CS340 - Analysis of Algorithms

Basics of Algorithm Analysis part 2

Logistics:

HW1 was due today

HW2 is released

Upcoming deadlines:

- Lab1 due Thursday
- HW2 due next Monday (9/22)

Quiz today

Agenda

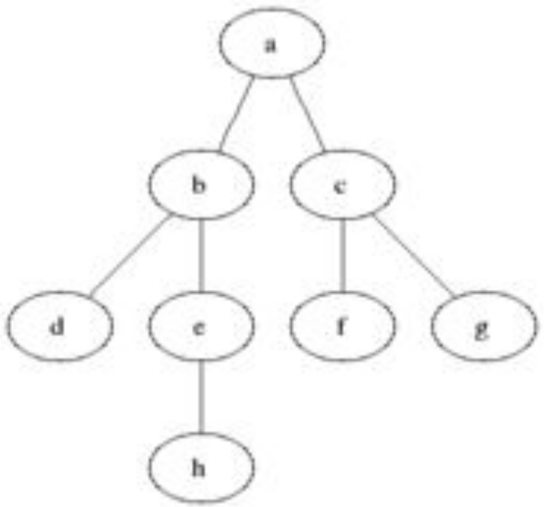
1. **Quiz**

2. Graphs pt 2

3. Basics of Analysis of Algorithms pt 2
    a. Complexity with limits
    b. Asymptotic lower bound
    c. Asymptotic equal bound

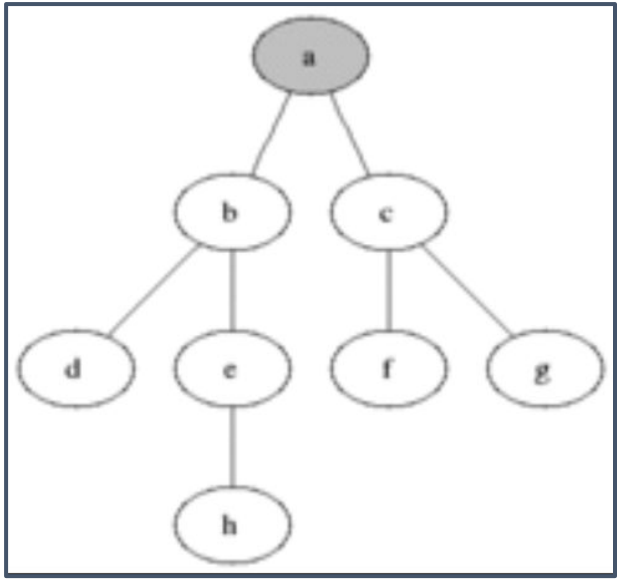
Breadth First Search (BFS)

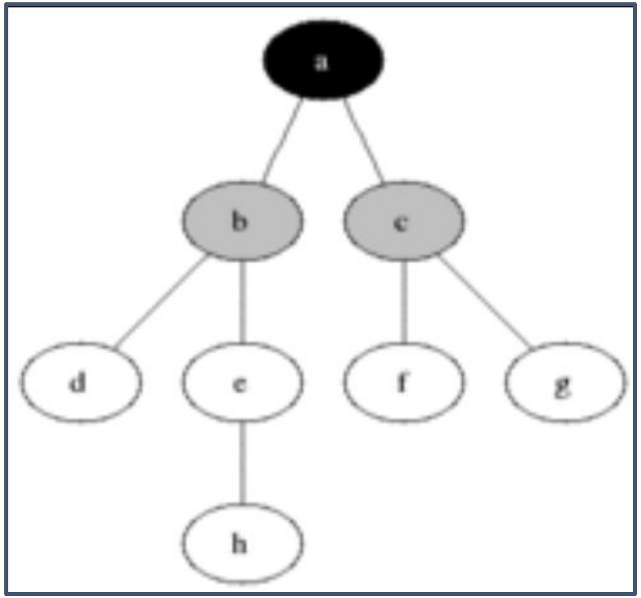
- Starts at the root and explores all nodes at the present “depth” before moving to nodes on the next level
- Extra memory is usually required to keep track of the nodes that have not yet been explored

