Merge Sort: Shortcomings

- Merging A and B creates a new array C
  - No obvious way to efficiently merge in place
- Extra storage can be costly
- Inherently recursive
  - Recursive call and return are expensive

Alternative approach

- Extra space is required to merge
- Merging happens because elements in left half must move right and vice versa
- Can we divide so that everything to the left is smaller than everything to the right?
  - No need to merge!

Divide and conquer without merging

- Suppose the median value in A is m
- Move all values ≤ m to left half of A
  - Right half has values > m
  - This shifting can be done in place, in time O(n)
- Recursively sort left and right halves
- A is now sorted! No need to merge
  - $t(n) = 2t(n/2) + n = O(n \log n)$

Divide and conquer without merging

- How do we find the median?
  - Sort and pick up middle element
  - But our aim is to sort!
- Instead, pick up some value in A — pivot
  - Split A with respect to this pivot element

Quicksort

- Choose a pivot element
  - Typically the first value in the array
- Partition A into lower and upper parts with respect to pivot
- Move pivot between lower and upper partition
- Recursively sort the two partitions

Quicksort

- High level view

43	32	22	78	63	57	91	13

Quicksort

- High level view

Quicksort

- High level view

43	32	22	78	63	57	91	13

Quicksort

- High level view

43	32	22	78	63	57	91	13

Quicksort

- High level view

13	22	32	43	57	63	78	91

Quicksort

