

Part I (Written component):

Due: **Friday, September 22 at 2:15 pm.**

Submission: Please hand in a paper copy of Part I at the beginning of class.

1. [4 points]:

(i) Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2, one at a time, into an initially empty binary heap.

(ii) Show the result of building a binary heap from the same input, using the linear-time `BuildHeap` algorithm instead.

Please show your work. (You do not have to show each and every step individually, but please show enough that I can follow it.)

2. [8 points]:

A *min-max heap* is a data structure that supports both `extractMin` and `extractMax` in logarithmic time. The structure property for a min-max heap is the same as for a binary heap. The min-max heap order property is that for any node X at even depth, the key (i.e., priority value) stored at X is the smallest in the subtree rooted at X , whereas for any node X at odd depth, the key stored at X is the largest in the subtree rooted at X . The root is at even depth.

(i) Draw a possible min-max heap from the elements 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. Note that the solution to this problem is not unique.

(ii) Give an algorithm to insert a new node into a min-max heap. (The description of your algorithm does not need to be formal, but please be clear and concise to ensure that I understand your algorithm. Providing pseudocode is one good way to do this.)

Extra Credit. [2 points]:

Show that the worst-case running time $T(n)$ for the `BuildHeap` algorithm is in $O(n)$. Recall that we began this problem in class. The hints I gave are repeated below.

Begin with the following sum, where the summand represents the longest possible percolation distance for each node at height h , multiplied by the number of nodes at height h .

$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} h * (\text{number of nodes at height } h) \quad (1)$$

Simplify the sum, allowing its magnitude to grow, to reach a result that is a constant multiple of n . Please show your work.

You may find these facts helpful:

(i) If n is the number of nodes in a binary tree, and n_h is the number of nodes at height h , then $n_h \leq \lceil \frac{n}{2^{h+1}} \rceil$.

(ii) For $|x| < 1$, $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$. (Consider $x = \frac{1}{2}$.)

(PLEASE TURN PAGE OVER)

Part II (Programming component):

Due: **Friday, October 6 at 2:15 pm.**

Submission: Please submit the files `DirectedGraph.java`, `BfsTest.java`, and any other files you may have written that are necessary to build and run your code, by sending them as email attachments to submit301@cs.grinnell.edu. Your submission should arrive by the due date and time. Your code should compile and run on MathLAN.

1. [8 points]:

Write a public class named `DirectedGraph` that implements an *adjacency list* data structure to store a directed graph. Your class should include the methods described below.

```
public DirectedGraph(int n)
```

This is a constructor that accepts the number of vertices in the graph. The resulting graph has n vertices and zero edges. The vertices will be numbered from 0 to $n - 1$.

```
public int inDegree(int v)
```

This method returns the in-degree of vertex v (i.e., the number of edges incident to and directed toward v). Your data structure should store the in-degree of each vertex, rather than computing it when the method is called.

```
public int outDegree(int v)
```

This method returns the out-degree of vertex v (i.e., the number of edges incident to and directed away from v). Again, you should store this information rather than computing it when the method is called.

```
public void insert(int v1, int v2, int c)
```

This method inserts the edge $(v1, v2)$ with cost c into the graph.

```
public void remove(int v1, int v2)
```

This method removes edge $(v1, v2)$ from the graph.

```
public EdgeData[] neighbors(int v)
```

This method returns an array of `EdgeData` objects (described below). Each element in the array corresponds to a neighbor of v (i.e., a vertex adjacent to v). For this assignment, an `EdgeData` object has two fields: `int endv` (the vertex number of a neighbor of v), and `int cost` (the cost of the edge from v to $endv$).

```
public String toString()
```

This method returns a string representation of the graph, which is useful for testing and debugging. You may decide on the appearance of the string, but it should be readable, and it should show the degree of each vertex and all edges incident to each vertex (including the cost of each edge).

Note: Your data structure will use linked lists to store the graph edges. You should write the code for the lists as part of this assignment, rather than using code from a previous course. (It is good practice, and it ensures that everyone has the same amount of code to write.) However, you do not need to write a full linked list data structure. You are only required to write the operations needed to implement the graph.

2. [4 points]:

a. Add the following method to your `DirectedGraph` class: `public BfsTree BFS(int source)`

This method performs a breadth-first search of the graph, starting at the vertex *source*. The method returns a `BfsTree`, which contains two arrays: `int[] distance`, and `int[] parent`. Each element i in `distance[]` gives the distance of an unweighted shortest path from vertex *source* to vertex v_i . The corresponding element in `parent[]` gives the vertex number of the predecessor of v_i along this path. If vertex v_i is not reachable from *source*, `parent[i]` should contain the value -1 , and `distance[i]` may contain any value.

b. Write a main class `BfsTest` that generates an example graph, prompts the user for two vertex numbers (*source* and *dest*), and then prints an unweighted shortest path from *source* to *dest*. If no such path exists, print a message to that effect.

3. [4 points]:

Add the following method to your `DirectedGraph` class.

```
public int[] topologicalSort()
```

This method performs a topological sort of the vertices of the graph. It returns an array of vertex numbers, sorted into a valid topological sort ordering. If no such ordering is possible, the method returns `null`.