Section 09: MT2 Review

1. Heaps 1: Insertion

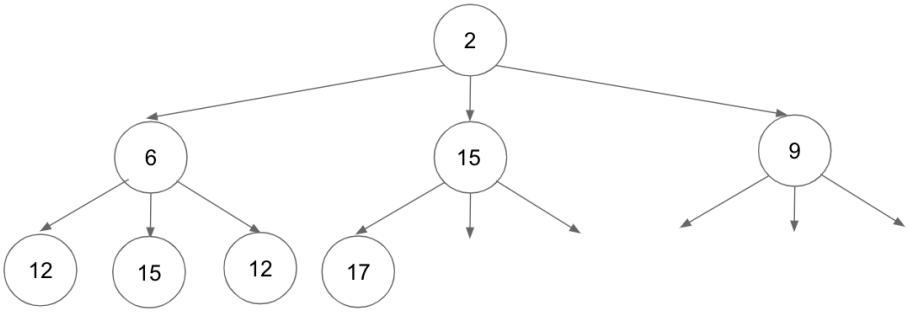
Insert the string **aa** into the **ternary max** heap denoted by the following array representation. Remember that in a ternary heap, each node has three children. Write out what the **final array** representation would look like.

To make things a little spicy, we're also going to use a **custom** `compareTo` function with our strings. A string with more characters is considered larger than one with fewer. In the event that two strings have the same number of characters, the tie can be broken by whichever string contains more copies of the letter "a".

Here is the starting array representation: `bat, ka, ba, rv, la, ty, bh`.

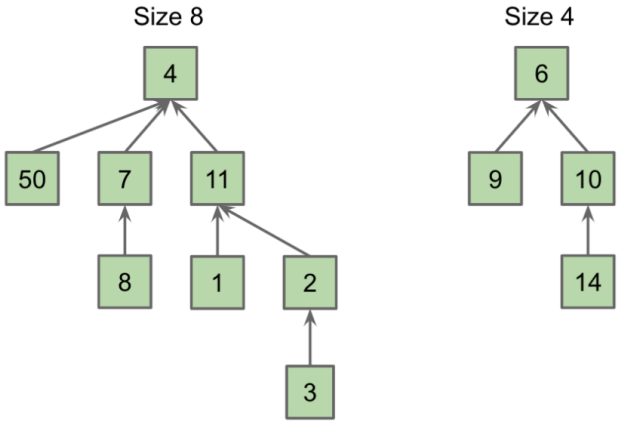
2. Heaps 2: Range

Consider the following min heap. Specify the **exact** range of values that, when inserted into this heap as the next element, would cause **exactly** one swap to occur.



3. Disjoint Sets 1: Array Representation

Fill in the correct array representation for the following disjoint sets structure given a mapping of items to their indices. Use the table below, and assume we are using the implementation from your **project**.



Items:	50	4	7	8	9	11	1	2	3	6	10	14
Index:	0	1	2	3	4	5	6	7	8	9	10	11
Value:												

4. Disjoint Sets 2: Find Set

Using the disjoint sets from **Problem 3**, perform `findSet(2)` with path compression. Give the return value (i.e. the index of the representative), draw the updated array, and draw the resulting disjoint sets.

5. Disjoint Sets 3: Union

Using the disjoint sets from **Problem 3**, instead perform `union(3, 14)`. Use both the path compression and union-by-size optimizations. Draw the resulting array.

6. Sorting

Given the following orderings of values, fill in the runtime table to sort each using the following sorting algorithms. Give a simplified, tight Theta bound for each entry in the table.

