Suppose we were to write a class implementing a set of integers using a sorted array, that is

```
public class SortedIntSet implements Set<Integer> {
    // invariant: The range [0, size) in internal is filled and sorted
    private int[] internal;
    private int size;
    public SortedInSet() {
        internal = new int[100];
        size = 0;
    }
    // allocate new internal with double size
    private int grow() { ... }  // allocate new internal with double size
    ....
}
```

For each of the following methods required by the Set interface, determine the *worst case running time* of the *best implementation* that maintains the invariant indicated in the comments. Cite the running times as big-Oh categories in terms of n, the number of items in the set at the time the method is called.

1. `add(Integer item)`
2. `contains(Integer item)`
3. `remove(Integer item)`
4. `size()`
5. `isEmpty()`

Minimum Spanning Tree Problem

Given a weighted, undirected graph, find the tree with least-total weight that connects all the vertices, if one exists.

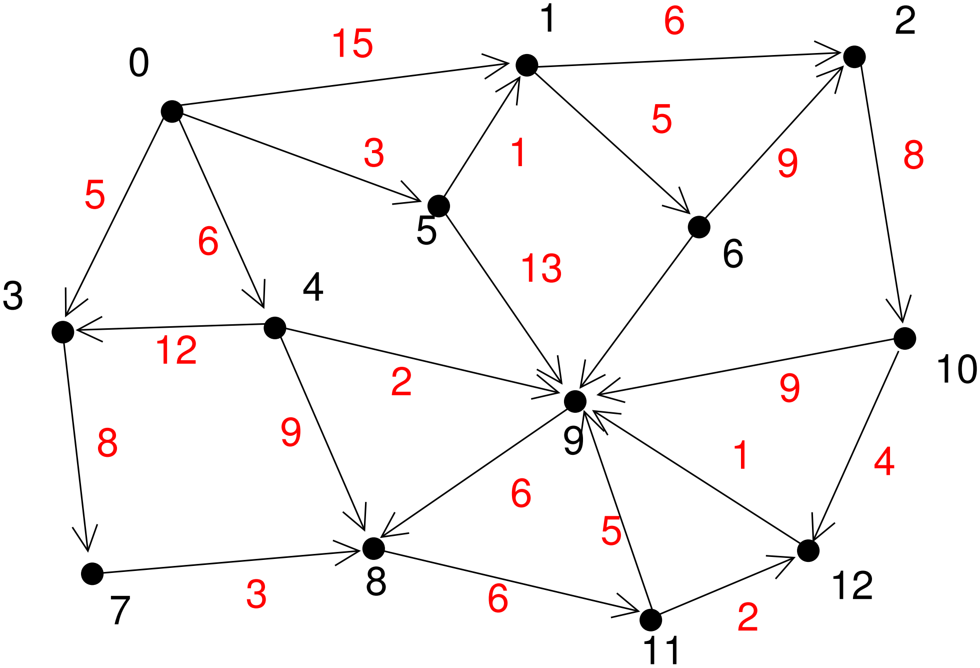
Single-Source Shortest Paths Problem

Given a weighted directed graph and a source vertex, find the tree comprising the shortest paths from that source to all other reachable vertices.

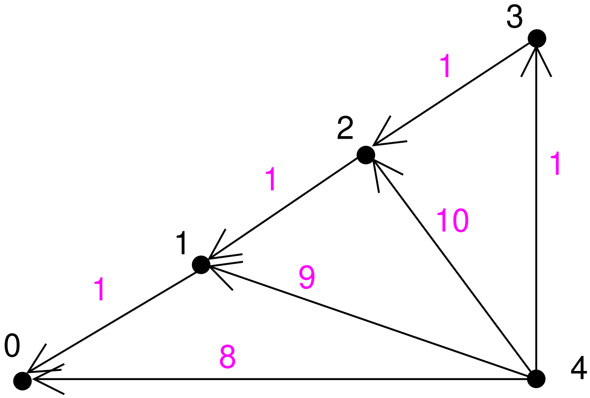
- Both are defined for weighted graphs
- Both produce trees as a result
- Both minmize by weight
- For each we have two algorithms

Input is only a graph
Problem usually is described on an undirected graph
Goal is to minimize total weight
There is no clear winner between the algorithms

Input is a graph and a starting point
Problem usually is described on a directed graph
Goal is to minimize weight on each path
One algorithm is clearly more efficient



1
6
2
15
0
5
3
1
9
8
5
6
5
6
13
3
4
12
2
10
9
9
8
9
1
4
6
5
12
8
3
7
6
2
11


3
1
2
1
1
10
1
9
1
0
8
4