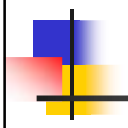
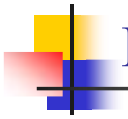


Time Complexity

Data Structures
Mr. Khaleifah Al.jada'
Al-Majma'ah Community College



Efficiency of programs

- What does it mean when we say that a program is efficient?
- First note that
  - Programs solve **problems**
  - There could be several programs that solve the same problem
  - We would probably want the **correct** program that runs **fastest**
- Efficient ⇔ Fast

Measuring efficiency

- How **fast** is the following code segment?

```
int sum = 0;
for( int i = 0; i < n; i++ )
{
   sum = sum + numbers[i];
}
System.out.println( sum );
```

- Unit of measurement
  - milliseconds? machine cycles?
  - statements? operations?
- The answer will have to be a function on n, T(n), or the **time complexity** of the program

Time Complexity

- T(n): time complexity or running time of a program
- n is typically the size of the program's input
- It is not practical to count milliseconds (would depend a lot on machine speed)
- Makes more sense to count operations
- But which operations?

Counting operations

- Back to example

```
int sum = 0;
for( int i = 0; i < n; i++ )
{
   sum = sum + numbers[i];
}
System.out.println( sum );
```

- If we counted
  - assignments: T(n) = n+2
  - additions: T(n) = n
  - assignments and additions: T(n) = 2n+2
  - assignments,additions, and increments: T(n) = 3n+2
  - ...

Eliminating arbitrariness

- What is important is to have a fair assessment on time complexity that captures the program's efficiency
- For the example, we say that **T(n) is O(n)** or **T(n) = O(n)**
  - "Big-Oh notation", "Order Arithmetic"
  - Meaning: the program runs in time proportional to n
    - Just drop the constant factors and less dominant terms
  - We make the same statement regardless of the operations that we decide to count

Time complexity analysis

- Analyzing time complexity allows us to
  - Categorize programs/algorithms according to their efficiency
  - Compare programs and possibly conclude that one is more efficient than the other
- Suppose
  - there are two programs A and B
  - A has time complexity $T_1(n)$ and $T_1(n) = O(n)$
  - B has time complexity $T_2(n)$ and $T_2(n) = O(n^2)$
- Conclusion: A is more efficient than B

If we wanted to be precise…

- Suppose
  $T_1(n) = 8n + 3$
  $T_2(n) = n^2 - n + 1$
- Tabulate and compare the time complexities for n = 1,2,3,…
- **A is faster than B** (as long as n >= 10)
  B is faster than A (when n < 10)
- The first statement is more meaningful
  - But better to preface it with "**for large n**,"…

Some categories

- O(1): constant time
- O(log n): logarithmic time
- O(n): linear time
- O(n log n)
- $O(n^2)$: quadratic time
- $O(n^k)$, for some fixed k: polynomial time
- $O(2^n)$: exponential time

Plot the functions and observe **growth rates**

Some familiar examples

- Searching
  - Linear search runs in O(n) time
  - Binary search runs in O(log n) time
- Sorting
  - Bubble sort runs in $O(n^2)$ time
  - Later, we will discuss an O(n log n) algorithm for sorting

Solution 1

```
for( int i = 0; i < n; i++ )
{
    double sum = 0;
    for( int j = 0; j <= i; j++ )
    {
        sum = sum + numbers[j];
    }
    means[i] = sum / (i+1);
}
```


Solution 2

```
double sum = 0;
for( int i = 0; i < n; i++ )
{
    sum = sum + numbers[i];
    means[i] = sum / (i+1);
}
```

Class Exercise

- Compute $T_1(n)$ and $T_2(n)$, the time complexities for the two solutions
  - Count arithmetic operations (additions, increments, divisions)
- Assess these algorithms
  - $T_1(n)$ is O( ) ?
  - $T_2(n)$ is O( ) ?

Summary

- The time complexity of an algorithm measures its efficiency
  - T(n), where n is the size of the input
  - There are many ways to compute T(n)
- Big-Oh notation provides a way to assess the efficiency of a program/algorithm
  - Regardless of what operations are counted
- More in the CS 110 (Data Structures) course
  - Formal definition of Big-Oh notation
  - There is also *space* complexity