Assignment 4

CS 577 *Divide & Conquer - Sorting - Binary Search* 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

1a) 1 point	1b) 1 point	1c) 1 point	1d) 1 point	1e) 1 point	1f) 5 points	1g) 5 points	1h) 5 points
2a) 5 points	2b) 5 points						
3a) 2.5 points	3b) 2.5 points	3c) 2.5 points	3d) 2.5 points				
4a) 5 points	4b) 5 points	4c) 5 points					
5a) 10 points	5b) 5 points						
6a) 15 points	6b) 10 points						
7) 10 points							
8a) 10 points	8b) 5 points	8c) 2.5 points	8d) 2.5 points	8e) 2.5 points	8f) 2.5 points	8g) 2.5 points	8h) 2.5 points
9a) 2.5 points	9b) 2.5 points						
10a) 5 points	10b) 5 points	10c) 10 points	10d) 5 points				
11a) 2.5 points	11b) 2.5 points						
12a) 2 points	12b) 2 points	12c) 2 points	12d) 2 points	12e) 2 points	12f) 5 points		
13a) 10 points	13b) 5 points	13c) 5 points					
14a) 2.5 points	14b) 7.5 points	14c) 7.5 points	14d) 12.5 points				

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:

$$\text{Score} = \left\lceil 100 \times \frac{1}{5}\left(\frac{\sum_{k=1}^{3}\text{Score}_k}{20} + \frac{\sum_{k=4}^{7}\text{Score}_k}{35} + \frac{\sum_{k=8}^{10}\text{Score}_k}{25} + \frac{\sum_{k=11}^{12}\text{Score}_k}{30} + \frac{\sum_{k=13}^{14}\text{Score}_k}{25}\right)\right\rceil$$

Example. *A student who solves all the exercises can earn up to 179 points. (Almost one extra assignment)*

Don't solve all of them!!! Choose strategically!!!

To get 100 points, you don't need more than 10 pages actually.

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.

Recurrence Relations

1. Compute the asymptotic order $\Theta()$ of the solutions to the following recurrence relations. For all of these, assume that $T(1) = \Theta(1)$ and $T(n)$ is a non-decreasing positive function.
   (a) $T(n) = 3T(n/4) + n \log^5 n$
   (b) $T(n) = 4T(n/4) + n \log^5 n$
   (c) $T(n) = 5T(n/4) + n \log^5 n$
   (d) $T(n) = 2T(n/3) + \frac{n}{\log^5 n}$
   (e) $T(n) = 2T(n/7) + 4T(n/9) + n$
   (f) $T(n) = T(n-1) + \log n$
   (g) $T(n) = 2T(\sqrt{n}) + \Theta(\log n)$
   (h) $T(n) = T(n/4) + \sqrt{n}$

2. *Erickson, Jeff. Algorithms (p.49, q. 6).* Use recursion trees to solve each of the following recurrences.
   (a) $C(n) = 2C(n/4) + n^2$; $C(1) = 1$.
   (b) $E(n) = 3E(n/3) + n$; $E(1) = 1$.

3. Use recursion trees or unrolling to solve each of the following recurrences. Make sure to show your work, and do NOT use the master theorem.
   (a) Asymptotically solve the following recurrence for $A(n)$ for $n \geq 1$.

$$A(n) = A(n/6) + 1 \qquad \text{with base case} \qquad A(1) = 1$$

   (b) Asymptotically solve the following recurrence for $B(n)$ for $n \geq 1$.

$$B(n) = B(n/6) + n \qquad \text{with base case} \qquad B(1) = 1$$

   (c) Asymptotically solve the following recurrence for $C(n)$ for $n \geq 0$.

$$C(n) = C(n/6) + C(3n/5) + n \qquad \text{with base case} \qquad C(0) = 0$$

   (d) Let $d > 3$ be some arbitrary constant. Then solve the following recurrence for $D(x)$ where $x \geq 0$.

$$D(x) = D\left(\frac{x}{d}\right) + D\left(\frac{(d-2)x}{d}\right) + x \qquad \text{with base case} \qquad D(0) = 0$$

Binary Search in the Enchanted Forest of Answers

4. *Kleinberg, Jon. Algorithm Design (p. 246, q. 1).* You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values—so there are $2n$ values total—and you may assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we will define here to be the nth smallest value.