Sorted input example: `[0, 1, 2, 3, 4, 5, 6, 7]`

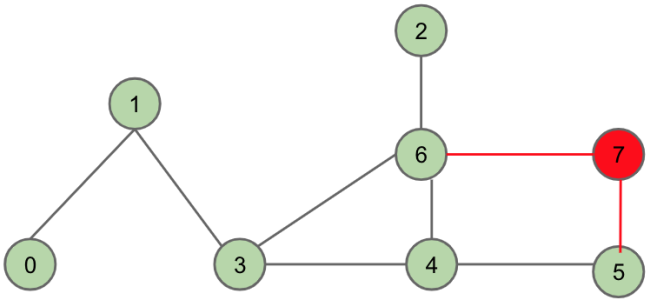
Reverse sorted input example: `[7, 6, 5, 4, 3, 2, 1, 0]`

Random input example: `[9, 12, 51, 2, 9, 12, 5]`

	Insertion Sort	Selection Sort	Merge Sort	Quick Sort	Heap Sort
Sorted input					
Reverse sorted input					
Random input (w/ duplicates)					

7. Graphs 1: Representation

Consider the graph below.



(a) Add the vertex shown in red to the adjacency list representation of this graph. State both which nodes have their entries change, and what entries are added or removed.

Original Adjacency List:

$0 \rightarrow [1]$

$1 \rightarrow [0 \rightarrow 3]$

$2 \rightarrow [6]$

$3 \rightarrow [1 \rightarrow 4 \rightarrow 6]$

$4 \rightarrow [3 \rightarrow 5 \rightarrow 6]$

$5 \rightarrow [4]$

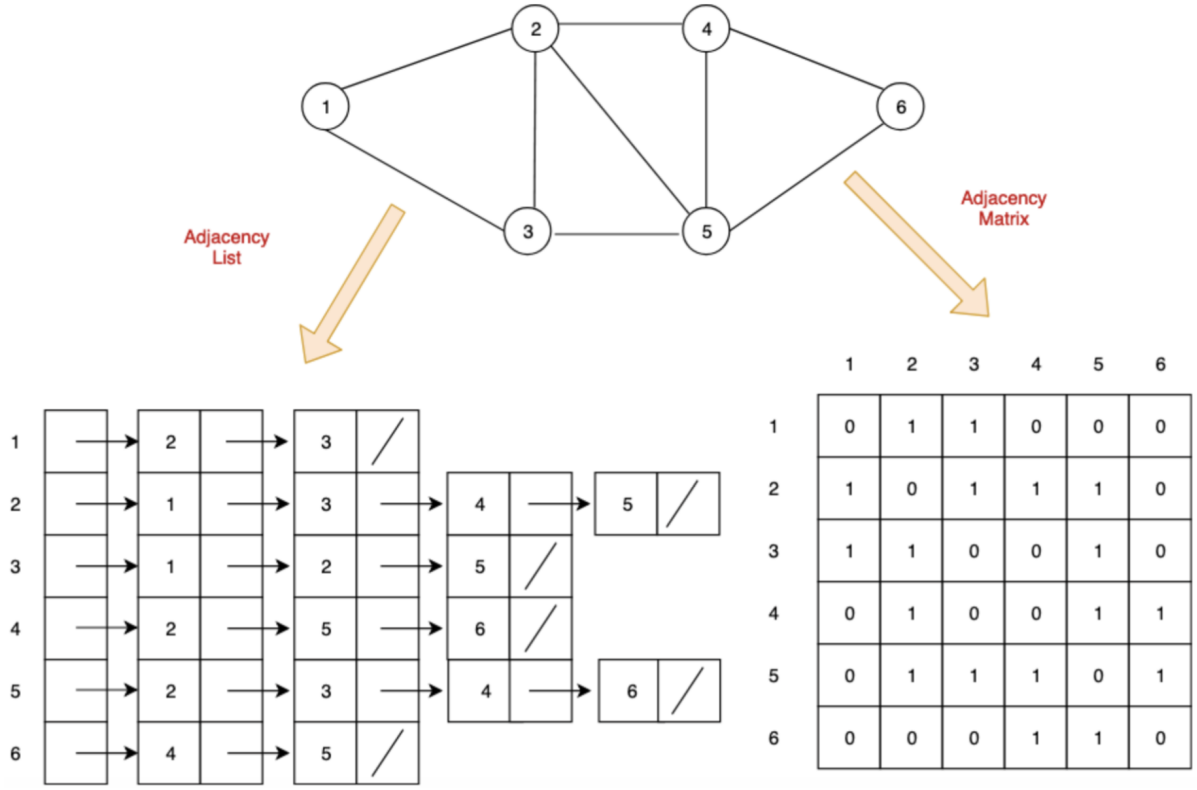
$6 \rightarrow [2 \rightarrow 3 \rightarrow 4]$