- High level view

13	32	22	43	63	57	91	78

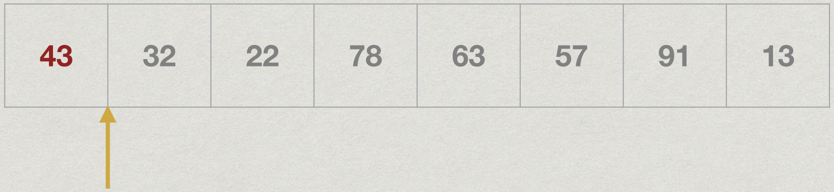
Quicksort: Partitioning

Quicksort: Partitioning

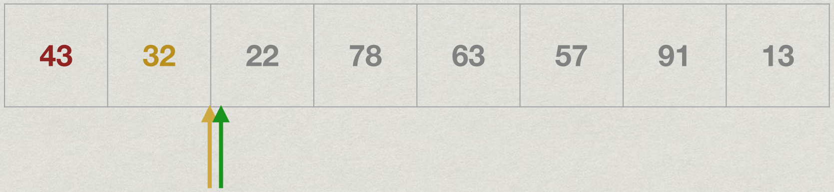
43	32	22	78	63	57	91	13

Quicksort: Partitioning

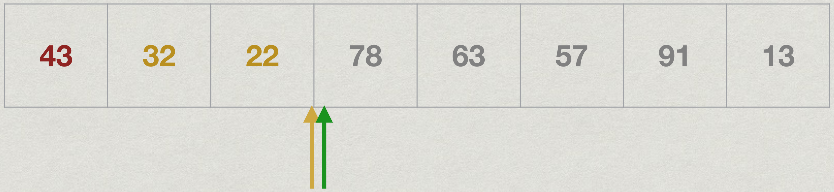
Quicksort: Partitioning



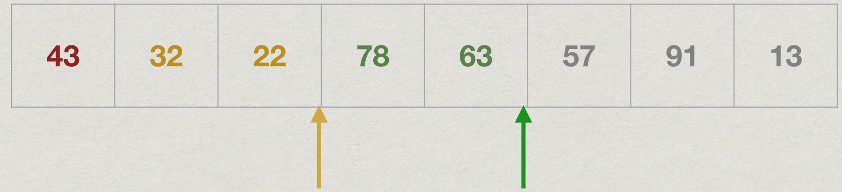
Quicksort: Partitioning



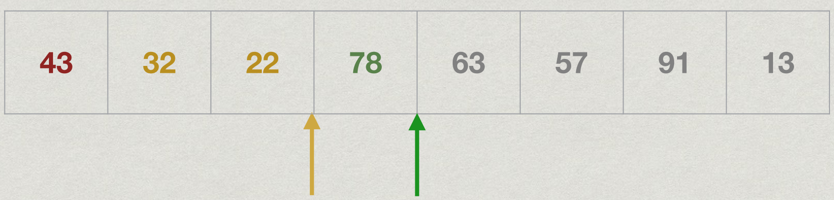
Quicksort: Partitioning



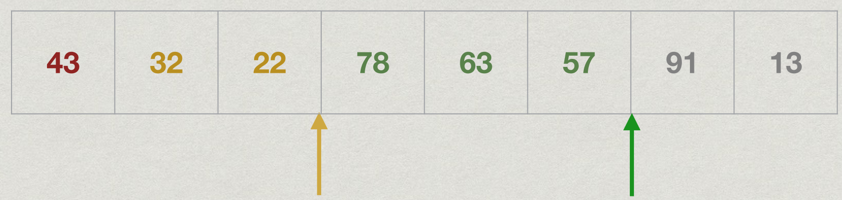
Quicksort: Partitioning



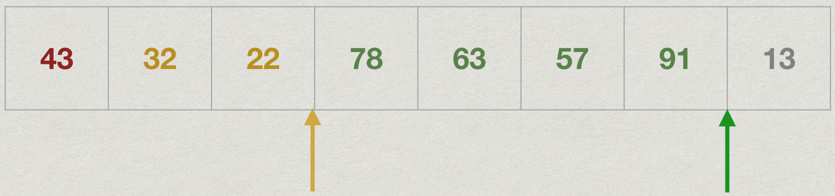
Quicksort: Partitioning



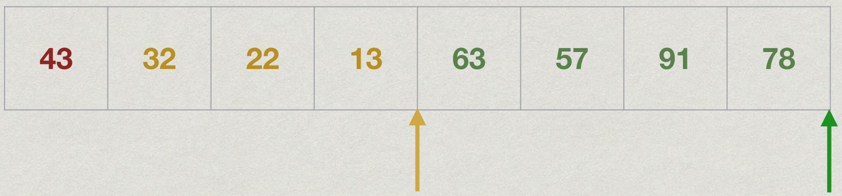
Quicksort: Partitioning



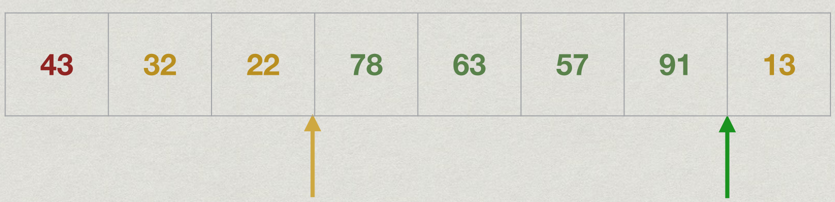
Quicksort: Partitioning



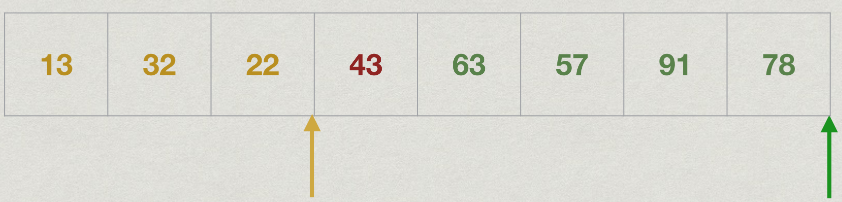
Quicksort: Partitioning



Quicksort: Partitioning



Quicksort: Partitioning



Quicksort: Implementation

```
Quicksort(A,l,r) // Sort A[l..r-1]

   if (r - l <= 1)) return; // Base case

   // Partition with respect to pivot, a[l]
   yellow = l+1;
   for (green = l+1; green < r; green++)
      if (A[green] <= A[l])
         swap(A,yellow,green);
         yellow++;

   swap(A,l,yellow-1); // Move pivot into place

   Quicksort(A,l,yellow);  // Recursive calls
   Quicksort(A,yellow+1,r);
```

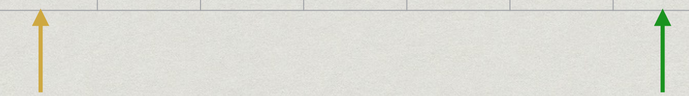
Quicksort: Another Partitioning Strategy

43	32	22	78	63	57	91	13

Quicksort: Another Partitioning Strategy

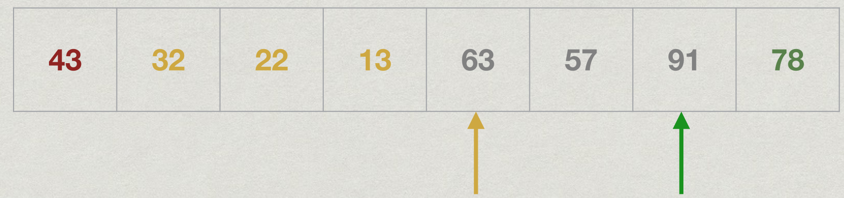
Quicksort: Another Partitioning Strategy