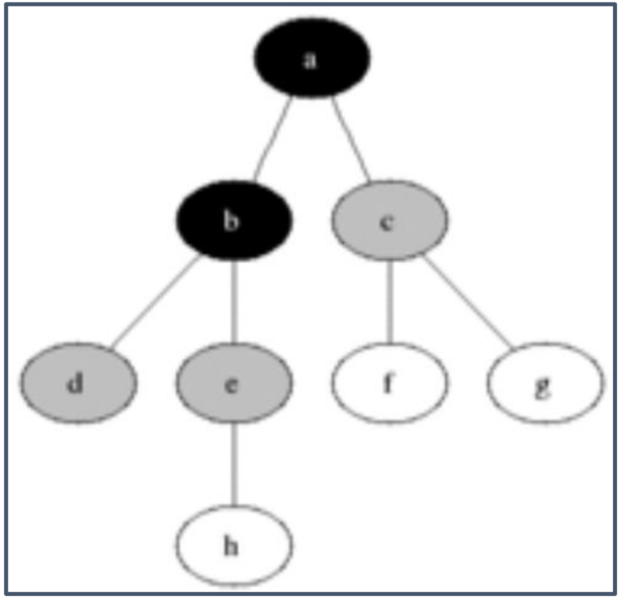


Breadth-First Traversal

pseudo-code?







Runtime Analysis of BFS

$|V| = n$, $|E| = m$

1. Initialization:
   a. $O(n)$

2. Traversal
   a. While-loop: we visit each vertex once
   b. Nested for-loop: we visit each child of that vertex
      i. Number of iterations depends on the *degree*

$$T(n) = n + \Sigma_{u \in V} (\deg(u) + 1)$$

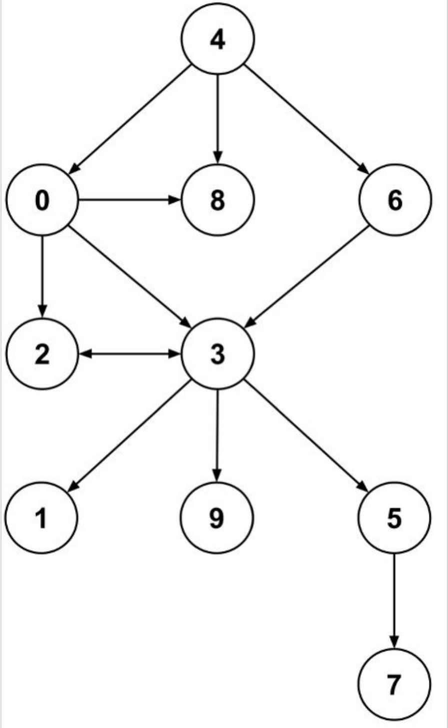
$$= n + (\Sigma_{u \in V} \deg(u)) + n = 2n + \Sigma_{u \in V} \deg(u)$$