   However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the kth smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.
   (a) Give an algorithm that finds the median value using at most $O(\log n)$ queries.
   (b) Give a recurrence for the runtime of your algorithm in part (a), and give an asymptotic solution to this recurrence. Show your work and do not use the master theorem.
   (c) Prove correctness of your algorithm in part (a).

5. *Before the invention of the printing press, copying books was a very painstaking process. Every page had to be meticulously transcribed by hand by specialized individuals. In the magical world, this task often fell to the goblins of Gringotts, who were known not just for guarding treasures but also for their precise and methodical work in replicating ancient and powerful spellbooks. In this tale, the librarian of Hogwarts has a stack of* $\boldsymbol{N}$ *magical books that need to be copied. To assist with this, she has enlisted the help of* $\boldsymbol{K}$ *skilled goblins. Each book contains a different number of pages filled with complex spells, potions, and enchantments. The librarian knows the number of pages in each book and must allocate the books among the goblins in such a way that the maximum number of pages copied by any single goblin is minimized. The challenge is that each goblin can only take on a consecutive set of books from the stack, and they cannot skip between books. The goal is to find the optimal way to distribute the books so that no goblin is overburdened with too many pages.*

   - **Example** If there are 5 books with page counts 10, 20, 40, 10, and 50, and there are 2 goblins, the best solution is to assign the first three books to the first goblin and the remaining two books to the second goblin. This way, the maximum number of pages copied by any single goblin is 70 (from the first goblin who copies 10 + 20 + 40 pages).
   - **Input** The input is the number of books **N** and the number of goblins **K**. Then, it will give you **N** positive integers representing the number of pages in each book.
   - **Output** Compute the solution to determine the minimum possible maximum number of pages that any goblin would need to copy.

   (a) Describe your algorithm and justify the complexity of your algorithm

   (b) Give a code sketch in your favorite programming language.

6. *In the enchanted land of Hogwarts, deep within the magical chambers, lies the Chocolate Factory run by none other than the industrious Babbity, the House-Elf chocolatier. The production of enchanted chocolate is a delicate and time-consuming process, requiring the precision and skill that only Babbity can provide. In the mystical Factory, there is a massive chocolate vat surrounded by* $\boldsymbol{N}$ *chocolate sources, each pouring into the vat starting at a magical moment* s_i *and stopping at another moment* f_i*. The magical sources each pour chocolate into the vat at a rate of one liter per magical unit of time. The production process can only commence when the vat contains at least* $\boldsymbol{K}$ *liters of chocolate, which is vital for the enchantment to take effect. However, Babbity has noticed that the closer the volume of chocolate in the vat is to exactly* $\boldsymbol{K}$ *liters when the process begins, the higher the quality of the magical chocolate produced. The problem, though, is that Babbity loves taking a nap between shifts and doesn't know precisely when to set the magical alarm to start the process at just the right time.*

   (a) Your task is to write a spell, er, program that computes the earliest possible moment when the production can start, ensuring the vat has at least **K** liters of chocolate.
      - **Explanation**: Assume that the clock is at moment t, such that $t \in [s_1, f_1]$, then source #1 has output $t - s_1 + 1$ liters. Each source contributes its share of chocolate only during its active time window.
      - **Input**: The input consists of the number of sources **N** and the required volume **K**. Following this, there will be **N** lines, each containing the start and stop times for each source $[\text{start}_i, \text{stop}_i]$ for $i \in [1, \mathbf{N}]$.
      - **Output**: The output should be the earliest time t at which the vat contains at least **K** liters of chocolate. If it is not possible to reach **K** liters, the output should indicate that as well.

