UMBC

AN HONORS UNIVERSITY IN MARYLAND

IS 709/809: Computational Methods in IS Research Fall 2017

Exam Review

Nirmalya Roy

Department of Information Systems

University of Maryland Baltimore County

Exam

- When: Tuesday (11/28) 7:10pm – 9:40pm
- Where: In Class

- Open book, Open notes
- Comprehensive (Algorithms, Queueing Theory)

- Materials for preparation:
  - Lecture slides
  - Quiz and Homework assignments
  - Textbooks

Course Overview

- Math review
- Algorithm development and analysis
  - Running time
- Sorting
  - Insertion sort, selection sort, merge sort, quick sort
- Graph Algorithms
  - Topological Sort
  - Shortest-Path Algorithms
    - Dijkstra's Algorithm
    - Network Flow Problems

Course Overview

- Graph Algorithms
  - Minimum Spanning Tree
    - Prim’s Algorithm
    - Kruskal’s Algorithm
- Queueing Theory
  - Little’s Theorem, Erlang Traffic
  - M/M/1
- Research papers and Development project

Algorithmic Complexity
and
Graph Theory

Math Review

- Floors, ceilings, exponents and logarithms
  - Definitions and manipulations
- Series: Arithmetic and Geometric series
  - Definitions and manipulations
- Proof Techniques: Read the definition, components, and how to use the following
  - Proof by induction
  - Proof by counterexample
  - Proof by contradiction

Math Review

- Floors, ceilings, exponents and logarithms
  - Definitions and manipulations

- Series: Arithmetic and Geometric series
  - Definitions and manipulations

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6}$$

Math Review

- Proof Techniques: Read the definition, components, and how to use the following
  - Proof by induction
    - Example: Prove $\sum_{j=1}^{n}(2j-1) = n^2$ (is a perfect square)
  - Proof by counterexample
  - Proof by contradiction

- Recursion
  - Read the definition and rules
  - Analyze running time of recursive algorithm

Algorithm Analysis

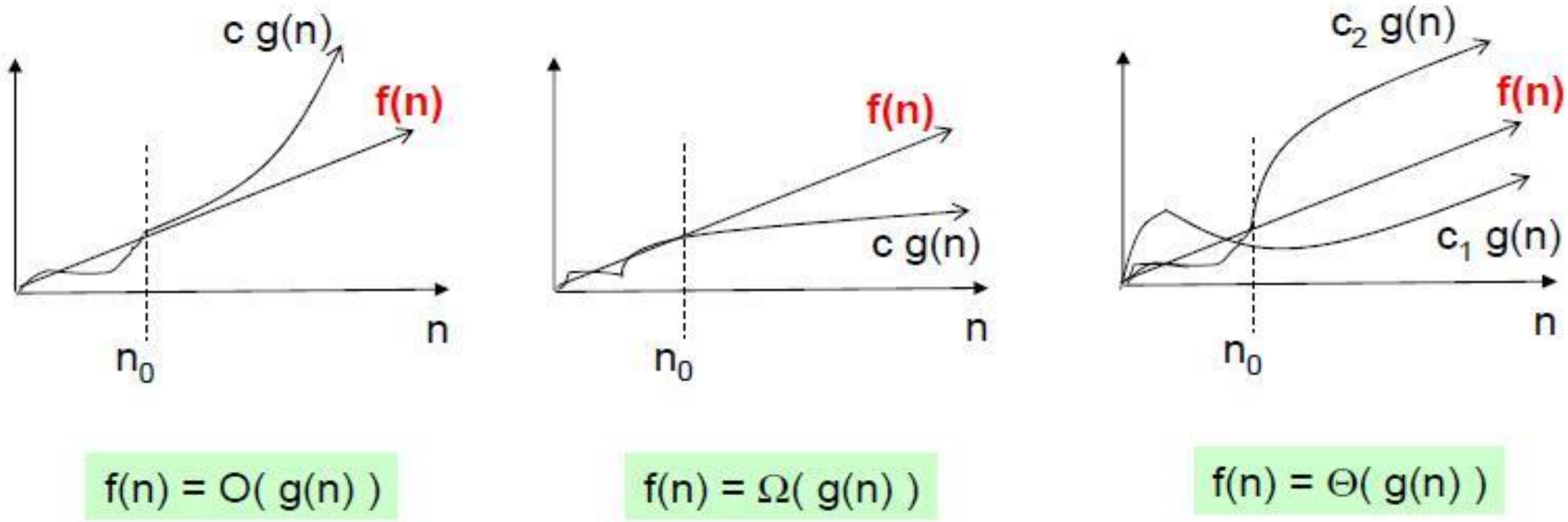
- Asymptotic analysis of an algorithm
- Best-case, worst-case and average-case analysis
- Rate of growth: Definitions and Notation (O, Ω, Θ, o)
  - Proofs for specific examples

Asymptotic Notations

- O() – upper bound
  - Definition: $T(N) = O(g(N))$ if there are positive constants c and n_0 such that $T(N) \leq cg(N)$ when $N \geq n_0$
- $\Omega()$ – lower bound
  - Definition: $T(N) = \Omega(g(N))$ if there are positive constants c and n_0 such that $T(N) \geq cg(N)$ when $N \geq n_0$
- $\Theta()$ – tight bound
  - Definition: $T(N) = \Theta(g(N))$ if and only if $T(N) = O(g(N))$ and $T(N) = \Omega(g(N))$
- o() – strict upper bound
  - Definition: $T(N) = o(g(N))$ if for all constants *c* there exists an n_0 such that $T(N) < cg(N)$ when $N > n_0$

Asymptotic Analysis

In this diagram; T(n) = f(n)



- O-notation gives an upper bound for a function to within a constant factor
- Ω-notation gives an lower bound for a function to within a constant factor
- Θ-notation bounds a function to within a constant factor
  - The value of f(n) always lies between c_1 g(n) and c_2 g(n) inclusive

