# Homework 8

Due by **12:30pm, Friday, November 18, 2016**.

Make sure you follow all the homework policies (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/policies/hw-policy.html).

All submissions should be done via Autolab (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/autolab.html).

# Question 1 (Programming Assignment) [40 points]

### </> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

### The Problem

In this problem, we will explore minimum spanning trees.

We are given a undirected, connected graph represented by its **adjacency matrix** representation. Our goal it to find a minimum spanning tree of that graph

### Input

The input file is given as an $n \times n$ matrix where each entry $(u, v)$ represents the weight of the edge between nodes $u \in \{0, 1, \ldots, n-1\}$ and $v \in \{0, 1, \ldots, n-1\}$. If there is no edge then the weight is `-1` . Edge weights will be `0 <= w <= 50` .

```
w0,0 w0,1 w0,2 w0,3 ...  w0,n-1
w1,0 w1,1 w1,2 w1,3 ...  w1,n-1
.
.
.
wn-1,0 wn-1,1 wn-1,2 wn-1,3 ...  wn-1,n-1
```

### Output

The output is a list where every index corresponds to the node ID and the value in the index is the parent of the node. In other words, you need to output a rooted form of an MST (you can root the MST at any node).

```
[p0 p1 p2 .... pn-1]      <- Where the subscript is the node ID and p is the parent node
```

### </> Note

The root of the minimum spanning tree should have a `-1` as its parent's node ID

### </> Note

There are more than one possible MST for a given input instance so your output might not match the sample output.

## Example

**Input:**

Here's an example input of the following graph

```
-1  3 -1  2 -1 -1 -1 -1  4
 3 -1 -1 -1 -1 -1 -1  4 -1
-1 -1 -1  6 -1  1 -1  2 -1
 2 -1  6 -1  1 -1 -1 -1 -1
-1 -1 -1  1 -1 -1 -1 -1  8
-1 -1  1 -1 -1 -1  8 -1 -1
-1 -1 -1 -1 -1  8 -1 -1 -1
-1  4  2 -1 -1 -1 -1 -1 -1
 4 -1 -1 -1  8 -1 -1 -1 -1
```

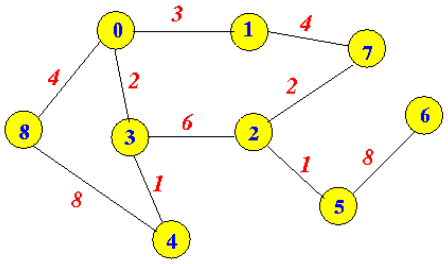
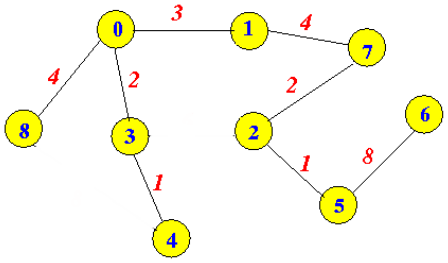



Image Source (http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/weighted.html)

**Output for the above example:**

Where node 0 is the root node.

```
[-1, 0, 7, 0, 3, 2, 5, 1, 0]
```




## Hint

One can implement Prim's algorithm (which we saw in class for the adjacency list format) in time $O(n^2)$ where $n$ in the total number of nodes. In Question 3, you will argue that this is optimal.

## ! Note

**Both the input and output parsers in each of the three languages are already written for you.**
Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file).

## ! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Java | Python | C++

Download Java Skeleton Code (HW8Java.zip)

### Directory Structure

```
                                    ├── Driver.java
                                    ├── HW8_Student.java
                                    ├── testcases/
                                    │   ├── input1.txt
                                    │   ├── input2.txt
                                    │   ├── input5.txt
                                    │   ├── output1.txt
                                    │   ├── output2.txt
                                    │   └── output5.txt
```

You are given two coding files: `Driver.java` and `HW8_Student.java`. `Driver.java` takes the input file, parses it and creates an instance of the class and prints result in output file. You only need to update the `HW8_Student.java` file. You may write your own helper methods and data structures in it.