43	32	22	78	63	57	91	13

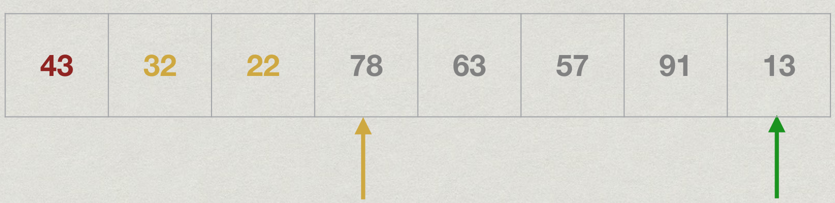


Quicksort:
Another Partitioning Strategy

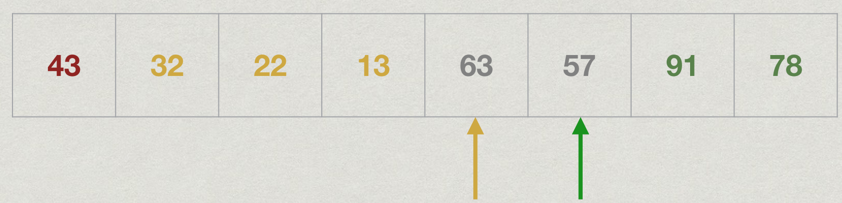
Quicksort:
Another Partitioning Strategy



Quicksort:
Another Partitioning Strategy



Quicksort:
Another Partitioning Strategy



Quicksort: Another Partitioning Strategy

43	32	22	13	63	57	91	78

Quicksort: Another Partitioning Strategy

13	32	22	43	63	57	91	78

Quicksort: Another Partitioning Strategy

43	32	22	13	63	57	91	78

Quicksort

- Choose a pivot element
  - Typically the first value in the array
- Partition A into lower and upper parts with respect to pivot
- Move pivot between lower and upper partition
- Recursively sort the two partitions

Analysis of Quicksort

- Partitioning with respect to pivot takes O(n)
- If pivot is median
  - Each partition is of size n/2
  - $t(n) = 2t(n/2) + n = O(n \log n)$
- Worst case?

Analysis of Quicksort

But …

- Average case is O(n log n)
  - Sorting is a rare example where average case can be computed
- What does average case mean?

Analysis of Quicksort

Worst case

- Pivot is maximum or minimum
  - One partition is empty
  - Other is size n-1
  - $t(n) = t(n-1) + n = t(n-2) + (n-1) + n$
    $= \ldots = 1 + 2 + \ldots + n = O(n^2)$
- Already sorted array is worst case input!

Quicksort: Average case

- Assume input is a permutation of {1,2,…,n}
  - Actual values not important
  - Only relative order matters
  - Each input is equally likely (uniform probability)
- Calculate running time across all inputs
- Expected running time can be shown O(n log n)

Quicksort: randomization

- Worst case arises because of fixed choice of pivot
  - We chose the first element
  - For any fixed strategy (last element, midpoint), can work backwards to construct $O(n^2)$ worst case
- Instead, choose pivot randomly
  - Pick any index in [0..n-1] with uniform probability
- Expected running time is again O(n log n)

Quicksort in practice

- In practice, Quicksort is very fast
  - Typically the default algorithm for in-built sort functions
    - Spreadsheets
    - Built in sort function in programming languages

Iterative Quicksort

- Recursive calls work on disjoint segments of array
  - No recombination of results required
- Can use an explicit stack to simulate recursion
  - Stack only needs to store left and right endpoints of interval to be sorted

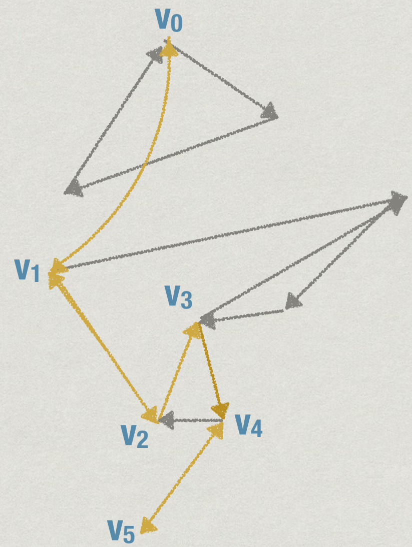
Graphs, formally

G = (V,E)

- Set of vertices V
- Set of edges E
  - E is a subset of pairs (v,v'): $E \subseteq V \times V$
  - Undirected graph: (v,v') and (v',v) are the same edge
  - Directed graph:
    - (v,v') is an edge from v to v'
    - Does not guarantee that (v',v) is also an edge

