Assignment 3

CS 577 *Union Find - Heap* Fall 2024

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Related Readings: `http://pages.cs.wisc.edu/~hasti/cs240/readings/`

Name: ______________________ Wisc id: ______________________

- **Purpose of Homework:**
  - Algorithm design and analysis, like any skill, can only be developed through consistent practice and feedback. Whether it's cooking, playing basketball, integration, gardening, interviewing, or teaching, theoretical knowledge alone is not sufficient. The comfortable feeling of "Oh, sure, I get it" after following a well-presented lecture or hearing a TA explain a homework solution is a seductive, yet dangerous trap. True understanding comes from doing the thing—by actually solving the problems yourself.
  - The homework assignments are your opportunity to practice. Lectures, textbooks, office hours, labs, and guided problem sets are designed to build intuition and provide justification for the skills we want you to develop. However, the most effective way to develop those skills is by attempting to solve the problems on your own. The process is far more important than the final solution.
  - Expect to get stuck. It's normal to have no idea where to start on some problems. That's why you have access to a textbook, lecture slides, and discussions. The journey of wrestling with the problem is an essential part of the learning process.

Scoring Guidelines

1) 10 points					
2) 10 points					
3a) 5 points	3b) 15 points				
4a) 5 points	4b) 15 points				
5a) 6 points	5b) 2 points	5c) 2 points	5d) 10 points	5e) 5 points	5f) 5 points
6a) 6 points	6b) 2 points	6c) 2 points			
7a) 15 points	7b) 10 points	7c) 15 points			

- **Note:** In Exercises [1,2], you are required to analyze various design choices for the structure. For the remaining exercises, you may assume the use of an optimal union-find structure with path compression, achieving an amortized time complexity of $O(\alpha(n))$ per operation.

To get the full score, you only need to collect 100 points. Any additional points will be calculated as a bonus. The score is calculated as:

$$\text{Score} = \left\lceil 100 \times \frac{1}{3}\left(\frac{\text{Score}_1 + \text{Score}_2}{20} + \frac{\text{Score}_3 + \text{Score}_4}{40} + \frac{\text{Score}_5 + \text{Score}_6 + \text{Score}_7}{40}\right)\right\rceil$$

Example. *A student who solves all the exercises can earn up to 184 points. - (Almost one extra assignment)* -

Homework Guidelines

- **Collaboration and Academic Integrity:**
  - You are encouraged to work together on homework problems, but you must list everyone you worked with for each problem.
  - You must write everything in your own words and properly cite every external source you use, including ideas from other students. The only sources that you are not required to cite are the official course materials (lectures, notes, homework solutions).
  - Plagiarism is strictly prohibited. Using ideas from other sources or people without citation is considered plagiarism. Copying verbatim from any source, even with citation or permission, is also considered plagiarism. Don't cheat.
- **Submission Instructions:**
  - Submit your homework solutions as PDF files on Gradescope. Submit one PDF file per numbered homework problem.
  - Gradescope will not accept other text file formats such as plain text, HTML, LaTeX source, or Microsoft Word (.doc or .docx).
  - Homework submitted as images (.png or .jpg) will not be graded.
  - Each submitted PDF file should include the following information prominently at the top of the first page: [your full name]_[course title]_[homework assignment number].pdf
- **Solution Writing:**
  - When writing an algorithm, a clear description in English is sufficient. Pseudo-code is not required.
  - Ensure that your algorithm is correct by providing a justification, and analyze the asymptotic running time of your solution. Even if your algorithm does not meet the requested time bounds, you may receive partial credit for a correct, albeit inefficient, solution.
  - Pay close attention to the instructions for each problem. Partial credit may be awarded for incomplete or partially correct answers.

Union-Find Constructions

1. Consider a union-find data structure that uses union by depth (or equivalently union by rank) that we discussed in class *without* path compression. For all integers m and n such that $m \approx 3n$, prove that there is a sequence of n `MakeSet` operations, followed by m `Union` and `Find` operations, that require $\Omega(m \log n)$ time to execute.

2. Consider the following solution for the union-find problem, called **Union-by-Weight**. Each set leader x stores the number of elements in its set in the field `weight`(x). Whenever we perform a `Union` of two sets, the leader of the smaller set becomes a new child of the leader of the larger set (breaking ties arbitrarily).

```
MakeSet(x):
parent(x) ← x
weight(x) ← 1
```

```
Find(x):
while x ≠ parent(x)
  x ← parent(x)
return x
```

```
Union(x, y):
x̄ ← Find(x)
ȳ ← Find(y)
if weight(x̄) > weight(ȳ)
  parent(ȳ) ← x̄
  weight(x̄) ← weight(x̄) + weight(ȳ)
else
  parent(x̄) ← ȳ
  weight(ȳ) ← weight(ȳ) + weight(x̄)
```

Prove that if we use **Union-by-Weight**, the worst-case running time of `Find(x)` is $O(\log n)$, where n is the cardinality of the set containing x.