   (b) *Assume that the input is provided in a more organized way, where there is a list of events*

   $$[(t_1, \{Id_1\}, \texttt{On/Off}), (t_2, \{Id_2\}, \texttt{On/Off}), \ldots, (t_{2N}, \{Id_{2N}\}, \texttt{On/Off})]$$

   where Id is the identifier of the source, and "`On`*" or "*`Off`*" indicates whether the source is starting or stopping at that moment. The events are sorted in ascending order of time* $t_1 < t_2 < \cdots < t_{2N}$*. Can you find a more efficient algorithm to solve this problem, given that the input is sorted by time?*

7. *In the enchanted world of Hogwarts, every year during the Halloween feast, the Great Hall is filled with the aroma of delicious Pumpkin Juice. This year, however, something extraordinary has happened. Professor Snape has brewed a batch of Pumpkin Juice infused with a rare potion that enhances its magical properties. The Pumpkin Juice is distributed among **N** goblets, each containing a different level of magical potency represented by **potencies[]**. Snape, using a counter-charm, has halted the levitation spell Wigardium Leviosa, forcing the students to stand in line. Harry, ever the considerate wizard, wants to ensure that no group is left with a meager share of the magic. The task for the students is to divide the queue of goblets among **K** groups such that each group receives consecutive goblets, and every goblet is assigned to exactly one group. The challenge is to maximize the minimum magical potency among all the **K** groups, ensuring that even the group with the least potent Pumpkin Juice receives a fair share. The goal of this assignment is to find the optimal way to distribute the magical Pumpkin Juice so that the group with the least potent juice receives an adequate amount of magic.*
   Example:

   - **Input**: N = 4, K = 2, potencies = 3, 1, 6, 4
     Output: 4
     Explanation: If we assign the first two goblets 3, 1 to the first group, and the last two goblets 6, 4 to the second group, the magical potency for the first group will be 4, and for the second group, it will be 10. The minimum potency is 4.

   Note: A group may receive no Pumpkin Juice at all, which means they might end up with 0 magical potency. *[Hint: Solve the problem using as black-box exercise* 5*]*

Sorting: To Infinity and Beyond

8. *Your friend, an aspiring programmer, recently started wondering what algorithms might have been known at the early stages of computer science. Back then, algorithms like SelectionSort or InsertionSort, with their $\Theta(n^2)$ time complexity, were probably the only sorting methods available. Your friend is curious if, with just such basic sorting tools (which we'll call* SlowSort*), it's possible to devise a more efficient algorithm using the Divide and Conquer strategy, without the help of more advanced techniques like QuickSort, MergeSort, AVL Trees, or Heaps. In this exercise, we will explore this question and use the Divide and Conquer technique to achieve progressively better sorting algorithms, step by step.*
   Story Inspiration: This was a conversation that your teacher once overheard during his undergraduate years, one late evening at the university dining hall, as fellow students were debating the origins of algorithmic thought.

   (a) **Merging K-Sorted Arrays:** Explain how you can merge K sorted arrays, each of size M, in $O(K^2 \cdot M)$ time.

   (b) **Using a Basic SlowSort Algorithm:** Suppose you have a black-box sorting tool called SlowSort(A) that sorts n elements with a time complexity of $O(n^2)$. If we divide the array into K groups of exactly M elements (and pad the last group with dummy elements if needed), describe a Divide and Conquer algorithm with an overall complexity of $O(n\sqrt{n})$. **Hint:** Use the Fermat criterion (first-order derivative) to show that the best choice for K and M occurs when $M = O(\sqrt{n})$.

   (c) **Improving the Sorting Algorithm:** Now, assume we have a sorting algorithm NotSoSlowSort(A) with a time complexity of $O(n^{3/2})$. Show how to design a Divide and Conquer algorithm with overall complexity $O(n^{4/3})$. What is the best choice for K and M in this case?

   (d) **Further Optimization:** Using a NotSoSlow^{2}Sort(A) algorithm with complexity $O(n^{4/3})$, demonstrate a Divide and Conquer algorithm with overall complexity $O(n^{5/4})$. What are the best choices for K and M?

(e) **General Case with Parameter α:** Suppose you have a black-box sorting tool, NotSoSlow*Sort(A), with time complexity $O(n^{1+\alpha})$, where $1 + \alpha \in [1, 2]$. Show how to use a Divide and Conquer approach to obtain an algorithm with complexity $O(n^{2-\frac{1}{\alpha+1}})$. What is the best choice for K and M in this case?

