ANALYSIS OF ALGORITHMS

ALGORITHMS

- Algorithm - sequence of steps
- Exact algorithm - verifiable optimal solution
- Heuristic algorithm - feasible solution

ANALYSIS OF EXACT ALGORITHMS

- Good, bad, and better algorithms
- Performance measures
  - Static measures
    - Number of lines of code
  - Dynamic measures
    - Number of basic computations
    - Running time
    - Number of data storage locations

NETWORK OPTIMIZATION PROBLEMS

DEFINITION: *Graph*

- A *graph* G(N, A) consists of a set N of nodes and a set A of unordered pairs of nodes called arcs.

DEFINITION: *Directed Arc*

- If the arc x - y $\in$ A with x, y $\in$ N implies a direction of flow between x and y, then the arc is said to be *directed*.
- If the arc may be traversed in either direction, then it is said to be *undirected*.

DEFINITION: *Network*

- If each arc in a graph has a number (cost, distance, capacity) associated with it, the graph is termed a *network*.

DEFINITION : *Network Optimization Problem*

- A *network optimization problem* typically involves the optimization of some objective function subject to constraints that specify a particular network structure.

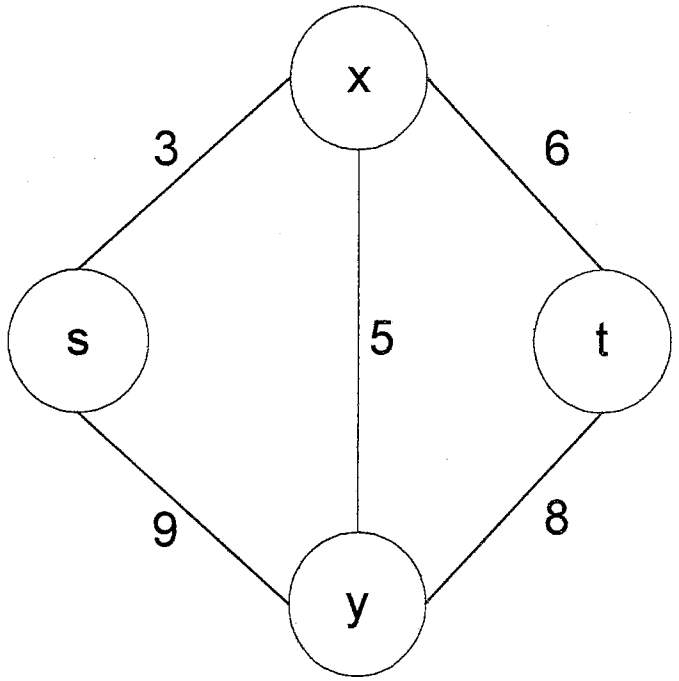
EXAMPLE

Shortest path problem with

N = {s, x, y, t} and

A = {s - x, s - y, x - y, x - t, y - t}.

s is origin, t is destination.



STORAGE REQUIREMENTS

■ Matrix Representation

Node Data

Entry Position	Nodes
1	s
2	x
3	y
4	t

	1	2	3	4
1	—	3	9	∞
2	3	—	5	6
3	9	5	—	8
4	∞	6	8	—

■ Straightforward Data Structure

Node Data		Arc Data			
Entry Position	Nodes	Entry Position	Start	End	Length
1	s	1	s	x	3
2	x	2	s	y	9
3	y	3	x	y	5
4	t	4	x	t	6
		5	y	t	8

■ Forward Star Data Structure

Node Data			Arc Data		
Entry Position	Nodes	Pointer	Entry Position	End	Length
1	s	1---------->	1	x	3
			2	y	9
2	x	3---------->	3	y	5
			4	t	6
3	y	5---------->	5	t	8
4	t	—			

COMPUTATIONAL REQUIREMENTS

- ♦ Running time vs. number of elementary operations
- ♦ Worst case time complexity

DEFINITION : *Worst Case Time Complexity*.

- Given a worst case time complexity function f(n), for a particular algorithm, the complexity of the algorithm is said to be of order g(n), denoted O(g(n)), if $f(n) \leq k\, g(n)$ for some constant $k > 0$ and for large enough n.

EXAMPLE

An algorithm requiring $3n^2 + 2n + 7$ operations is $O(n^2)$.