$$= 2n + 2m$$

$$= O(n + m)$$

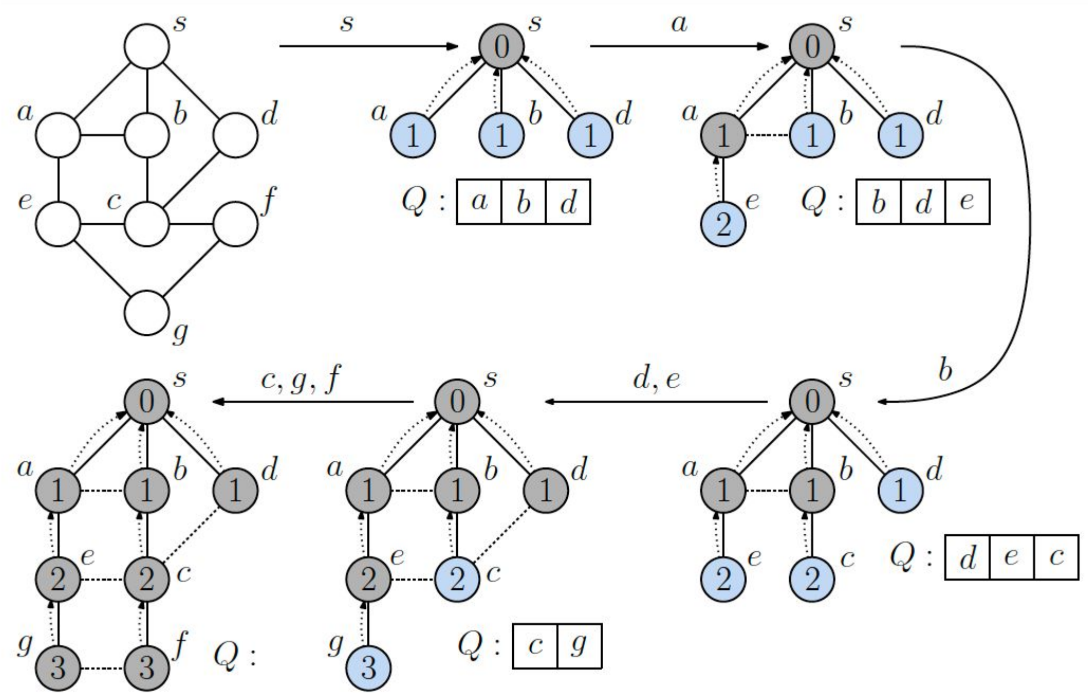
BFS trace

```
BFS(G, s) {
  set mark to all false
  mark[s] = true //print s
  Q = {s}
  while (Q is not empty) {
    u = dequeue of Q
    for each (v in Adj[u]) {
      if (!mark[v]){
        mark[v]=true //print v
        append v to Q
    }
  }
}
```



BFS on an undirected graph

```
BFS(G, s) {
  set mark to all false
  mark[s] = true //print s
  Q = {s}
  while (Q is not empty) {
    u = dequeue of Q
    for each (v in Adj[u]) {
      if (!mark[v]){
        mark[v]=true //print v
        append v to Q
      }
    }
  }
}
```



BFS with more record keeping

Let's add code to track:

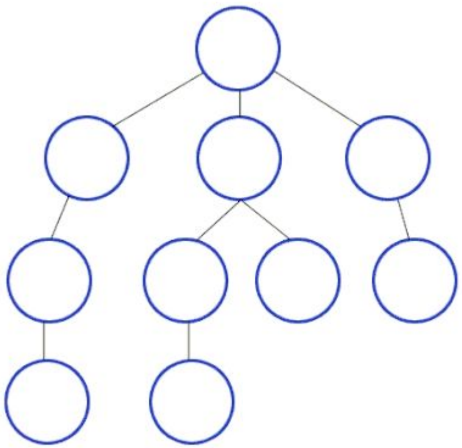
1. The distance from the start node
2. The predecessor (parent) of each node

BFS with more record keeping

```
BFS(G, s) {
  for each (u in V) {
    mark[u] = false, d[u] = infinity, pred[u] = null
  }
  mark[s] = true, d[s] = 0, Q = {s}
  while (Q is not empty) {
    u = dequeue of Q
    for each (v in Adj[u]) {
      if (!mark[v]){
        mark[v] = true
        d[v] = d[u]+1
        pred[v] = u
        append v to Q
    }
  }
}
```