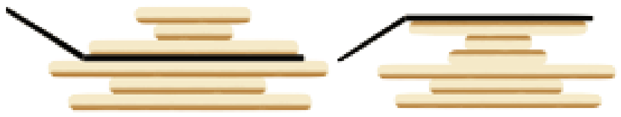
(f) **Achieving $O(n^{1+1/k})$:** How can you construct a sorting algorithm with complexity $O(n^{1+1/k})$ for every natural number k using a Divide and Conquer approach? Provide a justification for your answer.

(g) **Almost Linear Sorting:** For any $\epsilon \in (0, 1)$, design a sorting algorithm, AlmostLinearSort(A), with complexity $O(n^{1+\epsilon})$. Justify your approach.

(h) **Can We Do Better than $\Omega(n \log n)$?:** Can we construct sorting algorithms faster than the well-known $\Omega(n \log n)$ lower bound using the aforementioned approach? Is it possible to create a linear time sorting algorithm? Provide a detailed explanation.

Prefix Reverse Sorting or Pancakes Problem

9. You are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over, as shown in the figure below. *This is the famous* Pancake Sorting Problem*, which was popularized in a paper written by William (Bill) Gates, who later became the co-founder of Microsoft. This problem became well-known because Gates worked on this during his undergraduate studies at Harvard, and it later became a subject of one of his first published papers:* Bounds for Sorting by Prefix Reversal. *Gates' advisor, Christos Papadimitriou, commented on Gates' work on the problem. You can find more details on this story at the following link:* *https://www.reddit.com/r/SmartestExistive/comments/zstxqc/christos_papadimitriou_on_student_bill_gates_and/*.



(a) Describe an algorithm to sort an arbitrary stack of n pancakes using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

(b) Now suppose one side of each pancake is burned. Describe an algorithm to sort an arbitrary stack of n pancakes, so that the burned side of every pancake is facing down, using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

Sorting: Upper & Lower Bounds

10. (a) Suppose we have an array of positive integers $A[1 \dots n]$, and let M be the maximum element in A. Show that A can be sorted in $O(n + M)$ time. If $M = O(n)$, the sorting time is linear. Why does the lower bound of $\Omega(n \log n)$ not apply to this case? Provide a lower bound for the time complexity of the worst-case execution for any comparison-based sorting algorithm in this case, and investigate whether the given algorithm is optimal when $M = \Theta(n)$. *[Hint: Study CountingSort Algorithm]*

(b) Suppose we have an array of integers $A[1 \dots n]$ with multiple occurrences of elements. Specifically, assume that the number of distinct elements in A is poly-logarithmic (i.e., $O(\log^d n)$ for some constant $d \geq 1$). Formulate a comparison-based algorithm that sorts the array A in $O(n \log \log n)$ time. Why does the lower bound of $\Omega(n \log n)$ not apply in this case? *[Hint: Study AVL-TreeSort]*

(c) Prove the lower bound of $\Omega(n \log \log n)$ time.

(d) Let $M = n^d - 1$, where d is a positive integer constant. Design an algorithm that sorts the array A in linear time. *[Hint:] You can assume that the elements of A are represented in a radix-n system and extend the algorithm from part (a). It may help to explore how the (non-comparative) Radix Sort algorithm operates.*

Adversary Argument: Searching with Constraints

11. *Consider a game where Player 1 tries to guess a secret element x from a set of n possible candidates. Player 2, acting as an adversary, does not choose x at the start of the game but instead maintains a set S containing all the potential values for x. Player 2's goal is to delay the discovery of x as long as possible, while Player 1 aims to find x by asking a series of yes/no questions. Player 2 answers these questions strategically to keep the set S as large as possible for as long as possible.*

(a) Explain the strategy that Player 2 should follow to maximize the size of S. Show that if Player 1 asks k queries, successively reducing the initial set S_0, S_1, ..., S_k into smaller subsets after each query, where S_0 contains n elements and S_k contains just one element, Player 2 can ensure that $|S_{i+1}| \geq |S_i|/2$ for $1 \leq i \leq k - 1$.

