1. Depth-first search.
   (a) A B C F E G D H
   (b) E D G F H C B A
   (c) false, true, true

2. Minimum spanning tree.
   (a) 1 2 3 4 5 11 12 13 17
   (b) 4 3 2 1 5 13 12 17 11

3. Convex hull.
   (a) B C D G H F J I E
   (b) 1. A -> B -> C
       2. A -> B -> C -> D
       3. A -> B -> G
       4. A -> B -> H
       5. A -> B -> H -> F
       6. A -> B -> H -> F -> J
       7. A -> B -> H -> F -> J -> I
       8. A -> B -> H -> E
4. TST.

   (a) aaa aab ab ba bb bba bbbb

   (b) 

```
   o
   |
   +---+---+---+---
   |   |   |   |   |
   |   |   |   |   |
   +---+---+---+---
   |   |   |   |
   +---+---+---
   o   o
``` 

5. 2D tree.
6. **Radix sorting.**

Put an X in each box if the string sorting algorithm (the standard version considered in class) has the corresponding property.

<table>
<thead>
<tr>
<th></th>
<th>mergesort</th>
<th>LSD radix sort</th>
<th>MSD radix sort</th>
<th>3-way radix quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td>stable</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>in-place</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>sublinear time (in best case)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>fixed-length strings only</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that in the best case, mergesort takes $N \log N$ string compares, but a string compare can takes constant time (if the two strings differ after a constant number of characters). Thus, the overall running time can be sublinear in the input size (total number of characters in the $N$ strings).

7. **Data compression.**

```
 a b a b b a b a b a b a b a b b
```
8. Regular expressions.

9. **1D nearest neighbor.**

- **constructor:** create an empty red-black tree.
- **insert(x):** insert x into the red-black tree.
- **query(y):** if the data structure is empty return null; otherwise compute the floor and ceiling of y and return the (non-null) value closest to x.
10. **Prefix-free codes.**

(a) Insert all of the codewords into a binary trie, marking the terminating nodes. The set of string is not prefix-free if when inserting a codeword (i) you pass through a marked node (an existing codeword is a prefix of the codeword you are inserting) or (ii) the node you mark is not a leaf node (the codeword you’re inserting is a prefix of an existing codeword).

(b) \( W \) (we go down one level in the tree for each bit we examine).

(c) \( W \) (at most one trie node for each bit in the input).

Alternate solution 1: use a TST instead of a binary trie. Since the radix size is 2, the running time will still be linear: you at most double the length of a search path for a codeword.

Alternate solution 2: sort all of the codewords (MSD radix sort or 3-way radix quicksort) and check adjacent codewords to see if one is a prefix of the other.

11. **Shortest directed cycle.**

(a) The critical observation is that the shortest directed cycle is a shortest path (number of edges) from \( s \) to \( v \), plus a single edge \( v \rightarrow s \).

For each vertex \( s \):

* Use BFS to compute shortest path from \( s \) to each other vertex.
* For each edge \( v \rightarrow s \) entering \( s \), consider cycle formed by shortest path from \( s \) to \( v \) (if the path exists) plus the edge \( v \rightarrow s \).

Return shortest overall cycle.

(b) The running time is \( O(EV) \).

The single-source shortest path computation from \( s \) takes \( O(E + V) \) time per using BFS. Finding all edges entering \( s \) takes \( O(E + V) \) time by scanning all edges (though a better way is to compute the reverse graph at once and access the adjacency lists). We must do this for each vertex \( s \). Thus, the overall running time is \( O(EV) \).

(c) The memory usage is \( O(E + V) \).

BFS uses \( O(V) \) extra memory and we only need to run one at a time. (A less efficient solution is to compute a \( V \)-by-\( V \) table containing the shortest path from \( v \) to \( w \) for every \( v \) and \( w \). This uses \( O(V^2) \) memory.)

12. **Reductions.**

(a) \( 0, x_1, x_2, \ldots, x_N \).

This is the easy direction—we just choose \( b = 0 \).

(b) \( 3x_1 - b, 3x_2 - b, \ldots, 3x_N - b \).

Observe that \( (3x_i - b) + (3x_j - b) + (3x_k - b) = 0 \) if and only if \( x_i + x_j + x_k = b \). Note that we use \( 3x_i - b \) as the input instead of \( x_i - b/3 \) because the input must be integral.

(c) I, II, and III.