(1 + 1 + 1+ 1)

= n(n)

= n^2

Oder of Magnitude Big-O Highest Values wins: O(n^2)

It all depends on the value of n

Katherine Ayers
Code Nation Masters 38

Big-O notation

Selection Sort Bubble Sort

```
1   def selectionSort(aList):
2       for endSlot in range (len(aList) -1, 0, -1):            # n(
3           positionOfMax = 0                                   # 1 +
4           for location in range(1, endSlot + 1):              # n(
5               if aList[location] > aList[positionOfMax]:      # 1 +
6                   positionOfMax = location                    # 1 +
7           temp = aList[endSlot]                               # 1 +
8           aList[endSlot] = aList[positionOfMax]               # 1 +
9           aList[positionOfMax] = temp                         # 1 ))
10      return aList
11
12  #  T(n) = n(1 + n( 1 + 1 + 1 + 1 + 1))
13  #  T(n) = n(1 + n(5))
14  #  T(n) = n(n(5))
15  #  T(n) = 5n^2
16  #  O(n) = n^2
```

```
def selectionSort(aList):
    for endSlot in range (len(aList) -1, 0, -1):                n(
        positionOfMax = 0                                       1 +
        for location in range(1, endSlot + 1):                  n(
            if aList[location] > aList[positionOfMax]:          1 +
                positionOfMax = location                        1 +
        temp = aList[endSlot]                                   1 +
        aList[endSlot] = aList[positionOfMax]                   1 +
        aList[positionOfMax] = temp                             1))
    return aList
```

= n(1 + n(1 + 1 + 1+ 1+ 1))

= n(n)

= n^2

Oder of Magnitude Big-O Highest Values wins: O(n^2)

It all depends on the value of n

Katherine Ayers
Code Nation Masters 38

Big-O notation

Insertion Sort Bubble Sort

```python
def insertionSort(aList):
    for index in range(1, len(aList)):                          # n(
        currentValue = aList[index]                             # 1 +
        position = index                                        # 1 +
        while position > 0 and aList[position] > currentValue:  # n(
            aList[position] = aList[position - 1]               # 1 +
            position = position -1                              # 1 +
        aList[position] = currentValue                          # 1))
    return aList

# T(n) = n(1 + 1 + n(1 + 1 + 1))
# T(n) = n(2 + n(3))
# T(n) = n(2 + 3n)
# T(n) = 2n + 3n^2
# O(n) = n^2
```

```
def insertionSort(aList):
    for index in range(1, len(aList)):                          n(
        currentValue = aList[index]                             1 +
        position = index                                        1+
        while position > 0 and aList[position] > currentValue:  n(
            aList[position] = aList[position - 1]               1+
            position = position -1                              1+
        aList[position] = currentValue                          1))
    return aList
```

T(n) = n(1 + 1 + n(1+ 1 + 1))

T(n) = n(n)

T(n) = n^2

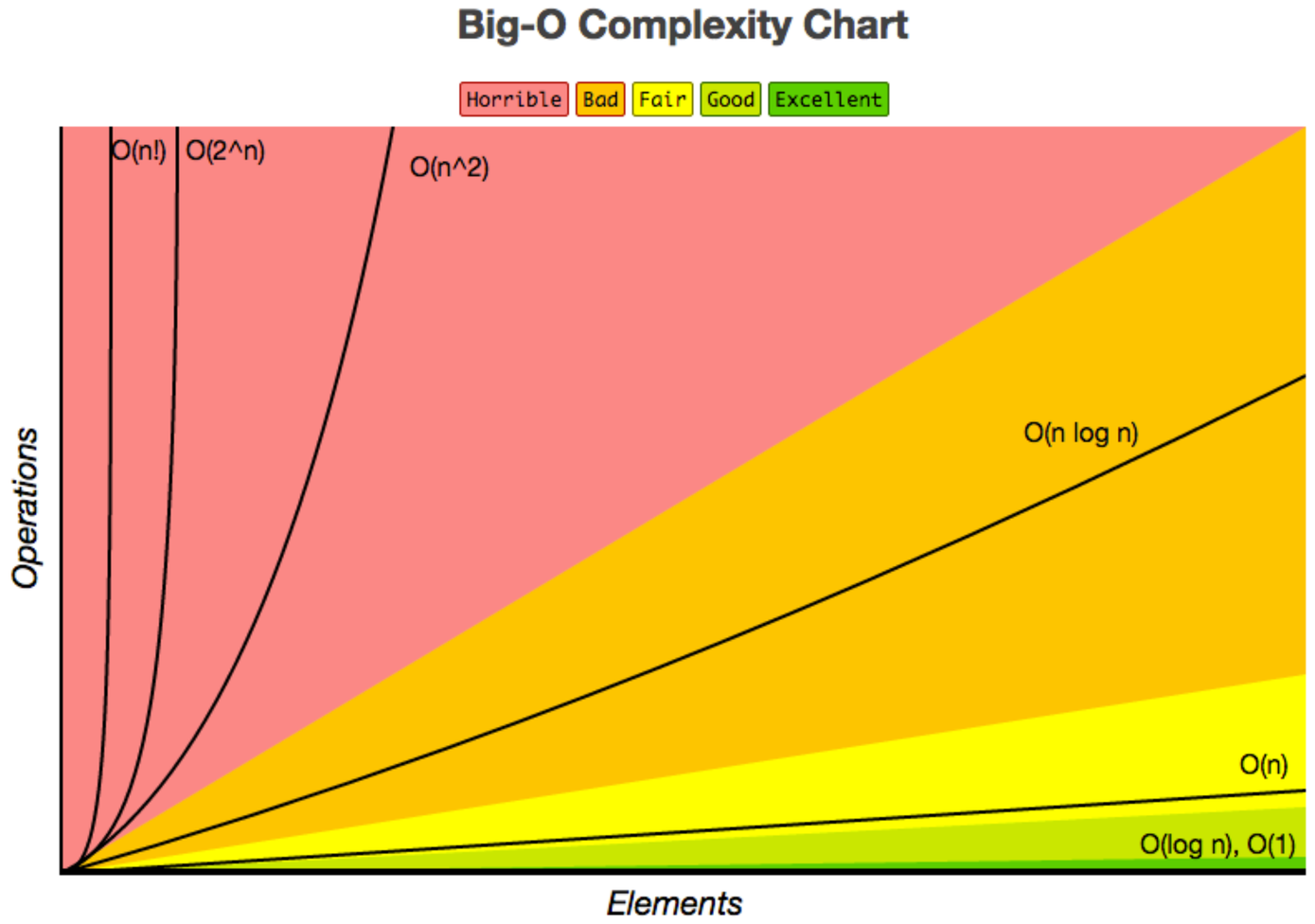
Oder of Magnitude Big-O Highest Values wins: O(n^2)