(b) Fill in the full adjacency matrix representation for this graph after the vertex shown in red has been added.

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

8. Graphs 2: DFS

Consider the following graph, which is represented as both an adjacency list and an adjacency matrix.



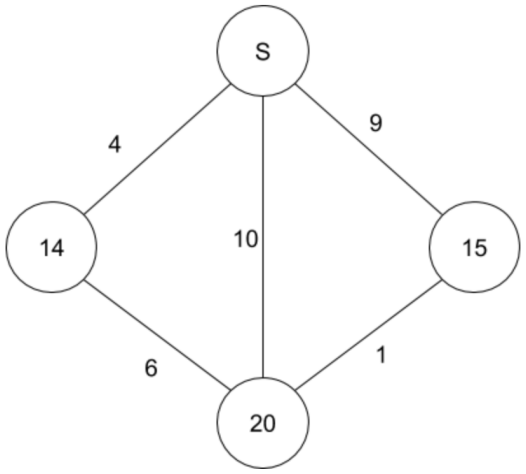
(a) Run DFS starting at Vertex 3, and give us the order of nodes visited. When choosing which neighbor to loop over and add to the stack next, loop in the order provided by the adjacency list value lists.

For example, say you have as a KV pair $[a] \rightarrow [d, b, c]$. Here, d would be added to the stack first and c last.

(b) Do the same thing as Part A, but now loop in the reverse order (c first and d last) when looping over neighbors.

9. Graphs 3: Dijkstra

Consider the following graph:



(a) Run Dijkstra's on this graph, filling in the **final** table of information about the current best distance from S to 20. If there is a tie in comparing distances (say there is a path to node A with a distance of 5 and you find another path of distance 5) choose to store the most recently found path. Think of this as adding a `from.dist + edge.weight <= to.dist` in the pseudocode instead of `from.dist + edge.weight < to.dist`.

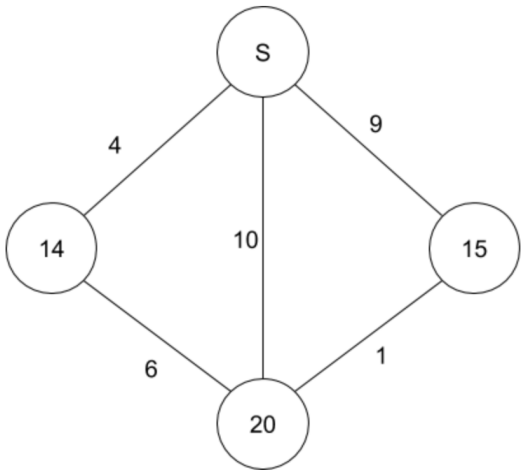
Vertex	Distance	Predecessor
S		
14		
15		
20		

(b) Do the same thing as Part A, but now do tie-breaking normally (strictly use <).

(c) Do the same thing as Part A, but for tie-breaking instead choose the path that has the fewest number of edges (go back and count or write it down if you need to).

10. Graphs 4: Prims

Using the same graph as Question 9, run Prim's algorithm. Fill out the table of information we saw in lecture, and show the table's **final** state. If there is a tie in comparing distances (say there is a path to node A with a distance of 5 and you find another path of distance 5) choose to store the most recently found path.



Vertex	Distance	Best Edge
S	-	-
14		
15		
20		