EXAMPLES OF WORST CASE TIME COMPLEXITY

- Searching an *unordered* list of n items.
  - Given a list of n items $\{I_1, I_2, \ldots, I_n\}$, determine whether or not the list contains a particular value. Does $L = \{7, 9, 2, 11, 1, 3, 10\}$ contain a 6? In the worst case, all elements must be examined $=>$ O(n).

- Searching an *ordered* list of n items.
  - Given an ordered list of n items, $\{I_1, I_2, \ldots, I_n\}$ where $I_j \leq I_k$ for $j < k$, determine whether or not the list contains a particular value. Does $L = \{1, 2, 3, 7, 9, 10, 11\}$ contain a 6?
  - Check median item. Since $7 > 6$, we are left with $\{1, 2, 3\}$.
  - Check median value. Since $2 < 6$, we are left with $\{3\}$. Since $3 \neq 6$, we are done.
  - Since at each step, half the items can be eliminated, the worst case time complexity is $O(\log_2 n)$.

Sorting a list of n items.

- Given an unordered list of n items, $\{I_1, I_2, \ldots, I_n\}$, sort the list into non-decreasing order (i.e., so that $I_j \leq I_k$ whenever $j < k$).
- Sort by considering the movement of each of the n items from the unordered list one at a time to its appropriate position on a sorted list that is being constructed.

$\{7, 9, 2, 11, 1, 3, 10\} \rightarrow$

$\{7\}$
$\{7, 9\}$
$\{2, 7, 9\}$
$\{2, 7, 9, 11\}$
$\{1, 2, 7, 9, 11\}$
$\{1, 2, 3, 7, 9, 11\}$
$\{1, 2, 3, 7, 9, 10, 11\}$

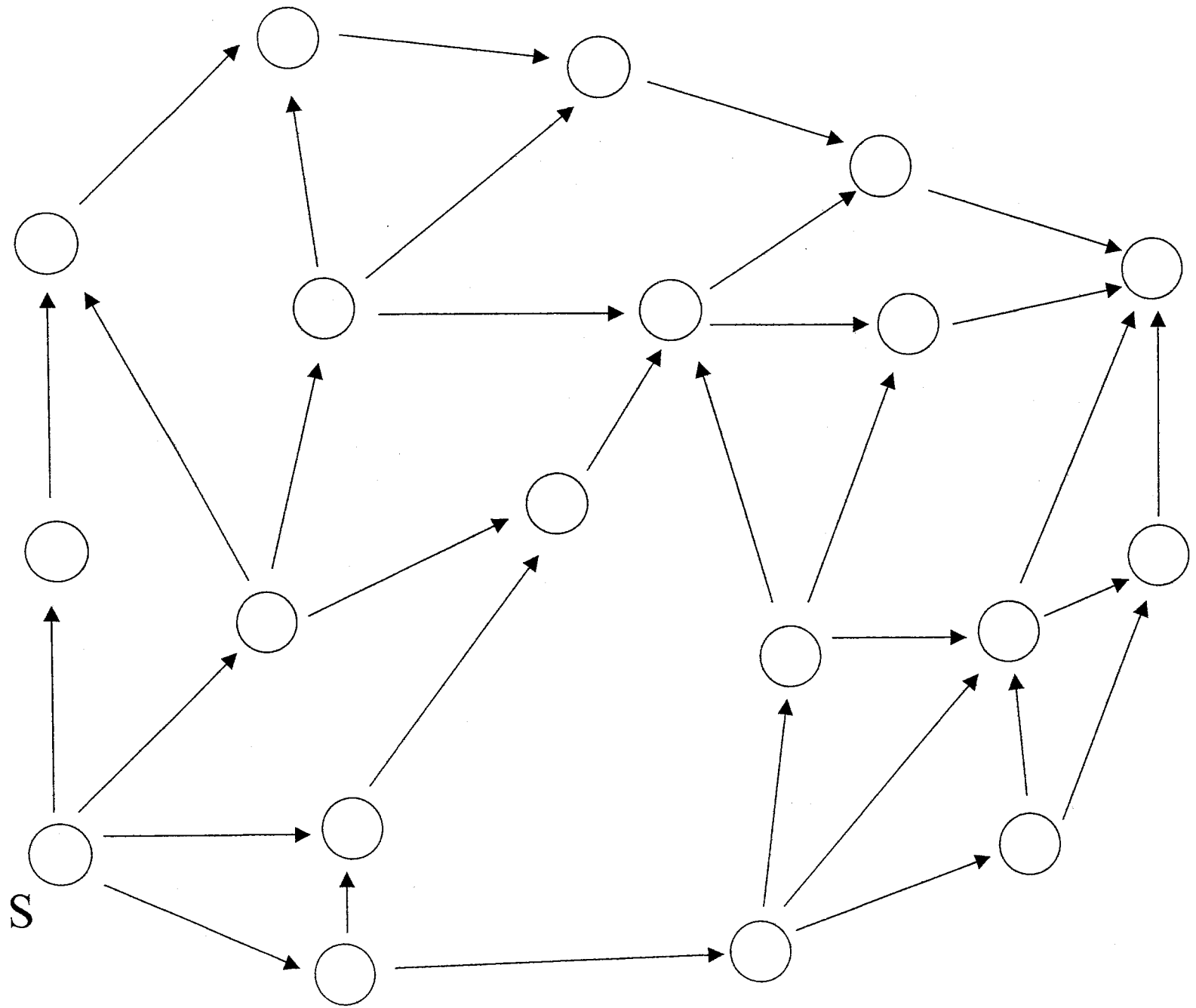
$$\text{Amount of work} = \sum_{x=1}^{n} \log_2 x \leq \sum_{x=1}^{n} \log_2 n = n \log_2 n \Rightarrow O(n \log_2 n).$$