Percolation Theory

Percolation theory is a branch of mathematics that studies the behavior of connected clusters in random graphs. It has important applications in fields like material science, where it models phenomena such as fluid flow through porous media, electrical conductivity in composites, and the spread of diseases or fires.

To illustrate the concept, imagine a sponge-like material where water is poured at the top. The key question is whether the water can percolate through, moving from the top to the bottom. This question has significant implications for understanding various natural and engineered systems. Let's see how we model this problem:

In a typical percolation problem, we consider an $n \times n$ grid where each cell is either blocked or open. Initially, all cells are blocked, preventing any flow. Over time, cells randomly become open, simulating processes like chemical reactions. Percolation occurs when there is a continuous path of open cells from one side of the grid to the other, similar to checking if an electrical current can pass through a material based on its connectivity.

Scientists often calculate the probability of percolation after a certain number d of cells have opened, as well as the size of the largest connected component of open cells. Since the mathematical analysis is complex, these probabilities are usually estimated through computational simulations. In the following exercises, your task is to assist in implementing these simulations efficiently.

3. In this exercise, we will consider the case of a 1d data structure. Suppose we want to maintain an array `X`$[1..n]$ of bits, all initially set to zero, subject to the following operations:

   - `Lookup(i)`: Given an index i, return `X`$[i]$.
   - `Blacken(i)`: Given an index $i < n$, set `X`$[i] \leftarrow 1$.
   - `NextWhite(i)`: Given an index i, return the smallest index $j \geq i$ such that `X`$[j] = 0$.
     (Since `X`$[n]$ is never modified, such an index always exists.)

   (a) Describe a simple data-structure to implement `Lookup` and `Blacken` in $\Theta(1)$ time, while `NextWhite` requires $\Theta(n)$ time.

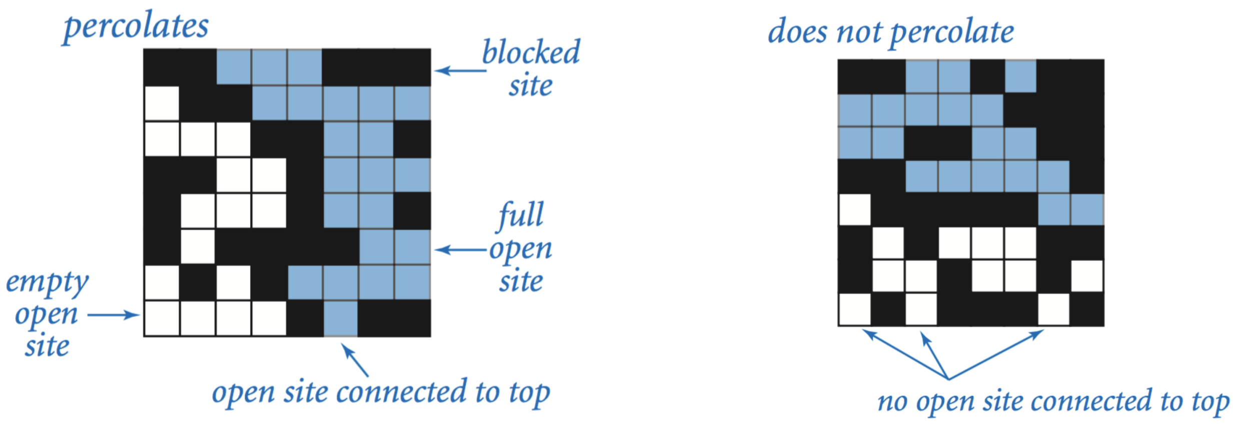
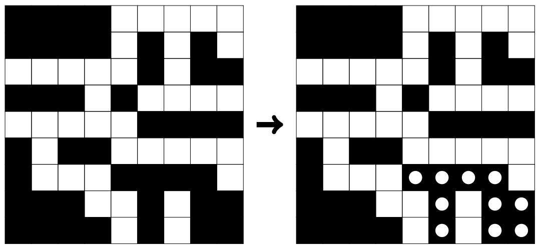


Figure 1: Illustration of a percolating system where a fluid moves through a porous medium. The system is said to percolate when there is a continuous path of open cells from one side to the other.

