CS91T Lab Assignment 3

This homework is due at 11:59PM on Wednesday, March 22. For the written portion of this assignment, write your solution using LaTeX. Submit this homework using **github**.

This is a partnered assignment, but you may choose to work on this solo, assuming no students in the class need a partner. You should primarily be discussing this assignment with your partner. It's ok to discuss approaches at a high level. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else.

When you submit homework assignments this semester, please keep the following in mind:

- Don't forget to fill out the **homework submission poll** using Pollster.
- Don't submit a .pdf – just the .tex will do.
- I will compile your .tex file using pdflatex. It is your responsibility to make sure the LaTeX compiles.
- For the coding portion of this assignment, feel free to use Python or C++. Submit your program along with the .tex file.

The main **learning goals** of this assignment are to practice implementing randomized algorithms and to understand the relative power of Las Vegas, Monte Carlo, and deterministic algorithms.

Part 1: Randomized Algorithms implementation. In the week of February 27, you saw three algorithms: two for computing the Median of a list of numbers, and one for computing the minimum cut of a graph. In this part you will implement these algorithms.

1. Median. In a file `median.py` or `median.cpp`, you should implement three functions:

   - `isMedian`. This function takes a List of numbers L and an item x and returns True iff $x = \text{Median}(L)$
   - `recursiveMedian` (L). This function takes a List of numbers L and uses the recursive partitioning algorithm to compute $\text{Median}(L)$.
   - `samplingMedian` (L). This function takes a List of numbers L and uses the sampling algorithm to compute $\text{Median}(L)$.

   You will then write a test program that runs both algorithms on the same input. In part 2, you'll write a short summary of your results.

2. MinCut. In a file `mincut.py` or `mincut.cpp`, you should implement Karger's randomized algorithm for `MinCut`. This algorithm takes as input an undirected graph G, and returns the number of edges in the minimum cut of G. You should also write a program that takes in an undirected graph as input and prints out the number of edges in the mincut.

Implementation Requirements

- Your programs should read in the input from standard input and write output to standard output.
- Inputs to the MEDIAN problem will consist of two lines. The first line of input should contain a single integer N, which is the number of items in the list for which you'll compute the median. The second input line should contain N integers separated by whitespace. These are the integers that need to be sorted.
- Inputs to the `MinCut` problem will consist of several lines.

  The first line of input will contain two integers N and M separated by whitespace, representing an undirected graph with N vertices and M edges. The vertices will be numbered $0 \ldots N-1$.

  The next M lines will each contain two integers u, v with $0 \leq u, v < N$. This represents an edge (u, v) in the graph.
- You're free to use Python or C++ for your `MinCut` implementation, and you should use an adjacency list to represent the graph. Beyond this, you have a lot of freedom in how you represent the graph and how you process edge deletions.
  - Represent the adjacency list as a List of Lists, or a Dictionary of Dictionaries.
  - It's possible to implement the edge contraction by actually deleting edges and/or vertices, or by *emulating* this deletion somehow (e.g. by keeping a dictionary with keys corresponding to edges and values corresponding to whether the edge has been deleted.
- For `MinCut`, I encourage you to practice incremental design. First, write code to read in the graph from standard input and store it in an adjacency list. Print out this graph, and verify that you've correctly read in the graph. Second, write a function that handles a single edge contraction. Make sure this works before finally implementing Karger's algortihm.
- The directories `/home/brody/public/cs91T/median` and `/home/brody/public/cs91T/mincut` contain several input files you can use to test your program. To use these files, you'll need to pipe the content of the files to standard input, e.g. as follows:

```
python3 median.py < /home/brody/public/cs91T/median/in10k.
```

As mentioned above, you're welcome to use Python or C++. Either way, I encourage you to consult the "implementation details" post on EdSTEM for tips and tricks for using randomness, running timing experiments, etc.

Part 2: Written Problems

1. **Median Analysis.** In Part 1, you wrote a test program that runs both MEDIAN algorithms on the same input. Which algorithm is faster? Does the runtime of each implementation scale linearly with n? Does the relative performance of each algorithm depend on the inputs or the randomness, or is it always the case that one of the algorithms is faster?

   Write a short paragraph summarizing your results.

2. Suppose you have a Monte Carlo algorithm $\mathcal{A}(x)$ that computes a function $f(x)$ with error $1/3$ and worst-case runtime T. Design and analyze a randomized algorithm $\mathcal{B}$ for $f(x)$ that has error $1/k$ for some positive integer $k > 3$. Argue that your algorithm correctly computes $f(x)$ except with error probability $1/k$, and analyze the runtime of your algorithm in terms of T and k.

3. Suppose you have a Las Vegas algorithm $\mathcal{A}(x)$ that computes some function $f(x)$ and has expected runtime T. Design a Monte Carlo algorithm $\mathcal{B}(x)$ that computes $f(x)$. Your algorithm can use $\mathcal{A}$ as a subroutine. It should have error at most $1/100$ and *worst-case* runtime $O(T)$.

4. Suppose you have a Monte Carlo algorithm $\mathcal{A}(x)$ that correctly computes a function $f(x)$ with error $1/3$ and worst-case runtime T. Furthermore, when $\mathcal{A}$ fails to compute $f(x)$, it always outputs FAIL. Design and analyze a Las Vegas algorithm $\mathcal{B}$ for $f(x)$. Your algorithm should have expected runtime $O(T)$.

Part 3: Extra Challenge Problems In many of the assignments, there will be a few extra challenge problems. These problems are completely optional.

1. **(extra challenge problem).** In class you've seen two different randomized algorithms for computing the median of a list of numbers. Both have expected $O(n)$ runtime. A deterministic $O(n)$ time algorithm exists for MEDIAN.
   - Review (https://www.ics.uci.edu/ eppstein/161/960130.html) to understand the deterministic MEDIAN algorithm.
   - Implement the deterministic MEDIAN algorithm.
   - How does the runtime compare to that of the two randomzied algorithms you wrote? Does it outperform `recursiveMedian` or `samplingMedian` on any inputs? on typical inputs?

2. **(extra challenge problem).** Implement a deterministic algorithm for MINCUT. How does the runtime compare to that of your randomized algorithm?

3. **(extra challenge problem).** Try to come up with a specific computational problem $\mathcal{P}$ for which (1) there is an efficient randomized algorithm, but (2) no polynomial-time deterministic algorithm is possible. Clearly define the problem, and argue why an efficient deterministic algorithm is unlikely, but a randomized algorithm is possible.