(b) Prove that the minimum number of queries required for Player 1 to guarantee finding x is $\lceil \log_2 n \rceil$.

12. Using the previous exercise, we will demonstrate that an algorithm to compute both the minimum and maximum elements of an array of n elements in parallel requires at least $\frac{3n}{2} - 1$ comparisons. Let's assume we have two players: Player 1 tries to guess the minimum and maximum elements x_{low}, x_{high} by making comparison queries. Player 2, however, tries to delay Player 1 as much as possible. Player 1 cannot compare the elements directly; they can only ask questions to Player 2.

(a) To achieve this, we define 4 sets: (i) Set A: Elements that could still be either the maximum or minimum. (ii) Set L: Elements that could still be only the minimum. (iii) Set H: Elements that could still be only the maximum. (iv) Set R: Elements that cannot be either the maximum or the minimum. Initially, the sets are $|A| = n$, $|L| = 0$, $|H| = 0$, and $|R| = 0$. The goal is to have $|A| = 0$, $|L| = 1$, $|H| = 1$, and $|R| = n - 2$. Let's assume Player 2's strategy is to delay the increase of R as much as possible. For every comparison asked by Player 1, what should Player 2 answer based on this strategy?

(b) Explain why the comparison $\text{elem}_1 \in A$ and $\text{elem}_2 \in A$ is more beneficial than comparisons like $\text{elem}_1 \in A$ and $\text{elem}_2 \in R$, $\text{elem}_1 \in A$ and $\text{elem}_2 \in H$, or $\text{elem}_1 \in A$ and $\text{elem}_2 \in L$ in decreasing size A.

(c) Prove that each element follows the path $A \to \{L, H\} \to R$.

(d) Based on the previous questions, what are the only useful comparisons to increase the size of R?

(e) Prove that to empty set A and ensure that L and H contain exactly one element, Player 1 has to query $\frac{3n}{2} - 2$ comparisons.

(f) Describe a simple divide and conquer algorithm to find the minimum and maximum of an array simultaneously with at most $\frac{3n}{2} - 2$ comparisons.

Inversions & Beyond

13. *Kleinberg, Jon. Algorithm Design (p. 246, q. 2).* Recall the problem of finding the number of inversions. As in the text, we are given a sequence of n numbers $a_1, ..., a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$.

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, this measure is very sensitive. Let's call a pair a *significant inversion* if $i < j$ and $a_i > 2a_j$.

(a) Give an $O(n \log n)$ algorithm to count the number of significant inversions between two orderings.

(b) Give a recurrence for the runtime of your algorithm in part (a), and give an asymptotic solution to this recurrence. Show your work and do not use the master theorem.

(c) Prove correctness of your algorithm in part (a).

14. *In the magical world of Madison, the brave squirrels of Squirrel Hill live in a row of* n *acorn storage units, all lined up along a mystical path in the enchanted Arboretum. Each storage unit is built with acorns and magical fortifications, and some have more acorns than others, making them stronger. One day, a mischievous wind spirit arrives in Madison, known for its ability to topple acorn storage units with a powerful gust. The wind blows from west to east, and whenever it encounters a storage unit containing* b *acorns, that unit collapses, along with all the storage units to its east that contain strictly fewer acorns. For every storage unit in Squirrel Hill, the wind spirit wants to know its "fallout," i.e., the number of storage units that would collapse if it blew on it from the west.*

(a) Suppose $n = 10$ and the number of acorns in each storage unit from west to east is

$$[34, 57, 70, 19, 48, 2, 94, 7, 63, 75].$$

Compute for this instance the fallout for every storage unit in Squirrel Hill.

(b) A storage unit in Squirrel Hill is special if it either (1) has no easterly neighbor or (2) its adjacent neighbor to the east contains at least as many acorns as it does. Given an array containing the number of acorns in each storage unit, describe an $O(n)$-time algorithm to return the fallout for every storage unit in Squirrel Hill when all but one unit is special.

(c) Prove the correctness of your algorithm

(d) Given an array containing the number of acorns in each storage unit, describe an $O(n \log n)$-time algorithm to return the fallout for every storage unit in Squirrel Hill.