The testcases folder has 3 input files and their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding.

### Method you need to write:

```
        /**
         * You  will fill this out.
         * We expect you to find out the minimum spanning tree. You may choose your root arbitrarily.
         * The int[] contained will represent this tree. Each value will be the parent of the node with the index value.
 5.      * For example, if output[7] = 12, then node 7 has 12 as its parent. For the root node, it should have a value o
         * -1.
         * @return a representation of the MST.
         */
        public int[] output_edges() {
10.
            return null;
        }
```

The `HW8_Student` class has 1 instance variables:

- `adj_matrix` which is a `ArrayList<ArrayList<Integer>>`, which is a 2D list which represents the Adjacency Matrix of the input graph.

Your method is expected to return an `int []` array whose indices represent node ids and whose values represent the parent node of the index node (the root of the MST should have a parent of $-1$).

### Compiling and executing from command line:

Assuming you're in the same directory level as `Driver.java`. Run `javac Driver.java` to compile.

To execute your code on `input1.txt`, run `java Driver testcases/input1.txt`. The output array will be printed to `stdout`.

### Submission

You only need to submit `HW8_Student.java` to Autolab.

## Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

# Question 2 (Exponentiation) [45 points]

### The Problem

We will consider the problem of exponentiation an integer to another. In particular, for non-negative integers $a$ and $n$, define `Power` $(a, n)$ be the number $a^n$. (For this problem assume that you can multiply two integers in $O(1)$ time.)

1. Present a naive algorithm that given non-negative integers $a$ and $n$ computes `Power` $(a, n)$ in time $O(n)$.

> **Note**
> For this part, there is no need to prove correctness of the naive algorithm.

2. Present a divide and conquer algorithm that given non-negative integers $a$ and $n$ computes `Power` $(a, n)$ in $O(\log n)$ time.

> **Important Note**
> **To get credit you must present a recursive divide and conquer algorithm and then analyze its running time by solving a recurrence relation. If you present an algorithm that is not a divide and conquer algorithm you will get a level 0 on this entire part.**

> **Hint**
> The following mathematical identity could be useful-- for any real numbers $b, c$ and $d$: $b^{c+d} = b^c \cdot b^d$.

## Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible. Also make sure that you preview your upload on Autolab to make sure it was uploaded correctly.

## Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part 1 of the problem:

1. `Algorithm details: 10` points. Nothing else is needed.

Below is grading rubric for part 2:

1. `Algorithm idea: 10` points.
2. `Algorithm details: 10` points.
3. `Proof of correctness idea: 5` points.
4. `Runtime analysis: 10` point.
5. `Note:` If the algorithm is not a recursive divide and conquer algorithm or the run time analysis does not go through the recurrence time bounds, then you get level 0 for the entire part 2.

> **! Note**
> The grading rubric above is somewhat non-standard. So please make sure you pay attention.

> **! Note**
> **If you do not have labeled and separated out proof idea, algorithm idea, algorithm details and runtime analysis you will get a zero(0) irrespective of the technical correctness of your solution.**

# Questions 3 (Optimality of Prim's algorithm) [15 points]

### The Problem

Argue that your algorithm from Question 1 is the best possible: i.e., **no** algorithm that solves the MST problem when the input graph is given in the adjacency matrix format can have a faster asymptotic running time. In other words argue that *any* algorithm that computes an MST on a graph presented in its adjacency matrix format needs time $\Omega(n^2)$.

> **Hint**
> The support page on proving adversarial lower bounds (http://www-student.cse.buffalo.edu/~atri/cse331/support/lower-bound/index.html) could be useful.

## Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible. Also make sure that you preview your upload on Autolab to make sure it was uploaded correctly.

## Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

1. `Proof idea : 5` points.
2. `Proof details : 10` points.

> **! Note**
> **If you do not have labeled and separated out proof idea and proof details, you will get a zero(0) irrespective of the technical correctness of your solution.**.