Maximum Subsequence Sum Problem

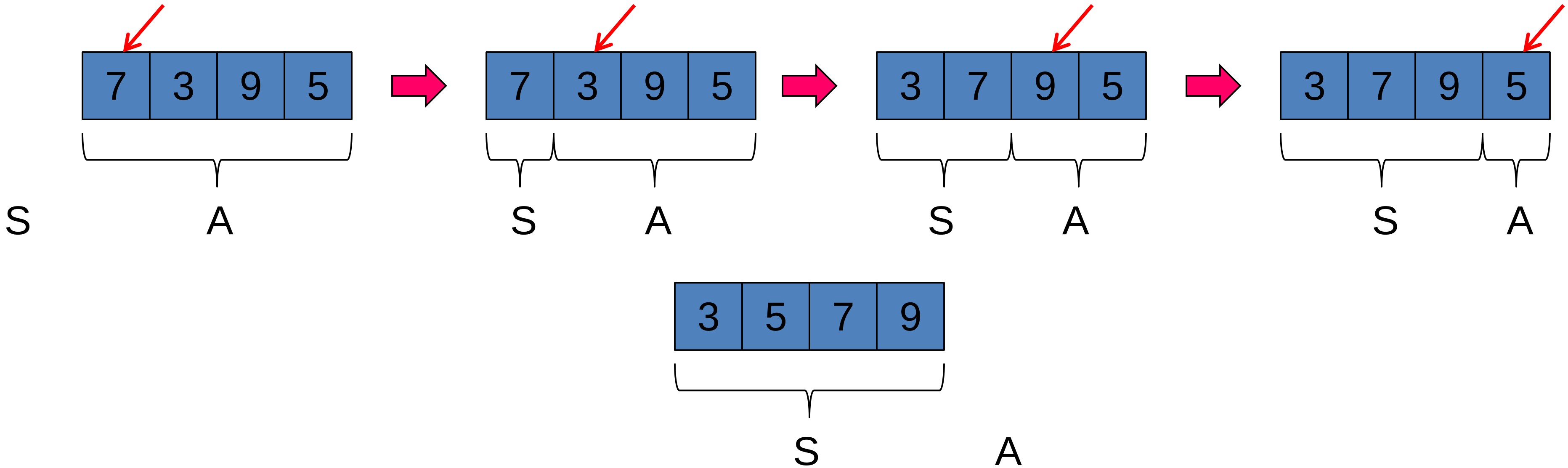
- Maximum subsequence sum problem
  - Given (possibly negative) integers A_1, A_2, …, A_N, find the maximum value (≥ 0) of: $$\sum_{k=i}^{j} A_k$$
  - E.g. <1, -4, 4, 2, -3, 5, 8, -2>   The maximum sum is 16
  - Solution # 1: $T(N) = O(N^3)$
  - Solution # 2: $T(N) = O(N^2)$
  - Solution # 3: Recursive, "divide and conquer" $T(N) = O(N \log N)$
  - Solution # 4: Online Algorithm $T(N) = O(N)$

Sorting

- Insertion sort; Worst case: $O(N^2)$; Best Case $O(N)$
- Selection sort ; Worst-case: $O(N^2)$
- Shell sort; Worst-case: $O(N^{3/2})$
- Merge sort; Worst-case: $O(N \log_2 N)$
- Quick sort; Worst-case: $O(N^2)$

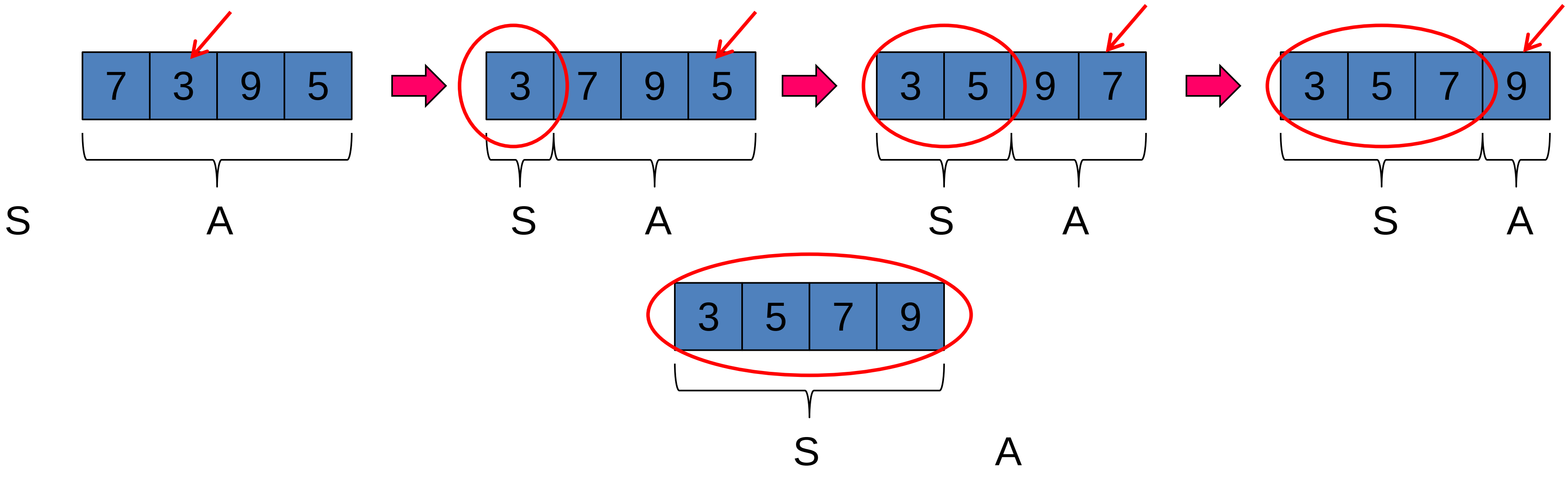
Insertion Sort

- Algorithm:
  - Start with empty list S and unsorted list A of N items
  - For each item x in A
    - Insert x into S, in sorted order
- Example:

