Worst Case Runtime Function
• Problem P: Given instance I compute a solution S
• A is an algorithm to solve P
• T(I) is the number of steps executed by A on instance I
• T(n) is the maximum of T(I) for all instances of size n

Ignore constant factors
• Constant factors are arbitrary
  – Depend on the implementation
  – Depend on the details of the model
• Determining the constant factors is tedious and provides little insight
• Express run time as \( T(n) = O(f(n)) \)

Formalizing growth rates
• \( T(n) = O(f(n)) \) \( [T : \mathbb{Z}^+ \rightarrow \mathbb{R}^+] \)
  – If \( n \) is sufficiently large, \( T(n) \) is bounded by a constant multiple of \( f(n) \)
  – Exist \( c, n_0 \), such that for \( n > n_0 \), \( T(n) < c f(n) \)
• \( T(n) = O(f(n)) \) will be written as:
  \( T(n) = O(f(n)) \)
  – Be careful with this notation

Prove \( 3n^2 + 5n + 20 \) is \( O(n^2) \)
Let \( c = \)
Let \( n_0 = \)

\( T(n) = O(f(n)) \) if there exist \( c, n_0 \), such that for \( n > n_0 \),
\( T(n) < c f(n) \)
Order the following functions in increasing order by their growth rate

a) \( n \log^{4} n \)
b) \( 2n^2 + 10n \)
c) \( 2^{n^{100}} \)
d) \( 1000n + \log^9 n \)
e) \( n^{100} \)
f) \( 3^n \)
g) \( 1000 \log^{100} n \)
h) \( n^{1/2} \)

Lower bounds

- \( T(n) = \Omega(f(n)) \)
  - \( T(n) \) is at least a constant multiple of \( f(n) \)
  - There exists an \( n_0 \) and \( \epsilon > 0 \) such that \( T(n) > \epsilon f(n) \) for all \( n > n_0 \)
- Warning: definitions of \( \Omega \) vary

- \( T(n) = \Theta(f(n)) \) if \( T(n) \) is \( O(f(n)) \) and \( T(n) \) is \( \Omega(f(n)) \)

Useful Theorems

- If \( \lim (f(n) / g(n)) = c \) for \( c > 0 \) then \( f(n) = \Theta(g(n)) \)

- If \( f(n) \) is \( O(g(n)) \) and \( g(n) \) is \( O(h(n)) \) then \( f(n) \) is \( O(h(n)) \)

- If \( f(n) \) is \( O(h(n)) \) and \( g(n) \) is \( O(h(n)) \) then \( f(n) + g(n) \) is \( O(h(n)) \)

Ordering growth rates

- For \( b > 1 \) and \( x > 0 \)
  - \( \log^b n \) is \( O(n^x) \)

- For \( r > 1 \) and \( d > 0 \)
  - \( n^d \) is \( O(r^n) \)

Graph Theory

- \( G = (V, E) \)
  - \( V \) – vertices
  - \( E \) – edges
- Undirected graphs
  - Edges sets of two vertices \( \{u, v\} \)
- Directed graphs
  - Edges ordered pairs \( (u, v) \)
- Many other flavors
  - Edge / vertices weights
  - Parallel edges
  - Self loops

Definitions

- Path: \( v_1, v_2, \ldots, v_n \), with \( (v_i, v_{i+1}) \) in \( E \)
  - Simple Path
  - Cycle
  - Simple Cycle
- Neighborhood
  - \( N(v) \)
- Distance
- Connectivity
  - Undirected
  - Directed (strong connectivity)
- Trees
  - Rooted
  - Unrooted
Graph Representation

\[ V = \{a, b, c, d\} \]
\[ E = \{(a, b), (a, c), (a, d), (b, d)\} \]

Adjacency List

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Incidence Matrix

- **Graph search**
  - Find a path from s to t
  
  \[ S = \{s\} \]
  
  while S is not empty
  
  \[ u = \text{Select}(S) \]
  
  visit \( u \)
  
  foreach \( v \) in \( N(u) \)
  
  if \( v \) is unvisited
  
  Add(S, v)
  
  \[ \text{Pred}[v] = u \]
  
  if \( (v = t) \) then path found

- **Breadth first search**
  - Explore vertices in layers
  - s in layer 1
  - Neighbors of s in layer 2
  - Neighbors of layer 2 in layer 3...

- **Bipartite Graphs**
  - A graph \( V \) is bipartite if \( V \) can be partitioned into \( V_1, V_2 \) such that all edges go between \( V_1 \) and \( V_2 \)
  - A graph is bipartite if it can be two colored

- **Key observation**
  - All edges go between vertices on the same layer or adjacent layers

- **Can this graph be two colored?**
Algorithm

- Run BFS
- Color odd layers red, even layers blue
- If no edges between the same layer, the graph is bipartite
- If edge between two vertices of the same layer, then there is an odd cycle, and the graph is not bipartite

Theorem: A graph is bipartite if and only if it has no odd cycles

Lemma 1

- If a graph contains an odd cycle, it is not bipartite

![Graph with an odd cycle](image1)

Lemma 2

- If a BFS tree has an *intra-level edge*, then the graph has an odd length cycle

*Intra-level edge: both end points are in the same level*

Lemma 3

- If a graph has no odd length cycles, then it is bipartite

Graph Search

- Data structure for next vertex to visit determines search order

![Graph search](image2)
Graph search

Breadth First Search
S = {s}
while S is not empty
u = Dequeue(S)
if u is unvisited
visit u
foreach v in N(u)
Enqueue(S, v)

Depth First Search
S = {s}
while S is not empty
u = Pop(S)
if u is unvisited
visit u
foreach v in N(u)
Push(S, v)

Breadth First Search

• All edges go between vertices on the same layer or adjacent layers

Depth First Search

• Each edge goes between vertices on the same branch
• No cross edges