Finding a path between two specific nodes

Given a network of n nodes and m arcs, we wish to find any path from a particular node s to another particular node t.

FAN-OUT PROCEDURE

- Starting from node s, "label" all nodes x that are terminal nodes of an arc s - x beginning at node s.
- Select any of the labeled nodes x and fan out from it. Continue in this way, fanning out from each labeled node exactly once until either labeling node t or running out of labeled nodes to fan out from (in which case, there is no path from node s to node t).
- Store the predecessor p(x) from each node x (that is, the node from which we labeled it) as we proceed, then trace predecessors back to node s to identify a path from s to t.
- Computational complexity is O(m) (each arc is visited at most once) which, since $m \leq n^2$, is no more than $O(n^2)$.



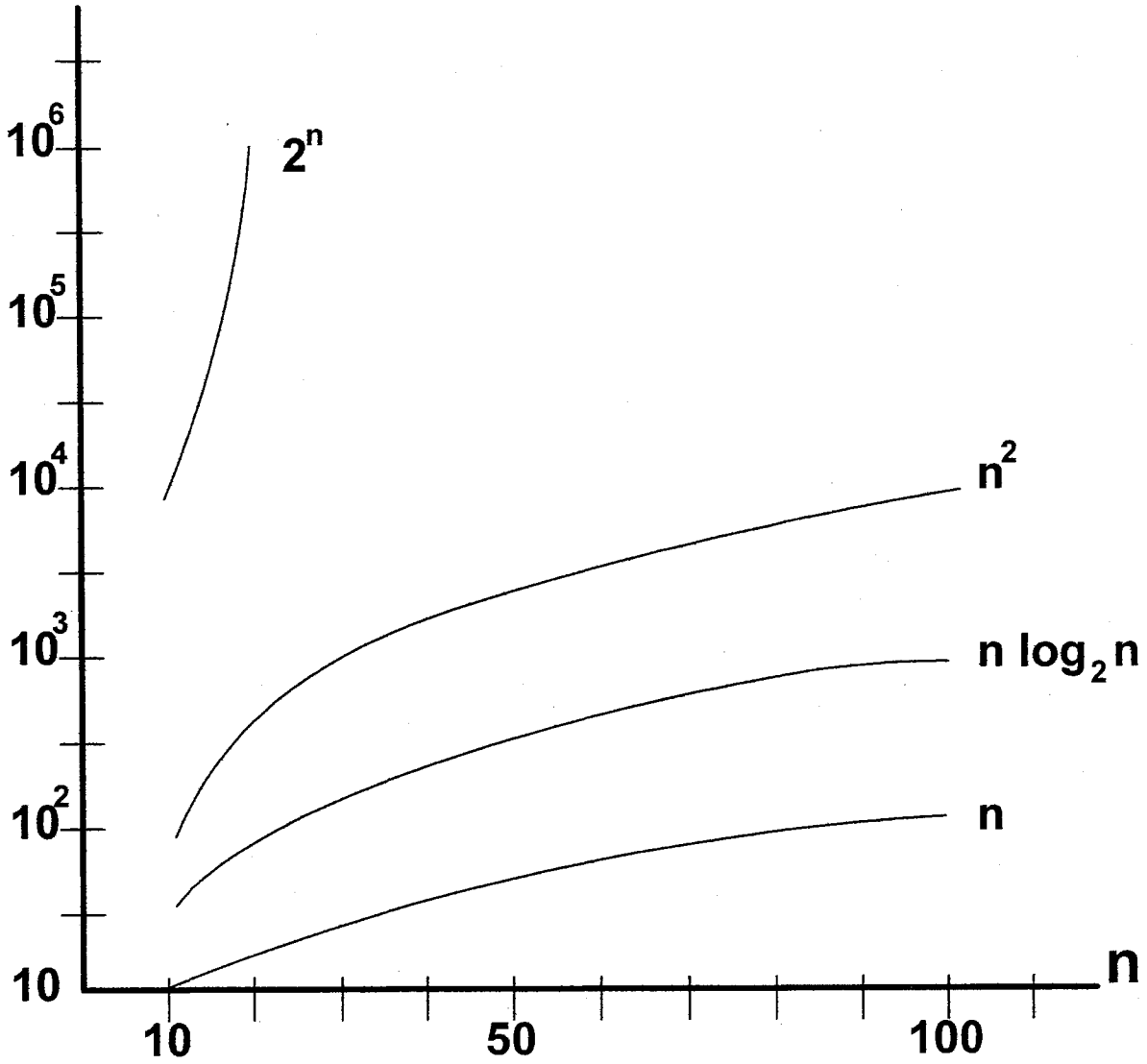
4-9a

Finding a path that visits each node exactly once

HAMILTONIAN PATHS AND CYCLES

- A path in a network that visits each node exactly once is called a *Hamiltonian path*.
- If the path is constructed so that it begins and ends at the same node, then it is called a *Hamiltonian cycle*.
- The well-known *traveling salesman problem* seeks a Hamiltonian cycle with minimum length as measured by the total length of arcs in the cycle.
- Since the path must visit each node in the network, finding a Hamiltonian path (or solving the traveling salesman problem) is more difficult than finding a shortest path joining two specified nodes.
- No algorithm exists that has polynomial time complexity. The best algorithms for solving these "harder" problems require $O(2^n)$ computations in the worst case.