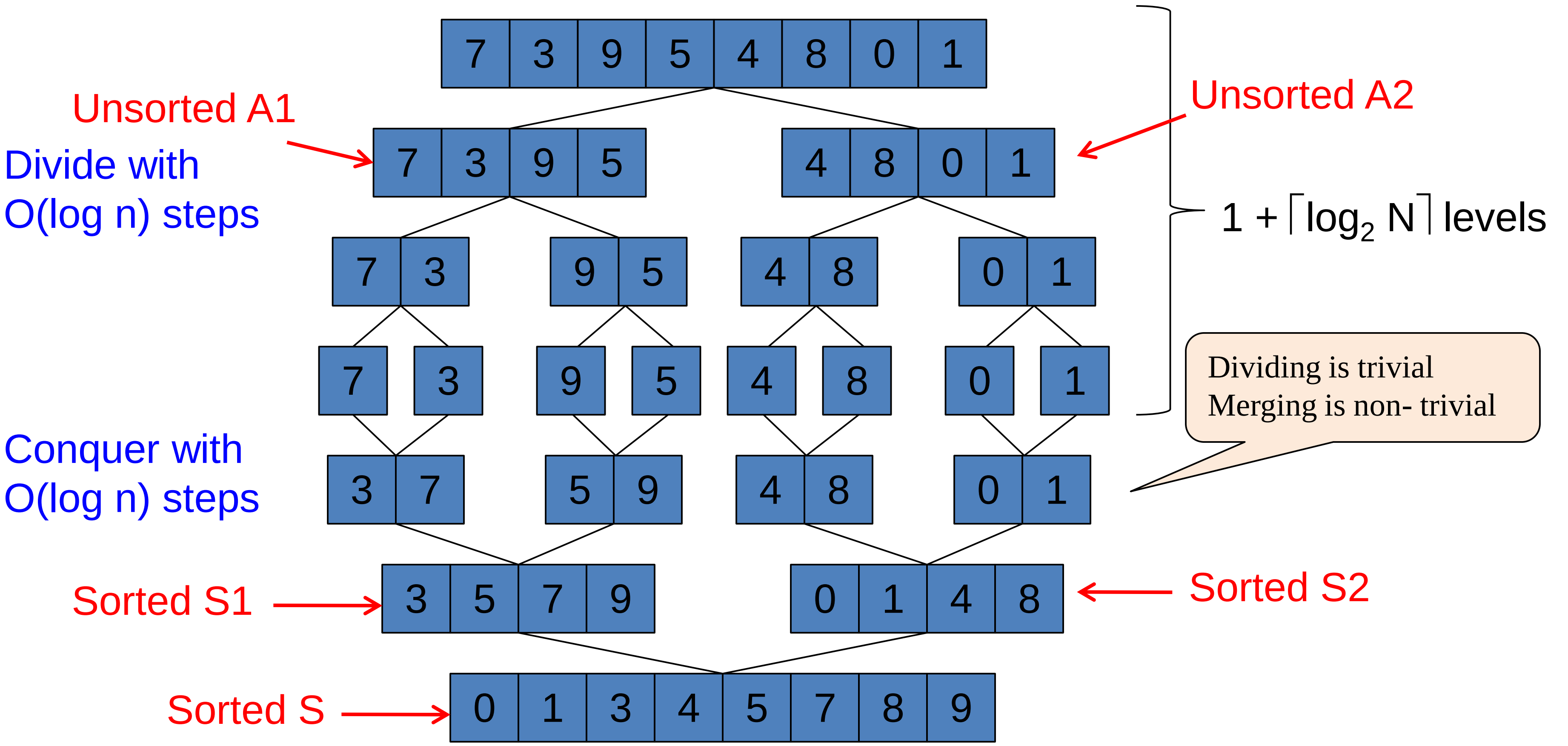


Selection Sort

- Algorithm:
  - Start with empty list S and unsorted list A of N items
  - for (i = 0; i < N; i++)
    - x ← item in A with smallest key
    - Remove x from A
    - Append x to end of S



Merge Sort



7 3 9 5 4 8 0 1
Unsorted A1
Unsorted A2
Divide with
O(log n) steps
7 3 9 5
4 8 0 1
$1 + \lceil \log_2 N \rceil$ levels
7 3
9 5
4 8
0 1
7
3
9
5
4
8
0
1
Dividing is trivial
Merging is non- trivial
Conquer with
O(log n) steps
3 7
5 9
4 8
0 1
Sorted S1
3 5 7 9
0 1 4 8
Sorted S2
Sorted S
0 1 3 4 5 7 8 9

Quick Sort Algorithm

- quicksort (array: S)
  1. If size of S is 0 or 1, return

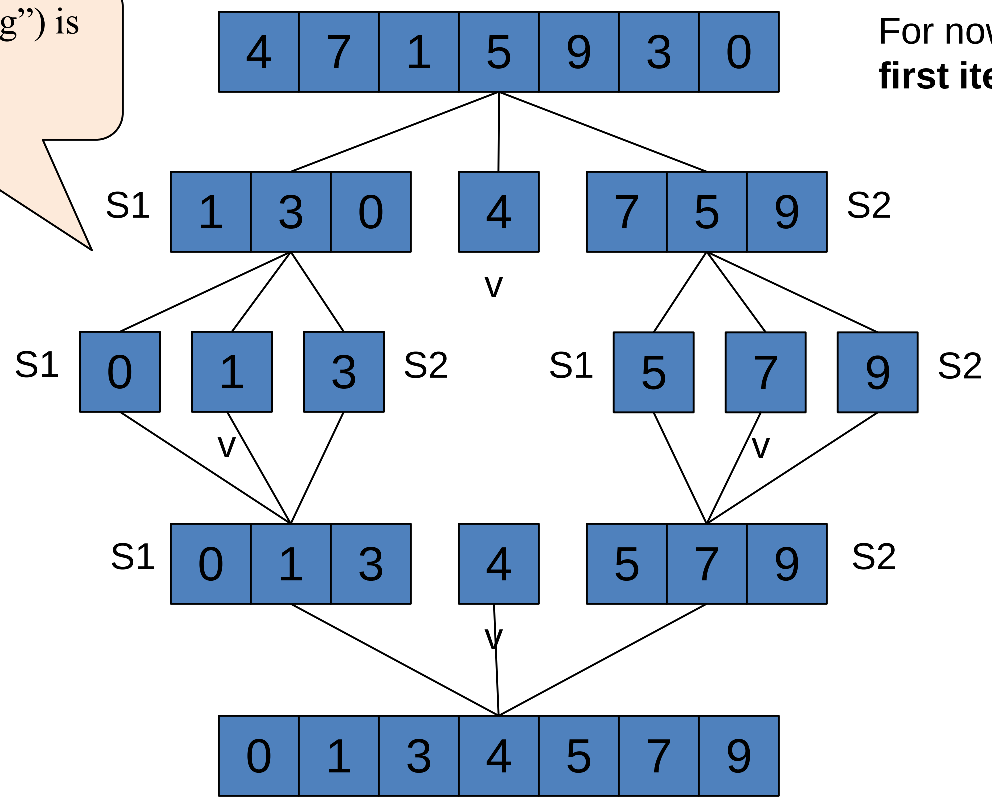
  2. Pivot = Pick an element v in S

  3. Partition S – {v} into two disjoint groups
     $S1 = \{x \in (S - \{v\}) \mid x < v\}$
     $S2 = \{x \in (S - \{v\}) \mid x > v\}$

  4. Return {quicksort(S1), followed by v, followed by quicksort(S2)}

Quick Sort

Dividing (“Partitioning”) is non-trivial
Merging is trivial



For now, let the pivot v be the **first item** in the list.