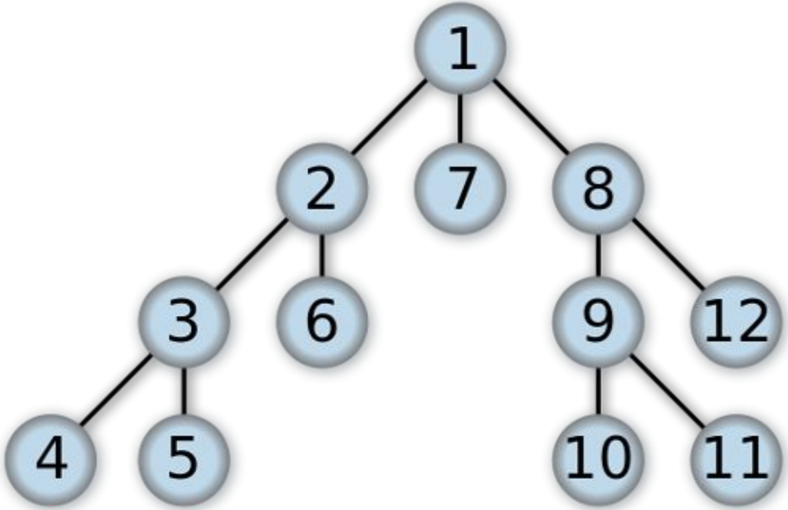
Depth First Search (DFS)

- start at root node and explore as far as possible along each branch
- Recursive algorithm



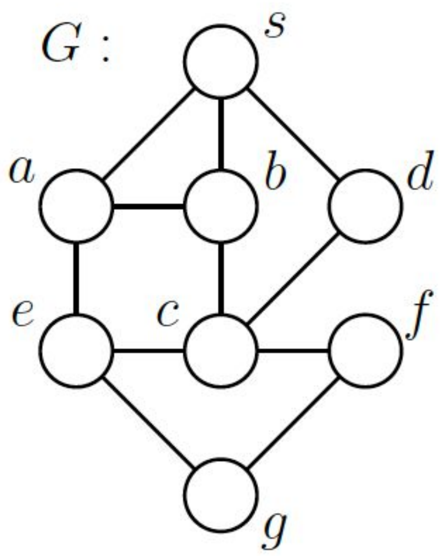
DFS Trace 1

```
DFSG(G) {
  set mark to all false
  for each (v in V) {
    if (!mark(v))
      DFS(v)
  }
}
DFS(u) {
  mark[u] = true  // print u
  for each (v in Adj[u]) {
    if (!mark[v]){
      DFS(v)
    }
  }
}
```