WORST CASE COMPLEXITY AS A FUNCTION OF PROBLEM SIZE



THE GROWTH OF POLYNOMIAL AND EXPONENTIAL FUNCTIONS

Function	Approximate Values		
n	10	100	1000
$n \log_2 n$	33	664	9966
n^3	1000	1,000,000	10^9
$10^6 n^8$	10^{14}	10^{22}	10^{30}
2^n	1024	1.27×10^{30}	1.05×10^{301}
$n^{\log_2 n}$	2099	1.93×10^{13}	7.89×10^{29}
$n!$	3,628,800	10^{158}	4.00×10^{2567}

POLYNOMIAL-TIME ALGORITHMS TAKE BETTER ADVANTAGE OF TECHNOLOGY

Function	Size of instance solved in one day	Size of instance solved in one day on a computer *10 times* faster	Size of instance solved in one day on a computer *1000 times* faster
n	10^{12}	10^{13}	10^{15}
$n \log_2 n$	9.48×10^{10}	8.7×10^{11}	7.2×10^{13}
n^2	10^6	3.16×10^6	3.16×10^7
n^3	10^4	2.15×10^4	10^5
$10^8 n^4$	10	18	56
2^n	40	43	50
10^n	12	13	15
$n^{\log_2 n}$	79	95	133
$n!$	14	15	17

GOOD, BAD, AND UGLY ALGORITHMS

- Researchers have emphasized the importance of finding algorithms of polynomial time complexity, by referring to all such polynomial algorithms as inherently good.
- Algorithms that are *not* polynomially bounded, including the class of exponential algorithms, are labeled inherently bad.

GOOD ALGORITHMS EXIST FOR THESE PROBLEMS

- Minimal spanning trees
- Shortest paths
- Transportation problem
- Matching
- Linear programming (Khachiyan, Karmarkar)
- Maximum flow
- Minimum cost flow

GOOD ALGORITHMS DON'T CURRENTLY EXIST FOR THESE PROBLEMS

- Capacitated minimal spanning tree
- Traveling salesman problem
- Vehicle routing problem
- General integer programming problem
- Bin packing problem
- Knapsack problem
- Multicommodity flow in integers

RELATED ISSUES

- The classes P, NP, and NP-complete
- Linear Programming