

CSC 212: Data Structures and Abstractions

Computational Cost

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

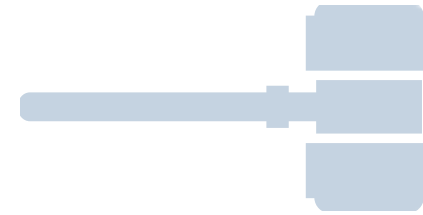
Fall 2020



Quick notes

- Attending lectures and labs
 - ✓ **take notes** (by hand or typing) — avoid just taking screenshots
 - ✓ be **ready to answer** questions
- Assignment 1
 - ✓ **autograder** activates tomorrow to accept submissions
 - ✓ use **double** variables in all calculations
 - ✓ the **median** of a list of even length is the average of the two elements in the middle
 - ✓ include **pixel i, j** as part of the local neighborhood
 - ✓ when calculating standard deviation, divide by **n**

Analyzing running time



Empirical Analysis

- ✓ **Run** algorithm
- ✓ Measure actual time

Mathematical Model

- ✓ **Analyze** algorithm
- ✓ Develop Model

Theoretical Models

Mathematical model

- High-level analysis — **no need to implement**
- Independent of HW / SW
- Based on **counts** of basic **instructions**
 - ✓ additions, multiplications, comparisons, etc
 - ✓ exact definition not important but **must be relevant** to the problem

Basic assumptions

- In order to use a **formal framework**, we will make certain assumptions
 - ✓ count basic instructions: additions, multiplications, comparisons, assignments
 - ✓ each instruction takes one time unit
 - ✓ instructions are executed sequentially
 - ✓ infinite memory
- Focus on **analyzing running time**

Example

- What to count?
 - ✓ only relevant instructions? all instructions?

```
double sum = 1;  
for (int i = 0 ; i < n ; i ++ ) {  
    sum = sum * i;  
}
```

lets also plot both cases ...

Example

```
for (int i = 0 ; i < n ; i ++ ) {  
    for (int j = 0 ; j < n ; j ++ ) {  
        sum = sum * j ;  
    }  
}
```


Example

```
for (int i = 0 ; i < n ; i ++ ) {  
    for (int j = 0 ; j < n*n ; j ++ ) {  
        sum = sum * j ;  
    }  
}
```

Example

```
for (int i = 0 ; i < n ; i ++ ) {  
    for (int j = 0 ; j < i ; j ++ ) {  
        sum = sum * j ;  
    }  
}
```

Example

```
for (int i = 0 ; i < n ; i ++ ) {  
    for (int j = 0 ; j < n ; j ++ ) {  
        for (int k = 0 ; k < n ; k ++ ) {  
            // count 1 instruction  
        }  
    }  
}
```

Example

```
for (int i = 0 ; i < n ; i ++ ) {  
    for (int j = 0 ; j < i*i ; j ++ ) {  
        for (int k = 0 ; k < j ; k ++ ) {  
            // count 1 instruction  
        }  
    }  
}
```

Some rules ...

- Single loops
 - ✓ essentially the number of iterations times the number of instructions performed at each iteration
- Nested loops
 - ✓ count instructions inside out
 - ✓ careful with the range of the loop
 - ✓ when possible, multiplications can be used for counts from each loop
- Consecutive statements
 - ✓ just add the counts
- Conditionals
 - ✓ consider the branch with the highest count

Computational cost

- Number of **basic instructions** required by the algorithm to process an input of a certain **size n**

$$T(n)$$

- ✓ basic instructions are always **relevant** to the problem
 - ex: find max in an array
 - # of comparisons
 - ex: sum elements in an array
 - # of additions

Comparing computational cost

find (x,y,z) s.t. x+y+z=k

find (x,y) s.t. x+y=k

find x=k

Size of Input	n^3	n^2	n
$n = 1$	1	1	1
$n = 10$	1,000	100	10
$n = 100$	1,000,000	10,000	100
$n = 1000$	1,000,000,000	1,000,000	1,000
$n = 10000$	1,000,000,000,000	100,000,000	10,000
$n = 100000$	1,000,000,000,000,000	10,000,000,000	100,000
$n = 1000000$	1,000,000,000,000,000,000	1,000,000,000,000	1,000,000
$n = 10000000$	1,000,000,000,000,000,000,000	100,000,000,000,000	10,000,000

Growth Rate

n	$\log \log n$	$\log n$	n	$n \log n$	n^2	n^3	2^n
16	2	4	2^4	$4 \cdot 2^4 = 2^6$	2^8	2^{12}	2^{16}
256	3	8	2^8	$8 \cdot 2^8 = 2^{11}$	2^{16}	2^{24}	2^{256}
1024	≈ 3.3	10	2^{10}	$10 \cdot 2^{10} \approx 2^{13}$	2^{20}	2^{30}	2^{1024}
64K	4	16	2^{16}	$16 \cdot 2^{16} = 2^{20}$	2^{32}	2^{48}	2^{64K}
1M	≈ 4.3	20	2^{20}	$20 \cdot 2^{20} \approx 2^{24}$	2^{40}	2^{60}	2^{1M}
1G	≈ 4.9	30	2^{30}	$30 \cdot 2^{30} \approx 2^{35}$	2^{60}	2^{90}	2^{1G}

growth of $T(n)$ as $n \rightarrow \infty$