It all depends on the value of n

Algorithm Complexity

Katherine Ayers
Code Nation Masters 38

Big-O Complexity



Algorithm Complexity

Katherine Ayers
Code Nation Masters 38

Best Case / Worst Case

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Algorithm Complexity

Katherine Ayers
Code Nation Masters 38

Common Data Structure

Common Data Structures

Data structures are the key in some of the most used algorithms and a good data structure is sometimes the way to achieve a very efficient and elegant solution.

Data Structure	Time							
	Average				Worst			
	Indexing	Search	Insertion	Deletion	Indexing	Search	Insertion	Deletion
Basic Array	O(1)	O(n)	-	-	O(1)	O(n)	-	-
Dynamic Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
Single-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Double-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Skip List*	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)
Hash Table	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)
Ternary Search Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)
Cartesian Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Splay Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))

Algorithm Complexity

Katherine Ayers
Code Nation Masters 38

Sorting

Sorting

Very important and useful type of algorithms for many common problems nowadays, Finder, iTunes, Excel and many more programs use some kind of sorting algorithm.

Algorithm	Time			Auxiliary Space
	Best	Average	Worst	Worst
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Mergesort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$
Quicksort	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(\log(n))$
Heapsort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(1)$
Shell Sort $(h=2^k-1)$	$O(n \log(n))$	$O(n^{1.25})$ conjectured	$O(n^{(3/2)})$	$O(1)$
Bucket sort	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(nk)$
Radix sort	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$

Algorithm Complexity

Katherine Ayers
Code Nation Masters 38

Searching

Searching

Searching efficiently is crucial for many operations in different parts of the software we use on a daily basis.

Algorithm	Data Structure	Time		Space
		Average	Worst	Worst
Depth First Search (DFS)	Graph of |V| vertices and |E| edges	-	O(|E| + |V|)	O(|V|)
Breadth First Search (BFS)	Graph of |V| vertices and |E| edges	-	O(|E| + |V|)	O(|V|)
Binary Search	Sorted array of n elements	O(log(n))	O(log(n))	O(1)
Linear (Brute force)	Array of n elements	O(n)	O(n)	O(1)
Shortest path by Dijkstra, Min-heap as priority queue	Graph of |V| vertices and |E| edges	O((|V|+|E|) log|V|)	O((|V|+|E|) log|V|)	O(|V|)
Shortest path by Dijkstra, unsorted array as priority queue	Graph of |V| vertices and |E| edges	$O(\|V\|^2)$	$O(\|V\|^2)$	O(|V|)
Shortest path by Bellman-Ford	Graph of |V| vertices and |E| edges	O(|V||E|)	O(|V||E|)	O(|V|)