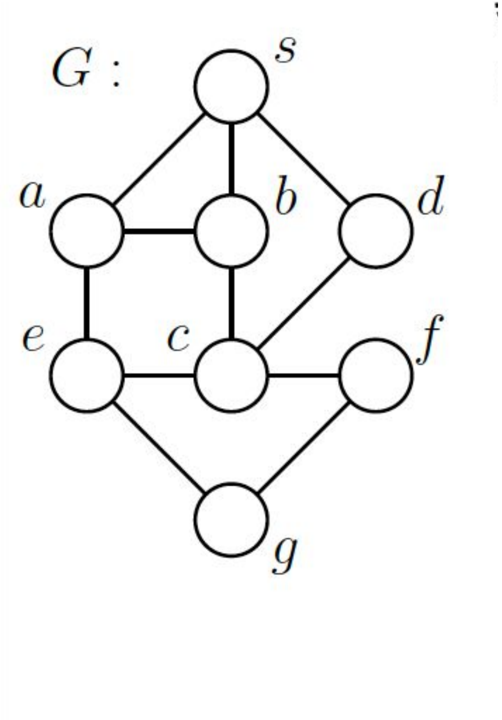


DFS Trace 2

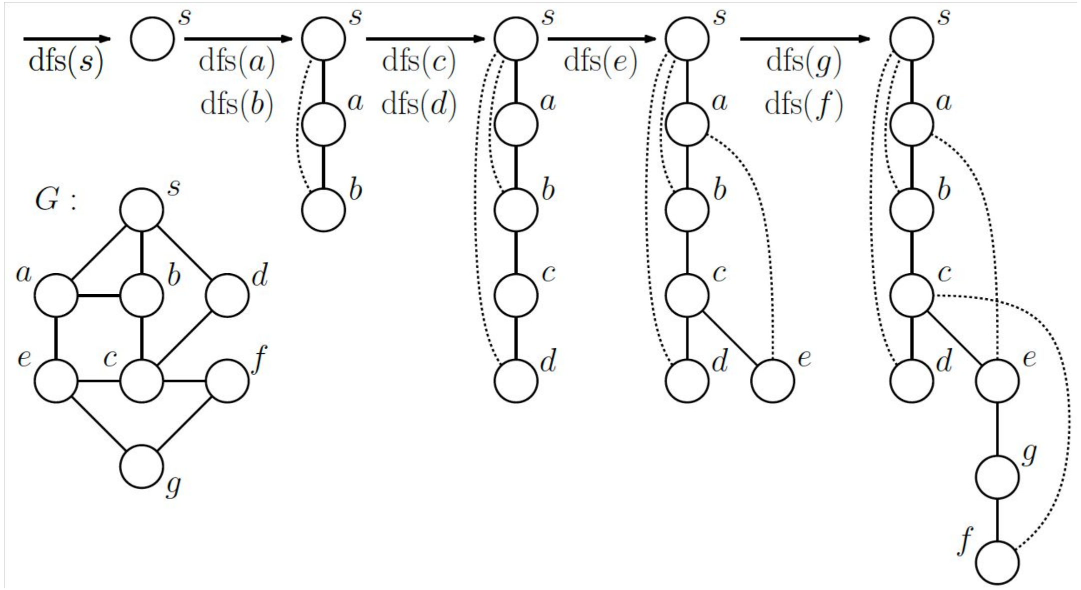
```
DFSG(G) {
  set mark to all false
  for each (v in V) {
    if (!mark(v))
      DFS(v)
  }
}
DFS(u) {
  mark[u] = true  // print u
  for each (v in Adj[u]) {
    if (!mark[v]){
      DFS(v)
    }
  }
}
```



DFS Trace 2



G :
s
a
b
d
e
c
f
g



dfs(s)
dfs(a)
dfs(b)
dfs(c)
dfs(d)
dfs(e)
dfs(g)
dfs(f)
G :
s
a
b
c
d
e
f
g

DFS Runtime Analysis

$|V| = n$, $|E| = m$

1. Wrapper:
   a. $O(n)$

2. Traversal:
   a. DFS is called once per vertex
   b. for-loop: we visit each child of that vertex
      i. Number of iterations depends on the *degree*

$$T(n) = n + \Sigma_{u \in V} (\deg(u) + 1)$$
$$= n + (\Sigma_{u \in V} \deg(u)) + n = 2n + \Sigma_{u \in V} \deg(u)$$
$$= 2n + 2m$$
$$= O(n + m)$$

DFS with more record keeping

Let's add code to track:

1. The start and finish time of node processing
    a. Time is the iteration # (kind of...)
        i. Time increases before and after node is processed
    b. Start is the node is printed / marked
    c. End is when all of its children have been processed
2. The predecessor (parent) of each node

DFS with more record keeping

- start times array: `s`
- finish times array: `f`
- predecessors array: `pred`

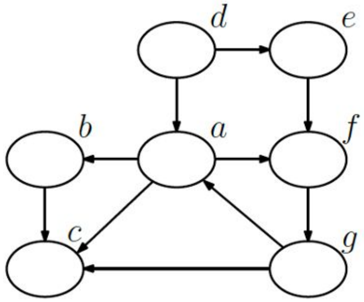
```
DFSG(G) {
  time = 0
  for each (u in V) {
    mark[u] = unseen
  }
  for each (u in V) {
    if (mark[u] == unseen)
      DFS(u)
  }
}
```

```
DFS(u) {
  mark[u] = seen
  s[u] = time++
  for each (v in Adj[u]) {
    if (mark[v] == unseen){
      pred[v] = u
      DFS(v)
    }
  }
  mark[u] = finished
  f[u] = time++
}
```