$O(N \log_2 N)$

Comparison of Sorting Algorithms

Sort	Worst Case	Average Case	Best Case	Comments
InsertionSort	Θ(N^2)	Θ(N^2)	Θ(N)	Fast for small N
ShellSort	Θ($N^{3/2}$)	Θ($N^{7/6}$) ?	Θ(N log N)	Increment sequence?
HeapSort	Θ(N log N)	Θ(N log N)	Θ(N log N)	Large constants
MergeSort	Θ(N log N)	Θ(N log N)	Θ(N log N)	Requires memory
QuickSort	Θ(N^2)	Θ(N log N)	Θ(N log N)	Small constants

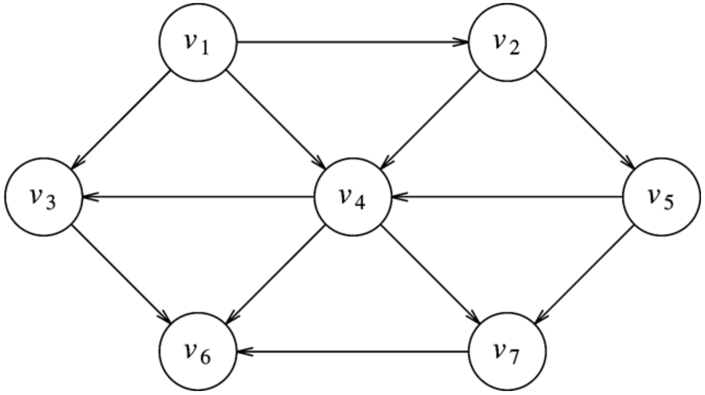
Graph Algorithms

- Graph Definition
  - Directed graph, undirected graph, complete graph
- Graph Algorithms
  - Topological sort
  - Shortest paths
  - Network flow
  - Minimum spanning tree

Topological Sort

- Order the vertices in a directed acyclic graph (DAG), such that if (u, v) $\in$ E, then u appears before v in the ordering
- Solution #1 is $O(|V|^2)$
- Solution #2 is $O(|V| + |E|)$ (queue implementation)

	Indegree Before Dequeue #						
Vertex	1	2	3	4	5	6	7
v_1	0	0	0	0	0	0	0
v_2	1	0	0	0	0	0	0
v_3	2	1	1	1	0	0	0
v_4	3	2	1	0	0	0	0
v_5	1	1	0	0	0	0	0
v_6	3	3	3	3	2	1	0
v_7	2	2	2	1	0	0	0
Enqueue	v_1	v_2	v_5	v_4	v_3, v_7		v_6
Dequeue	v_1	v_2	v_5	v_4	v_3	v_7	v_6

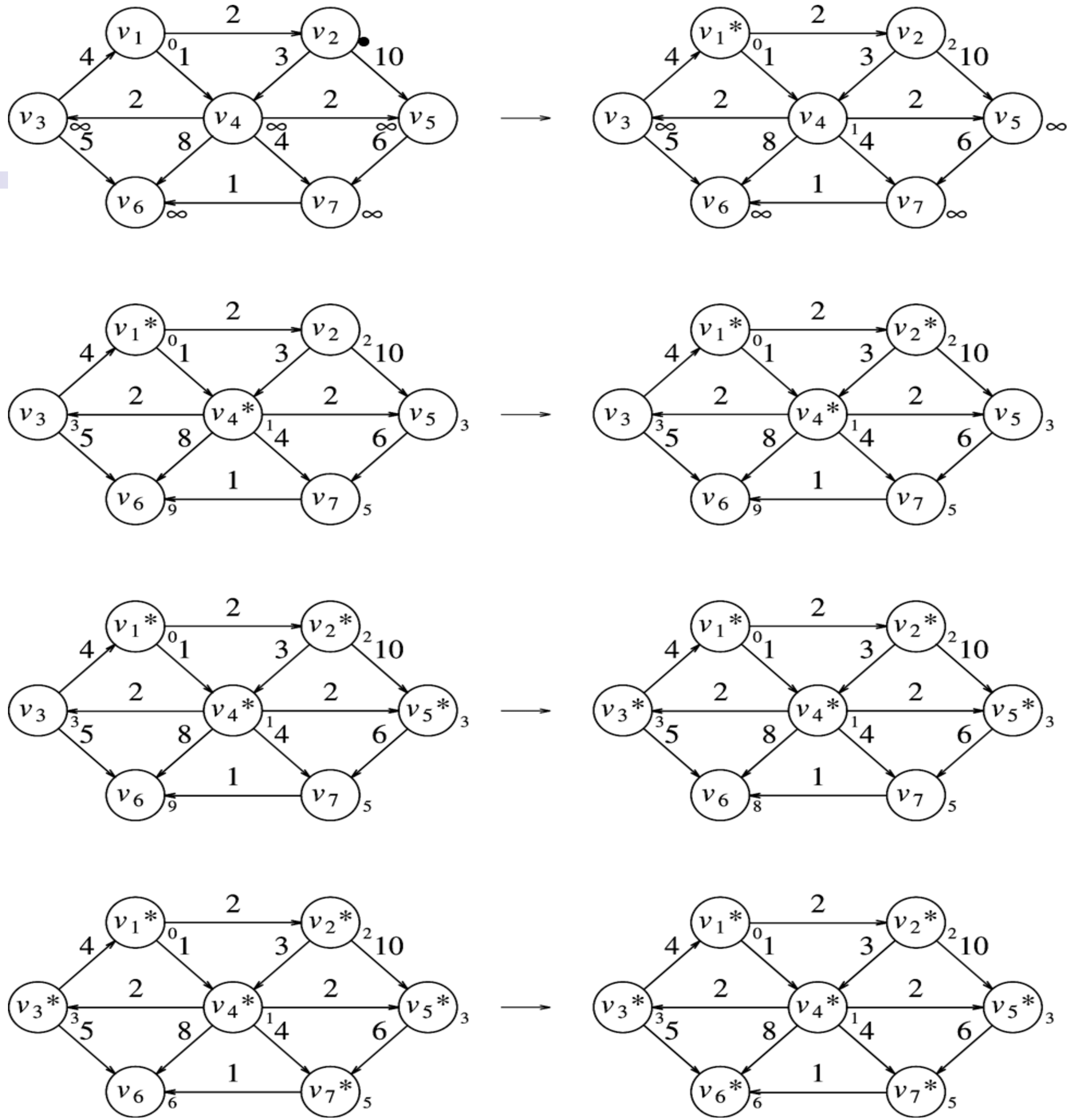


Possible topological orderings:
$v_1, v_2, v_5, v_4, v_3, v_7, v_6$ and
$v_1, v_2, v_5, v_4, v_7, v_3, v_6$.

Shortest Path Problems

- Input is a weighted graph where each edge (v_i, v_j) has cost $c_{i,j}$ to traverse the edge
- Cost of a path $v_1 v_2 \ldots v_N$ is
  - Weighted path cost
- Unweighted shortest path (number of edges on path): $O(|E| + |V|)$
- Weighted shortest path (weighted path cost)
  - Dijkstra's algorithm : $O(|E| \log |V|)$

Dijkstra's Algorithm



Dijkstra's Adjacency List

v	*known*	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	∞	0
v_4	F	1	v_1
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	3	v_4
v_4	T	1	v_1
v_5	F	3	v_4
v_6	F	9	v_4
v_7	F	5	v_4

v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	F	3	v_4
v_4	T	1	v_1
v_5	F	3	v_4
v_6	F	9	v_4
v_7	F	5	v_4