Algorithm Complexity

Katherine Ayers
Code Nation Masters 38

Heaps

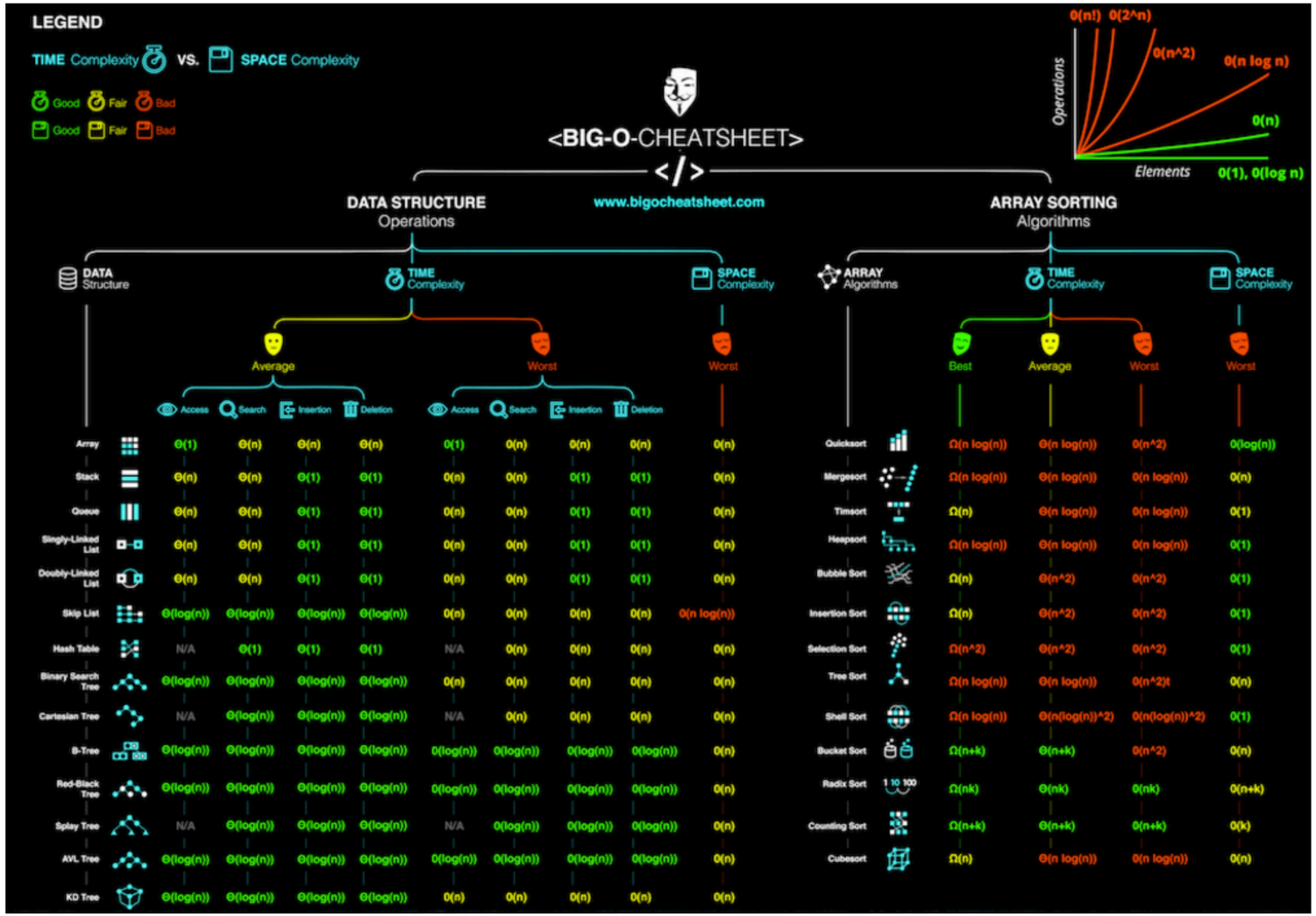
Heaps

Heaps	Time						
	Heapify	Find Max	Extract Max	Increase Key	Insert	Delete	Merge
Linked List	-	O(1)	O(1)	O(n)	O(n)	O(1)	O(m+n)
Linked List (unsorted)	-	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
Binary Heap	O(n)	O(1)	O(n log(n))	O(n log(n))	O(n log(n))	O(n log(n))	O(m+n)
Binomial Heap	-	O(n log(n))	O(n log(n))	O(n log(n))	O(n log(n))	O(n log(n))	O(n log(n)
Fibonnaci Heap	-	O(1)	O(log(n))*	O(1)*	O(1)	O(log(n))*	O(1)

Algorithm Complexity

Katherine Ayers
Code Nation Masters 38

Big-O Cheat Sheet



Algorithm Game

Katherine Ayers
Code Nation Masters 38

Fun!

Algorithm Game

Katherine Ayers
Code Nation Masters 38

Human Resource Machine

Katherine Ayers
Code Nation Masters 38

Algorithm Game

7 Billion Humans

The End

Katherine Ayers
Code Nation Masters 38

Thank you for watching!

Any Questions? ...