THINK BIG WE DO™

CSC 212

Data Structures & Algorithms

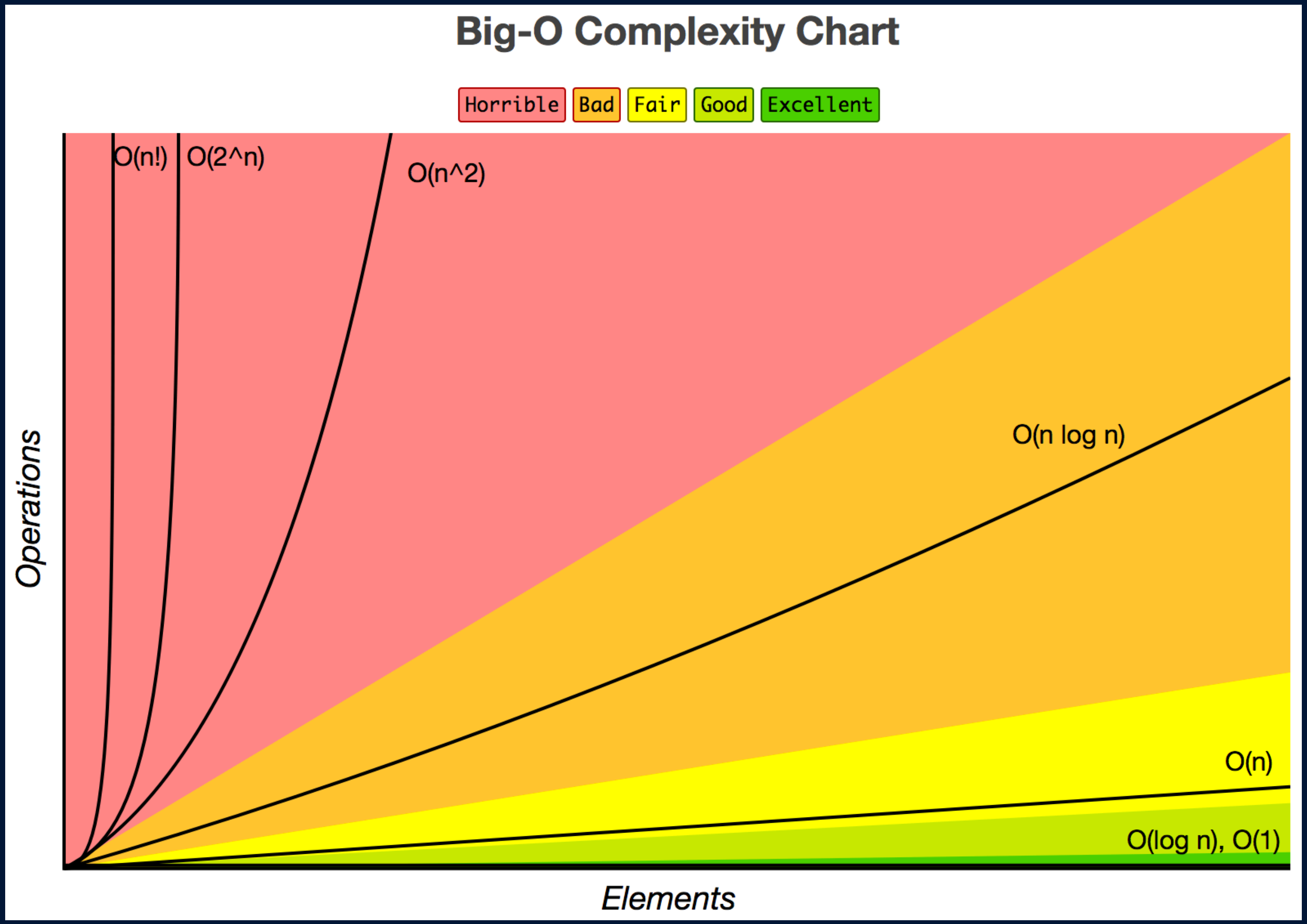
Fall 2022 | Jonathan Schrader

Computational Cost

Housekeeping

- Assignment 1 Due Friday
- Review Project

Computational Cost



$O(1)$

Constant Time Algorithms

```
// COMPLEXITY OF CONSECUTIVE STATEMENTS
int main () {
  int x=2, y=3;
  int z=x+y;
  cout << z;
}
```

- $O(1)$ is also called as constant time, it will always execute in the same time regardless of the input size.

$O(LOG\ N)$

Logarithmic Time Algorithms

Binary Search

```
int main () {
  for (int i = 1; i < n; i = i * 2) {
    cout << "Count: " << i << endl;
  }
}
```

- $O(log\ n)$ function the complexity increases as the size of input increases.

$O(N)$

Linear Time Algorithms

```
// COMPLEXITY OF A SIMPLE LOOP
//    Time complexity of a loop can be determined by running
//    time of statements inside loop multiplied by total number
//    of iterations.
int main () {
  int n = 5;
  for (int i = 1; i <= n; i++ >) {
    cout << i << " " << endl;
  }
}
```

- O(n) is also called as linear time, it is direct proportion to the number of inputs. For example, if the array has 6 items, it will print 6 times.

Note: In $O(n)$ the number of elements increases, the number of steps also increases.

$O(NLOGN)$

Linear Logarithmic Time Algorithms

Merge sort / Quicksort

```
// slide 2
int main () {
  for (int i = 1; i < n; i = i++) {
    for (int j = 1; j < n; j = j * 2) {
      cout << "Count: " << i << " and " << j << endl;
      // sum = sum * j;
    }
  }
}
```

The $O(n\ log\ n)$ function fall between the linear and quadratic function (i.e $O(n)$ and $\mathrm{O}(n^2)$. It is mainly used in sorting algorithm to get good Time complexity.

$O(N^2)$

Polynomial-Time Algorithms

```
// COMPLEXITY OF A NESTED LOOP
//    It is product of iterations of each loop.
int main () {
  int n = 3;
  for (int i = 1; i < n; i++ >) {
      for (int i = 1; i < n; i++ >) {
        cout << i << ", " << j << endl;
      }
  }
}
```

$O(2^N)$

Exponential Time Algorithms

```
int main () {
  for (int i = 1; i <= power(2, n); i++) {
    cout << "Count " << i << endl;
  }
}
```

- Algorithms with complexity $O(2^N)$ are called as Exponential Time Algorithms
- These algorithms grow in proportion to some factor exponentiated by the input size
- Example
  - $O(2^N)$ algorithms double with every additional input. So, if $N = 2$, these algorithms will run four times; if $N = 3$, they will run eight times (kind of like the opposite of logarithmic time algorithms)
- $O(3^N)$ algorithms triple with every additional input, $O(k^N)$ algorithms will get k times bigger with every additional input

$O(N!)$

Factorial Time Algorithms

```
int main () {
  unsigned int fact(unsigned int n) {
    if (n == 0)
      return 1
    return n * fact(n - 1);
  }
}
```

- This class of algorithms has a run time proportional to the factorial of the input size
- A classic example of this is solving the traveling salesman problem using a brute-force approach to solve it

COMPLEXITY OF IF AND $ELSE$ BLOCK

```
int main () {
  int countOfEven = 0;
  int countOfOdd = 0;
  int k = 0;
  for (int i = 0; i < n; i++) {
    if (i % 2 == 0) {
      countOfEven++;
      k = k + 1;
    } else {
      countOfOdd++;
    }
  }
}
```

- When you have if and else statement, then time complexity is calculated with whichever of them is larger