DFS with more record keeping

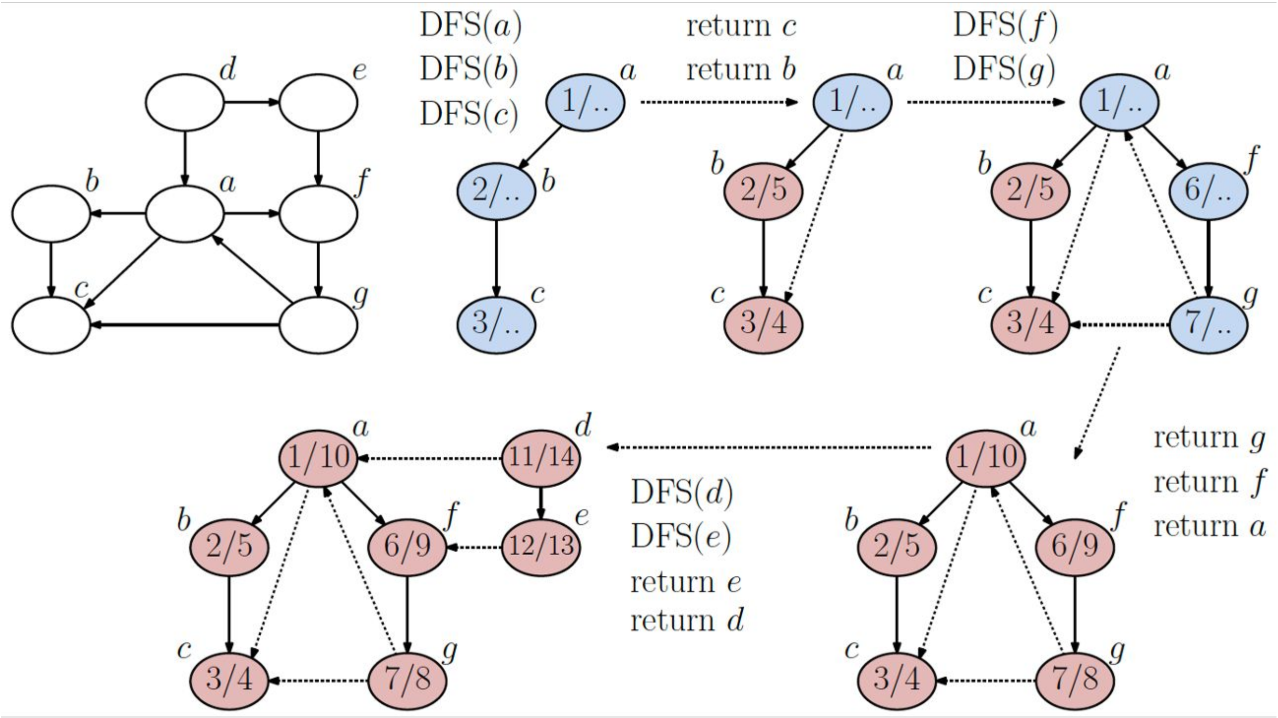
- start times array: `s`
- finish times array: `f`
- predecessors array: `pred`

```
DFSG(G) {
  time = 1
  for each (u in V) {
    mark[u] = unseen
  }
  for each (u in V) {
    if (mark[u] == unseen)
      DFS(u)
  }
}
```

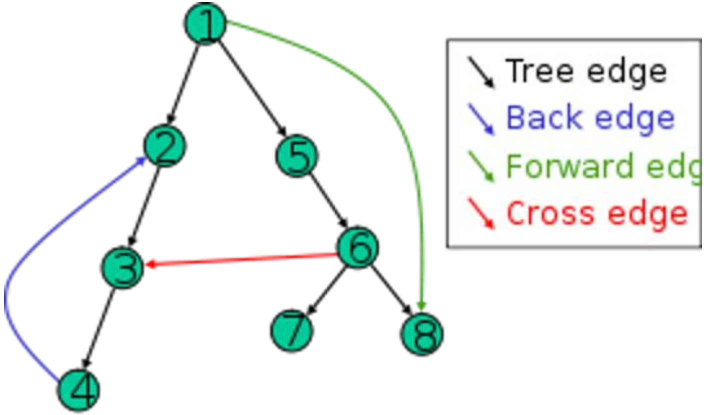



```
DFS(u) {
  mark[u] = seen
  s[u] = time++
  for each (v in Adj[u]) {
    if (mark[v] == unseen){
      pred[v] = u
      DFS(v)
    }
  }
  mark[u] = finished
  f[u] = time++
}
```