(b) Describe a data structure that supports `Lookup` in $O(1)$ worst-case time, `Blacken` in $O(1)$, and the amortized time for `NextWhite` is $O(\alpha(n))$.

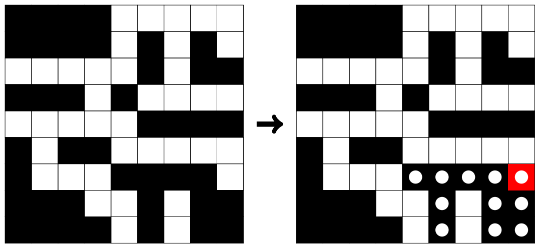
4. In this exercise, we will explore the 2D case and examine the differences between static and dynamic problems.



10

(a) Describe and analyze a linear algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1 \ldots n, 1 \ldots n]$.

(b) Design and analyze an algorithm `Blacken(i, j)` that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap (break the ties arbitrarily). For full credit, the amortized running time of your algorithm (starting with an all-white bitmap) must be as small as possible.



11

5. *During the last time I taught the course "Introduction to Algorithms," at Greece with 1000 students, I encountered a rather unexpected challenge just moments before class. I had meticulously arranged all the exams in alphabetical order, ready to present them to the students. However, disaster struck—only 30*

minutes before the presentation, the stack of exams slipped from my hands and scattered all over the floor. In a rush, I gathered the exams and noticed that while they were no longer perfectly ordered, each exam was still relatively close to its original position. Specifically, each exam was no more than k positions away from where it should have been in the sorted stack. I decided to call this situation k-confused, where each element is at most k positions away from its final sorted position. With the clock ticking, I needed to quickly sort the n exams that were now k-confused. The challenge was to find a fast and efficient way to restore the exams to their correct order.

I look forward to seeing how you would have solved this problem under similar pressure!

(a) Develop an efficient comparative algorithm to sort a $k = \Theta(1)$-confused array of n elements.

(b) Justify the correctness of your algorithm.

(c) Analyze the computational complexity of your solution.

(d) Develop an efficient comparative algorithm to sort a k-confused array of n elements when $\omega(1) = k = o(n)$.

(e) Justify the correctness of your algorithm.

(f) Analyze the computational complexity of your solution.

6. *During my doctoral research, I encountered a challenge with the precision of my measurements. I was working with sensitive instruments to capture electrical readings, but each reading was slightly off due to small electrical noise. These readings, recorded in an array A, should have followed a simple pattern where each $A[i] = i + b_i$, with b_i representing the noise. Although the noise was small—within the range $[-10, 10]$—it caused the data to be somewhat scrambled. To analyze my data using `matplotlib.pyplot`, I needed to sort these measurements accurately and efficiently in linear time due to their enormous size, despite the noise.*

   (a) Given an array A of length n where $A[i] = i + b_i$ and b_i is a real number in the range $[-10, 10]$, devise an algorithm that sorts A in $O(n)$ time, assuming that comparing two real numbers takes $O(1)$ time.

   (b) Justify the correctness of your algorithm.

   (c) Analyze the computational complexity of your solution.

To Heap or not to heap.

7. **Minimum Length Interval Covering All Arrays**

   (a) Given two sorted arrays of integers $A_1[1 \dots n_1]$ and $A_2[1 \dots n_2]$, we want to compute positions i_1 in A_1 and i_2 in A_2 such that the difference $|A_1[i_1] - A_2[i_2]|$ is minimized (in other words, we want to compute a minimum length interval containing at least one element from each array). Describe an algorithm with linear running time for this problem. Briefly justify the correctness and the computational complexity of your algorithm.

   (b) Given m sorted arrays of integers $A_1[1 \dots n_1]$, $A_2[1 \dots n_2]$, ..., $A_m[1 \dots n_m]$, similar to part (a), we want to compute positions i_1 in A_1, i_2 in A_2, ..., i_m in A_m such that the difference is minimized:

   $$\max\{A_1[i_1], \dots, A_m[i_m]\} - \min\{A_1[i_1], \dots, A_m[i_m]\}$$

   Describe an algorithm with running time $O(mN)$ for this problem, where $N = \sum_{k=1}^{m} n_k$. Briefly justify the correctness and the computational complexity of your algorithm.

   (c) What is the operation that determines the running time of the algorithm in part (b)? Explain how (and by how much) its implementation can be improved. Based on this, optimize the running time of the algorithm.