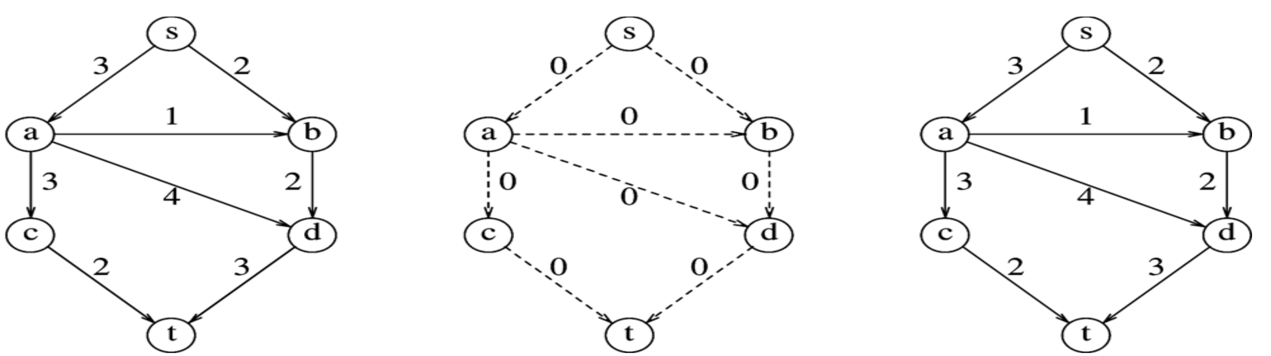
v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	T	3	v_4
v_6	F	8	v_3
v_7	F	5	v_4

v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	T	3	v_4
v_6	F	6	v_7
v_7	T	5	v_4

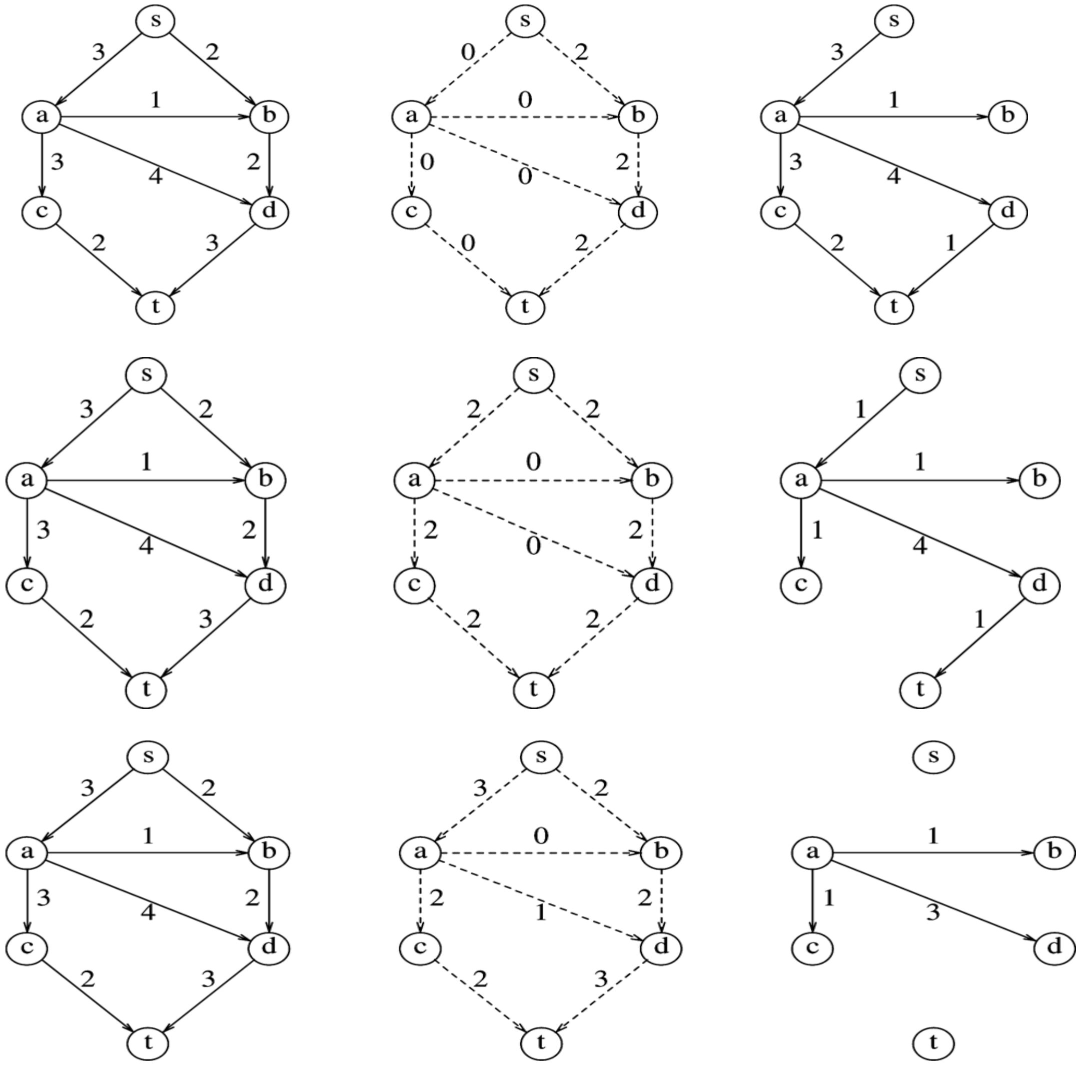
v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	T	3	v_4
v_6	T	6	v_7
v_7	T	5	v_4

Network Flow Problems

- Given a directed graph G = (V, E) with **edge capacities** $c_{v,w}$
- Give two vertices `s`, called the **source**, and `t`, called the **sink**
- Through any edge, (v, w), at most $c_{v,w}$ units of "flow" may pass
- At any vertex `v`, that is not either `s` or `t`, the total flow coming in must equal the total flow coming out



Maximum Flow Algorithm



Augmenting path (s, b, d, t)

Augmenting path (s, a, c, t)

Augmenting path (s, a, d, t)