DFS with more record keeping



DFS Edge Classification



- If v is visited for the first time as we traverse (u, v), then (u, v) is a tree edge
- else, v has already been visited
  - if v is an ancestor of u, (u, v) is a back edge
  - if v is a descendent of u, then (u, v) is a forward edge
  - if v is neither, then (u, v) is a cross edge

Agenda

1. Quiz

2. Graphs pt 2

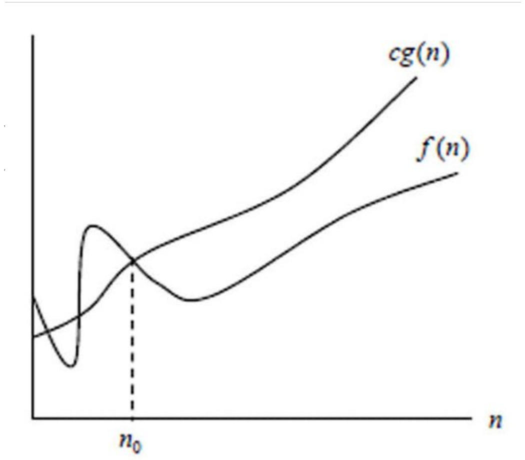
3. **Basics of Analysis of Algorithms pt 2**
    a. Complexity with limits
    b. Asymptotic lower bound
    c. Asymptotic equal bound

Big O

Big O notation expresses **Asymptotic Upper Bounds**

“If f(n) doesn’t grow faster than g(n), up to a constant factor, for large enough n.”

If $\exists\ n_0 \geq 0, c > 0 : f(n) \leq c \cdot g(n)\ \forall\ n \geq n_0$ then O(f(n)) = g(n



Big Ω

Big Ω notation expresses **Asymptotic *Lower* Bounds**

"If f(n) is at least a constant multiple of g(n) for large enough n,

then "Ω(f(n)) = g(n)"

If $\exists\ n_0 \geq 0, c > 0 : f(n) \geq c \cdot g(n)\ \forall\ n \geq n_0$ then $\Omega(f(n)) = g(n)$

Big Θ

Big Θ notation expresses **Asymptotically Tight Bounds**

“If f(n) is both f(n) and g(n), then Θ(f(n)) = g(n)”

From a limits POV: If the ratio of functions f(n) and g(n) converges to a positive constant as n goes to infinity, then Θ(f(n)) = g(n)

From the limits POV

Notation	Relational Form	Limit Definition
$f(n) = O(g(n))$	$f(n) \preccurlyeq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c, 0$
$f(n) = \Theta(g(n))$	$f(n) \approx g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$
$f(n) = \Omega(g(n))$	$f(n) \succcurlyeq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c, \infty$

How are these f(n) and g(n) asymptotically related?

$$f(n) \prec \approx \succ g(n)?$$

- $f(n) = 3^{\frac{n}{2}}, g(n) = 2^{\frac{n}{3}}$
- $f(n) = \log(n^2), g(n) = (logn)^2$
- $f(n) = n^{log4}, g(n) = 2^{2logn}$
- $f(n) = \max(n^2, n^3), g(n) = n^2 + n^3$
- $f(n) = \min(2^n, 2^{1000}n), g(n) = n^{1000}$

Summary

1. Summary
   a. Graph Traversals - DFS and BFS
      i. DFS recursive
      ii. BFS requires additional memory (queue)
      iii. Both $O(m+n)$
   b. Asymptotic bounds can be found by computing limits
      i. $\lim f(n) / g(n) = 0$ then $f(n) = O(g(n))$
      ii. $\lim f(n) / g(n) = \infty$ then $f(n) = \Omega(g(n))$
      iii. $\lim f(n) / g(n) = c$ then $f(n) = \Theta(g(n))$

2. Upcoming deadlines:
   a. Lab 1 due thursday
   b. HW2 due Sep 22nd (next Monday)
   c. Continue reading textbook

3. Next class: greedy algorithms