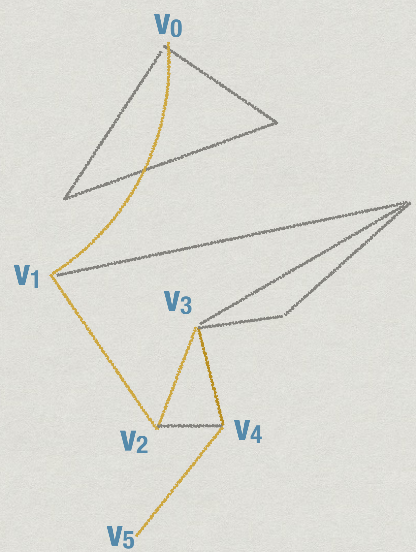
Finding a route

- Directed graph
- Find a sequence of vertices v_0, v_1, …, v_k such that
  - v_0 is New Delhi
  - Each (v_i,v_{i+1}) is an edge in E
  - v_k is Trivandrum



Finding a route

- Also makes sense for undirected graphs
- Find a sequence of vertices v_0, v_1, …, v_k such that
  - v_0 is New Delhi
  - Each (v_i,v_{i+1}) is an edge in E
  - v_k is Trivandrum



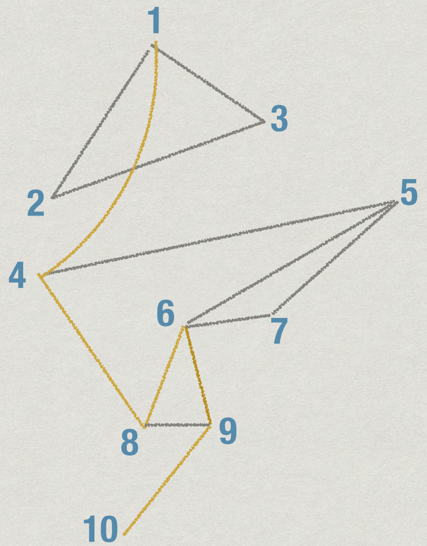
Working with graphs

- We are given G = (V,E), **undirected**
- Is there a path from source v_s to target v_t?
- Look at the picture and see if v_s and v_t are connected
- How do we get an algorithm to “look at the picture”?

Representing graphs

- Let V have n vertices
  - We can assume vertices are named 1,2,…,n
- Each edge is now a pair (i,j), where $1 \leq i,j \leq n$
- Let A(i,j) = 1 if (i,j) is an edge and 0 otherwise
- A is an n x n matrix describing the graph
  - **Adjacency matrix**

Adjacency matrix



	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Adjacency matrix

- Neighbours of i
  - Any column j in row i with entry 1
  - Scan row i from left to right to identify all neighbours
- Neighbours of 4 are {1,5,8}

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Adjacency matrix

- Neighbours of i
  - Any column j in row i with entry 1
  - Scan row i from left to right to identify all neighbours
- Neighbours of 4 are {1,5,8}

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Finding a path

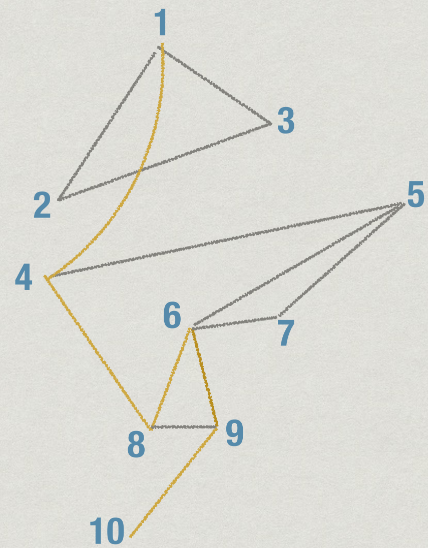
- Start with v_s
  - New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
  - v_t =10 = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Exploring graphs

- Need a systematic algorithm
  - Mark vertices that have been visited
  - Keep track of vertices whose neighbours have already been explored
    - Avoid going round indefinitely in circles
- Two fundamental strategies: breadth first and depth first

Adjacency list



- For each vertex, maintain a list of its neighbours

1	2,3,4
2	1,3
3	1,2
4	1,5,8
5	4,6,7
6	5,7,8,9
7	5,6
8	4,6,9
9	6,8,10
10	9

An alternative representation

- Adjacency matrix has many 0's
- Size of the matrix is n^2 regardless of number of edges
- Maximum size of E is n(n-1)/2 if we disallow self loops
- Typically E is much smaller

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Comparing representations

- Adjacency list typically requires less space
- **Is j a neighbour of i?**
  - Just check if A[i][j] is 1 in adjacency matrix
  - Need to scan neighbours of i in adjacency list
- **Which vertices are neighbours of i?**
  - Scan all n columns in adjacency matrix
  - Takes time proportional to neighbours in adjacency list