Network graph **Flow graph (f = 5)** **Residual graph**

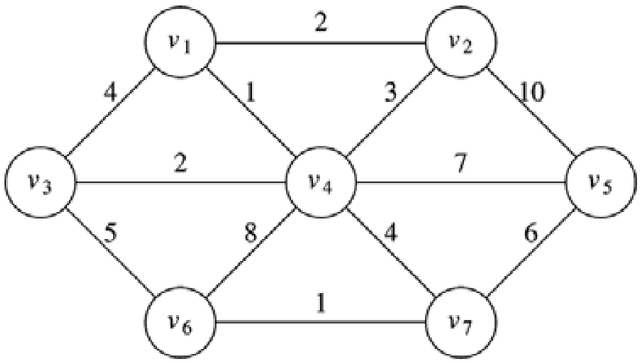
Minimum Spanning Tree Problem

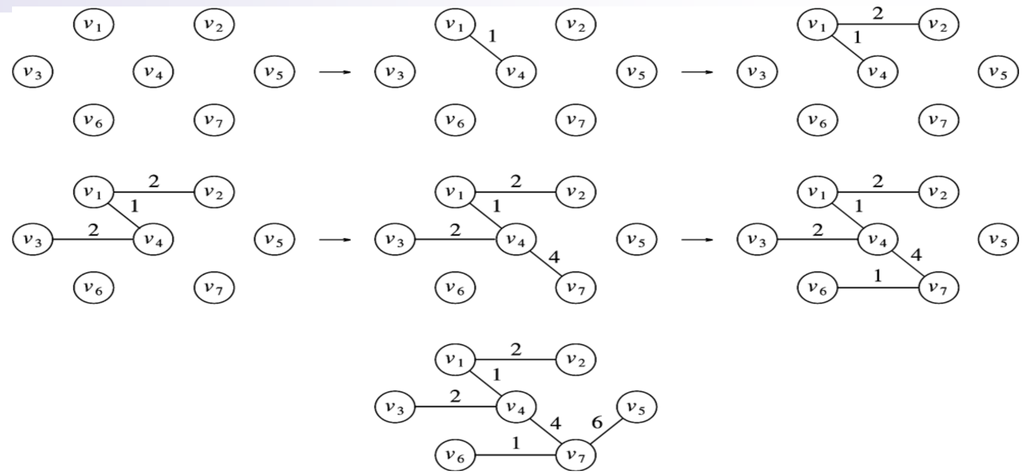
- Find a minimum-cost set of edges that *connect all vertices* of a graph at *lowest total cost*
- Solution # 1: Prim's Algorithm:
  - $O(|V|^2)$ without heap
  - $O(|E| \log |V|)$ using binary heaps
- Solution # 2: Kruskal's Algorithm: $O(|E| \log |V|)$

Prim's Algorithm

- Solution #1 (Prim's Algorithm (1957))
  - Start with an empty tree T
  - T = {x}, where x is an arbitrary node in the input graph
  - While T is not a spanning tree
    - Find the lowest-weight edge that connects a vertex in T to a vertex not in T
    - Add this edge to T
  - T will be a minimum spanning tree

Prim's Algorithm: Example





v	*known*	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	4	v_1
v_4	F	1	v_1
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	2	v_4
v_4	T	1	v_1
v_5	F	7	v_4
v_6	F	8	v_4
v_7	F	4	v_4

v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	F	7	v_4
v_6	F	5	v_3
v_7	F	4	v_4

v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	F	6	v_7
v_6	F	1	v_7
v_7	T	4	v_4

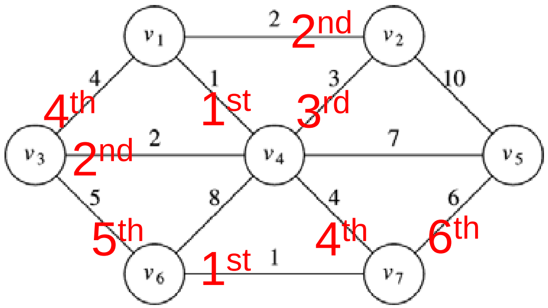
v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	T	6	v_7
v_6	T	1	v_7
v_7	T	4	v_4

Kruskal's algorithm

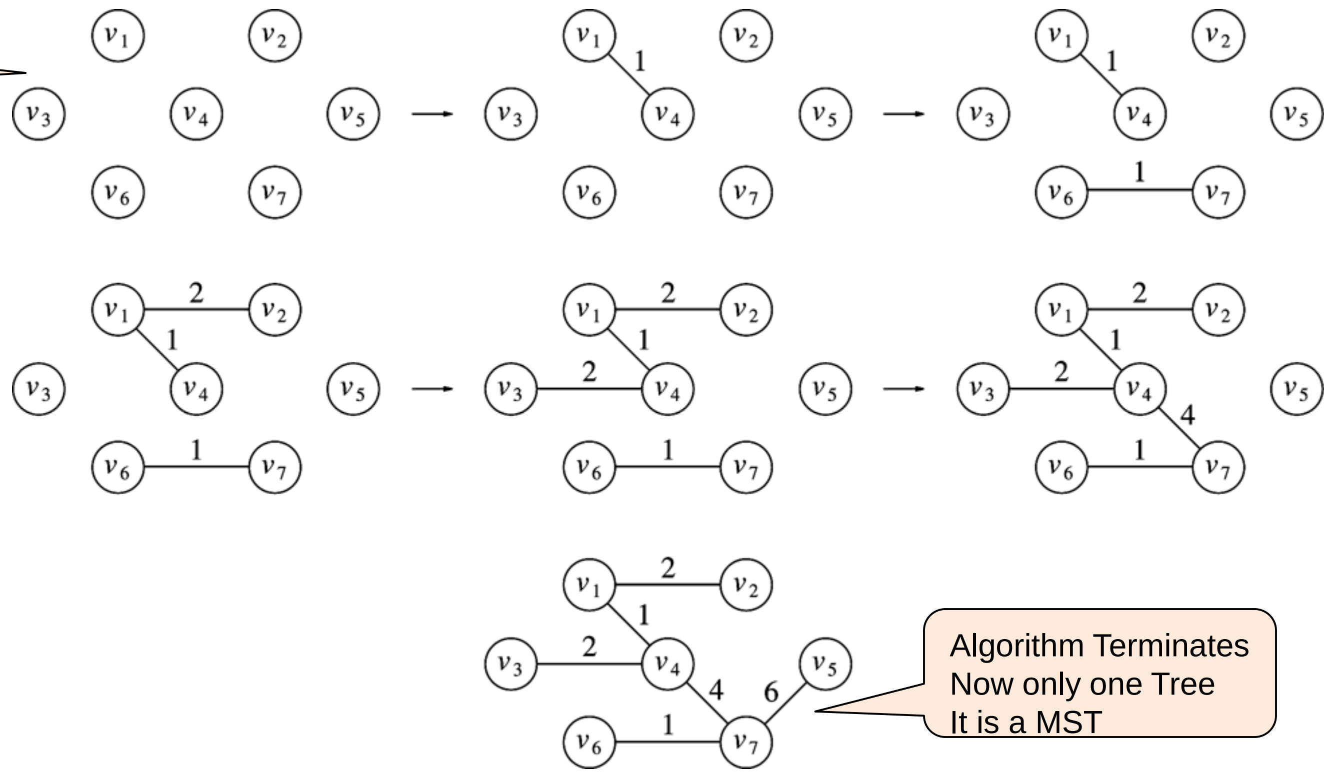
- Solution #2 (Kruskal's algorithm (1956))
  - Start with T = V (with no edges)
  - For each edge in increasing order by weight
    - If adding edge to T does not create a cycle
    - Then add edge to T
  - T will be a minimum spanning tree

Kruskal’s Algorithm: Example

Input graph:



Collection of Trees
|V| single-node Trees



Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

Algorithm Terminates
Now only one Tree
It is a MST

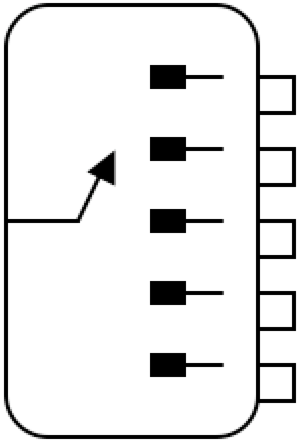
Queueing Theory

Characteristics of Queueing Process

- Arrival patterns of customers
- Service patterns of servers
- Queue disciplines
- System capacity
- Number of service channel
- Number of service stages

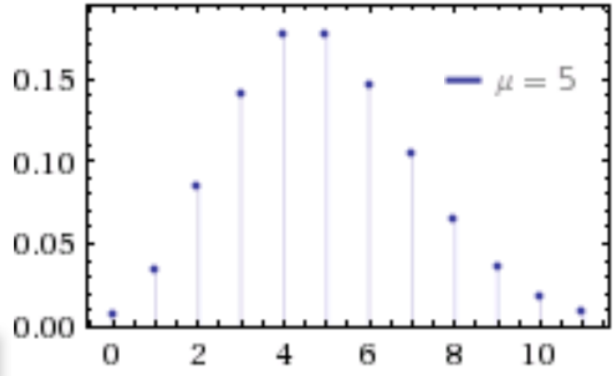
Solution

Now, that we know there are 5 calls/hr on average in the busy-hour-time, we should just put 5 trunk lines. ***RIGHT?***



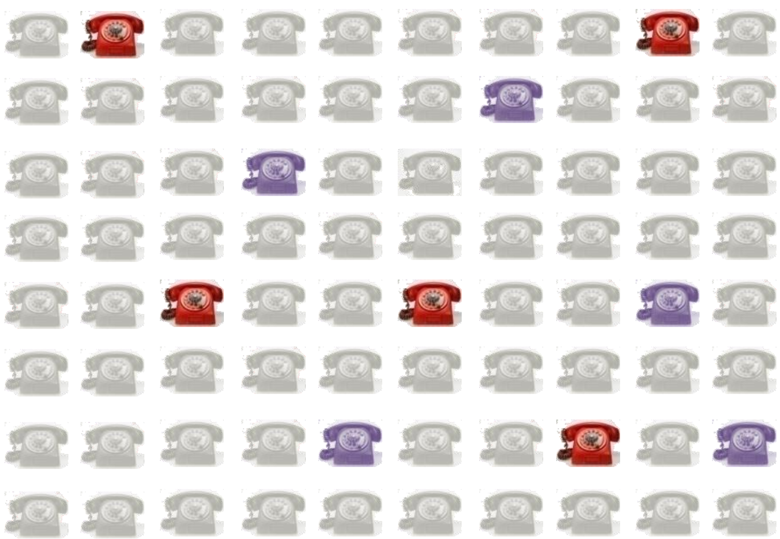
WRONG!
Calls bunch up!

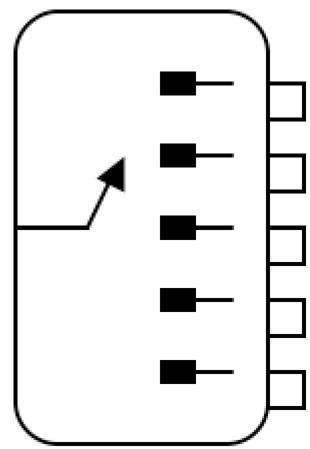
The number of calls in a hour is a random variable and follows a Poisson distribution with expected value of 5 calls/hour

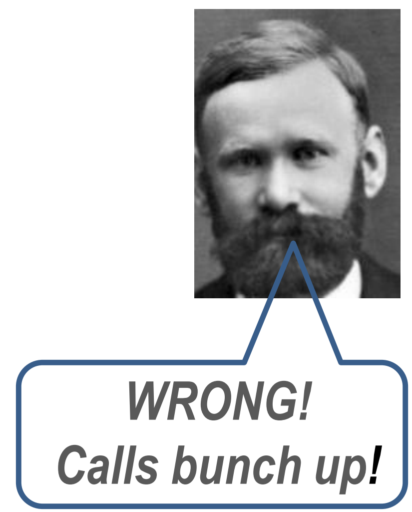


Solution

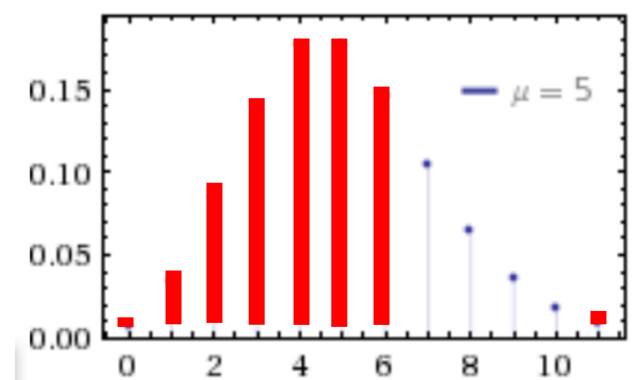
Now, that we know there are 5 calls/ hr on average in the busy-hour-time, we should just put 5 trunk lines. ***RIGHT?***







