

## Runtime Complexity

### Graph Traversals

### Topological Sort

## Computational Complexity



### Useful notation to discuss growth rates

For any monotonic functions  $f, g$  from the positive integers to the positive integers, we say

if  $f(n) = O(g(n))$   
 $g(n)$  eventually dominates  $f(n)$

Formally: there exists a constant  $c$  such that for all sufficiently large  $n$ :  $f(n) \leq c \cdot g(n)$

### Another useful notation: $\Omega$

For any monotonic functions  $f, g$  from the positive integers to the positive integers, we say

if:  $f(n) = \Omega(g(n))$   
 $f(n)$  eventually dominates  $g(n)$

Formally: there exists a constant  $c$  such that for all sufficiently large  $n$ :  $f(n) \geq c \cdot g(n)$

### More useful notation: $\Theta$

For any monotonic functions  $f, g$  from the positive integers to the positive integers, we say

if:  $f(n) = \Theta(g(n))$   
 $f(n) = O(g(n))$  and  $f(n) = \Omega(g(n))$

In this class we will be mostly concerned with a big-O notation.

$$T(n) = n \log n + 1024 n \log (\log n)$$

Circle **ALL** answers that apply.

Is  $T(n) = O(n^2)$ ?

$O(n^2)$	$O(n \log n)$	$O(n \log \log n)$	$O(n)$	$O(\log n)$
----------	---------------	--------------------	--------	-------------

$$T(n) = n \log n + 1024 n \log (\log n)$$

Circle **ALL** answers that apply

$\Omega(n^2)$	$\Omega(n \log n)$	$\Omega(n \log \log n)$	$\Omega(n)$	$\Omega(\log n)$
---------------	--------------------	-------------------------	-------------	------------------

$$T(n) = n \log n + 1024 n \log (\log n)$$

Circle **ALL** answers that apply

$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log \log n)$	$\Theta(n)$	$\Theta(\log n)$
---------------	--------------------	-------------------------	-------------	------------------

What is the Big-O runtime complexity of the following function?

```
int bigOh1(int n):
  int value = 0;
  for i=0 to n
    for j=0 to length(binary(n))
      value = i * j;
  return value;
```

converts decimal into binary

$O(n \log n)$

What is the Big-O runtime complexity of the following function?

```
string bigOh2(int n):
  string tmp = "";
  for k=0 to n
    tmp = tmp + toString(k);
  return tmp;
```

+ is a concatenation

$O(n^2)$

What is the Big-O runtime complexity of the following function?

```
void bigOh3(int n):
  for i=1 to n
    j=1;
    while j < n
      j = j*2;
```

$O(n \log n)$

What is the Big-O runtime complexity of the following function?

```
void bigOh4(int[] L, int n)
  while (n > 0)
    find_max(L, n); //finds the max in L[0...n-1]
    n = n/4;
```

$O(n \log n)$

$$n + n/4 + n/16 + \dots + 1 =$$

$$= n (1 + 1/4 + 1/16 + \dots) = \Theta(n)$$

What is the Big-O runtime complexity of the following function?

```
string bigOh5(int n)
    if(n == 0) return "a";
    string str = bigOh5(n-1);
    return str + str;
```

**HARD - !!!**

bigOh5(0) → "a"  
bigOh5(1) → "aa"  
bigOh5(2) → "aaaa"  
bigOh5(3) → "aaaaaaaa"

$$1 + 2 + 4 + 8 + 16 + \dots = O(2^n)$$

What is the Big-O runtime complexity of the following function?

```
void bigOh6(int n)
    for i = 1 to n
        for j = i to n
            sum = 0;
            for k = i to j
                sum += A[k];
            B[i, j] = sum;
```

$O(n^3)$

Sum[1, {i,1,n}, {j,i,n}, {k,i,j}] =

$$n(n+1)(n+2)/6 = \Theta(n^3)$$

See the next slide

Triple summation in bigOh6

$$\sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^j 1 = \sum_{i=1}^n \sum_{j=i}^n (j-i+1) = \sum_{i=1}^n \left( \sum_{j=i}^n j - \sum_{j=i}^n (i-1) \right) = \sum_{i=1}^n \left( \frac{(n+1-i)(n+2-i)}{2} \right) = \frac{n(n+1)(n+2)}{6}$$

BFS and DFS



## Graph Traversals

Depth-First Search (DFS)  
Breadth-First Search (BFS)

DFS uses a **stack** for bookkeeping.