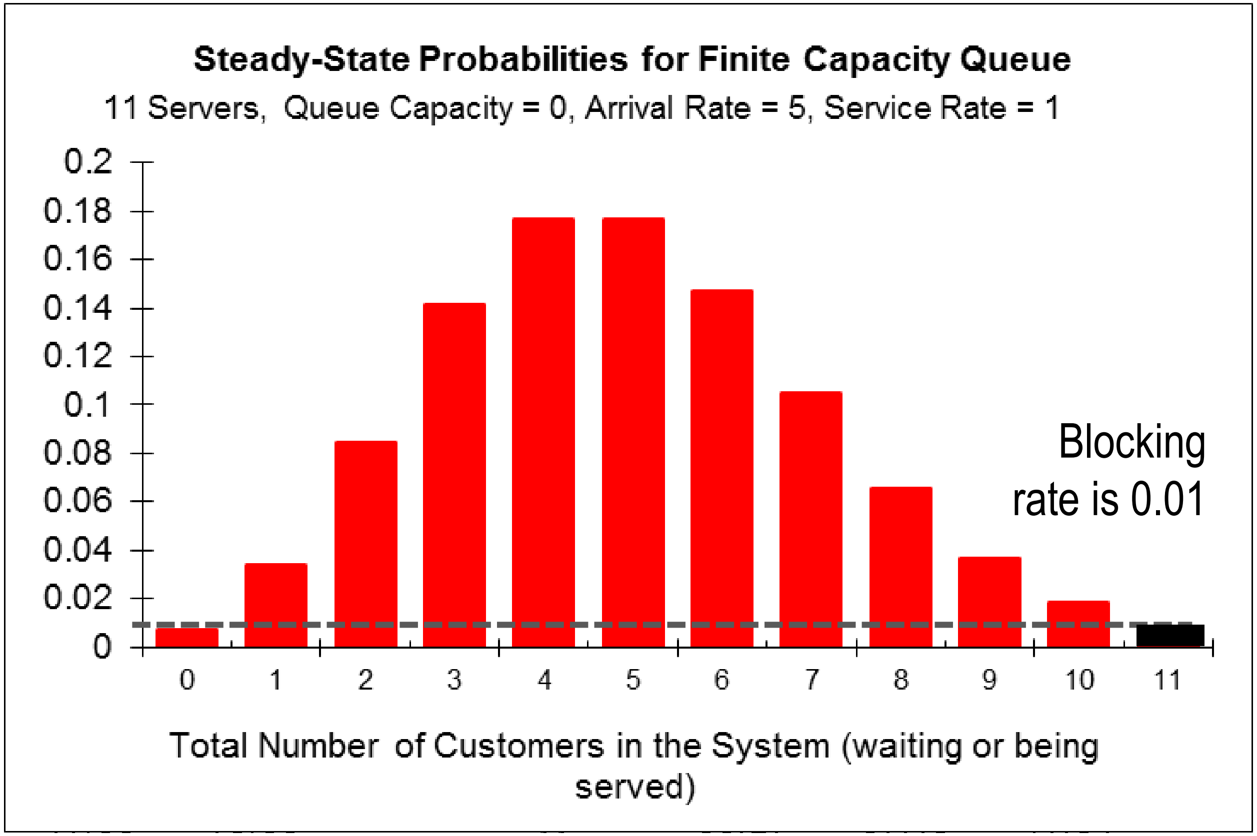
The number of calls in a hour is a random variable and follows a Poisson distribution with expected value of 5 calls/ hour



Erlang B Table

11

Ch	1%						
7	2.50						
8	3.12						
9	3.78						
10	4.46						
11	5.16						
12	5.87						
13	6.60						
14	7.35						
15	8.10						
16	8.87						
17	9.65						
18	10.43						
19	11.23						
20	12.03						
21	12.83						
22	13.65						
23	14.47						
24	15.29						
25	16.12						
26	16.95	18.38	20.94	48	36.11	38.39	42.54
27	17.97	19.26	21.90	49	37.00	39.32	43.53
28	18.64	20.15	22.86	50	37.90	40.25	44.53



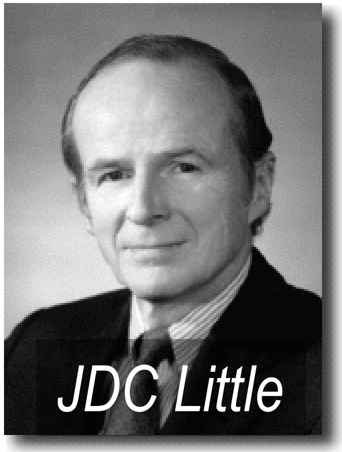
To carry 5 Erlangs of traffic (5 calls/ hr with average call duration of 1 hour) with blocking probability of 0.01 only, 11 trunk lines are needed

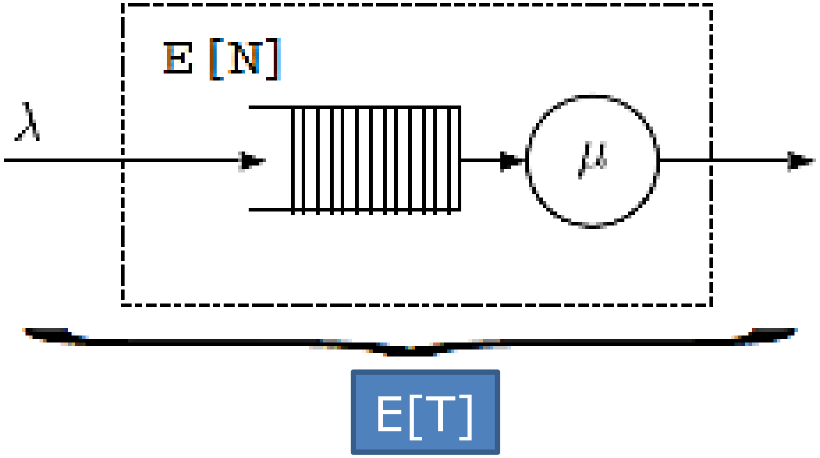
Little's Theorem

***L**ength = **A**rrival-rate x **W**ait-time*

Little's law states that time-average of queue length is equal to the product of the arrival rate and the customer-average waiting time (response time)

$$L = E[N] = \lambda.E[T] = \lambda.W$$





Little's Theorem

- Little's formulas are:
  - $L = \lambda W$
  - $L_q = \lambda W_q$
- L_q = mean # of customer in the queue
- L = mean # of customer in the system
- λ = arrival rate
- $E[T] = E[T_q] + E[S] \Rightarrow W = W_q + 1/\mu$

Summary of general results for G/G/c queues

$\rho = \frac{\lambda}{c\mu}$ Traffic intensity; offered work load rate to a server

$L = \lambda W$ Little's formula

$L_q = \lambda W_q$ Little's formula

$W = W_q + 1/\mu$ Expected-value argument

$p_b = \lambda/c\mu = \rho$ Busy probability for an arbitrary server

$r = \frac{\lambda}{\mu}$ offered work load rate

$L = L_q + r$

$p_0 = 1 - \rho$ G/G/1 empty system probability

$L = L_q + (1 - p_0)$

M/M/1 queue Steady State Solution

- The full steady state solution for M/M/1 system is the *geometric* probability function

$$p_n = (1-\rho)\rho^n \qquad (\rho = \frac{\lambda}{\mu} < 1)$$

- Note the existence of a steady state solution depends on the condition that $\rho < 1$
  - Equivalently $\lambda < \mu$
  - Intuitively if $\lambda > \mu$ the mean arrival rate > mean service rate
    - Server gets further and further behind; system size increases without bound over time
- Why there is no steady state solution when $\lambda = \mu$

M/M/1 queue Steady State Solution

- Why there is no steady state solution when $\lambda = \mu$
  - Infinite build up
  - As the queue grows it is more and more difficult for the server to decrease the queue
  - Average service rate is *no higher than* the average arrival rate

Tentative Exam Structure

- Short Multiple Choice Questions
  - 10 multiple choice questions with 2 points each
- Short & Medium Type Questions
  - 10 to 12 short questions with 5 points each
  - 2 to 3 medium questions with 10 points each

- When: Tuesday (11/28) 7:10 pm – 9:10 pm (2 hours)
- Where: In Class

Conclusion

Please take a few minutes to complete the online course evaluations.

Thank you for taking IS 709/809.

Good Luck!