BFS uses a **queue** for bookkeeping.

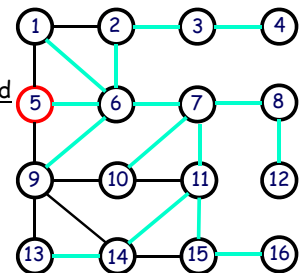
## Properties of Traversals


### Property 1

They visit all the vertices in the connected component

### Property 2

The result of traversal is a **spanning tree** of the connected component





### Exercise

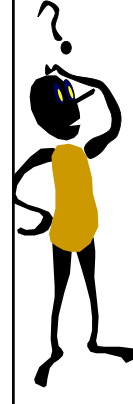
The complete graph on  $n$  vertices, denoted  $K_n$ , is a simple graph in which there is an edge between every pair of distinct vertices.

What is the height of the DFS tree for the complete graph  $K_n$ ?

$n-1$

What is the height of the BFS tree for the complete graph  $K_n$ ?

$1$



### Exercise

What is the visual difference between BFS and DFS trees?

DFS trees is tall and skinny.


BFS trees is short and bushy.

### Problem 2

Mathematicians often keep track of a statistic called their Erdős Number, after the great 20th century mathematician. Paul Erdős himself has a number of zero. Anyone who wrote a mathematical paper with him has a number of one, anyone who wrote a paper with someone who wrote a paper with him has a number of two, and so forth and so on. Supposing that we have a database of all mathematical papers ever written along with their authors:

- Explain how to represent this data as a graph.
- Explain how we would compute the Erdős number for a particular researcher.
- Explain how we would determine all researchers with Erdős number at most two.

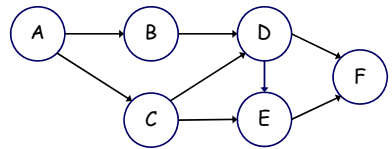
### Topological Sort



### Definition

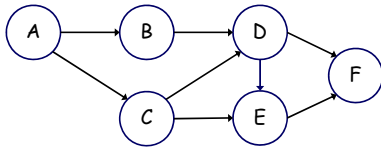
Suppose each vertex represents a task that must be completed, and an edge  $(u, v)$  indicates that task  $u$  depends on task  $v$ . That is  $v$  must be completed before  $u$ . The topological ordering of the vertices is a valid order in which you can complete the tasks.

### Simple Algorithm



- Select a vertex that has in-degree zero.
- Add the vertex to the output.
- Delete the vertex and all its outgoing edges.
- Repeat.

### Better Algorithm

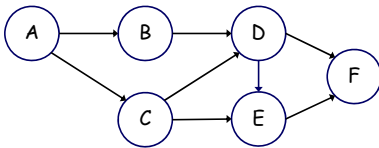


- Select a vertex that has in-degree zero.
- Run DFS and return vertex that has no undiscovered leaving edges

### Problem 3

In class we discussed finding the shortest path between two vertices in a graph. Suppose instead we are interested in finding the longest simple path in a directed acyclic graph. In particular, we are interested in finding a path (if one exists) that visits all vertices (also known as a *Hamiltonian Path*). Given a DAG, give a linear time algorithm to determine if there is a simple path that visits all vertices.

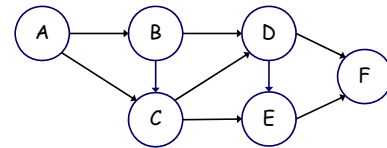
### Solution 3



Topological Orderings: A B C D E F or A C B D E F

There is no simple path that visits every node

### Solution 3



Topological Ordering: A B C D E F

The ordering is a valid path in the DAG