COMS 3110: Homework 2
Due: October 10^{th}, 11:59pm
Total Points: 50

Late submission policy. An assignment that is submitted one day late will get a penalty of 10%. An assignment that is submitted two days late will get a penalty of 20%. Assignments submitted after two days will not be graded and will get no points. For example, if an assignment is due on Friday by midnight, a submission on Saturday will be penalized by 10%, and a submission on Sunday will be penalized by 20%. A submission on Monday will get no points.

Submission format. Homework solutions must be **handwritten**. You may either write directly on a tablet or write on paper and scan your work. In both cases, the final submission must be compiled into a **single PDF file**. Typed solutions will only be accepted with prior, explicit permission from the instructor. We reserve the right **NOT** to grade homework that does not follow these requirements. Name your submission file: `<Your-net-id>-3110-hw2.pdf`. For instance, if your netid is `asterix`, then your submission file will be named `asterix-3110-hw2.pdf`. Each student must hand in their own assignment. If you discussed the homework or solutions with others, a list of collaborators must be included with each submission. Each of the collaborators has to write the solutions in their own words (copies are not allowed). Also, recall the AI policy:

- You may **not use AI tools to generate complete solutions** to assignments, coding problems, or any assessed work that is intended to measure your independent knowledge and skills.
- Whenever you use AI tools to assist in your assignment, you must **include a paragraph at the end of the assignment explaining what you used the AI for and include all the prompts you used to get the results. Failure to do so is in violation of the academic honesty policies**.

General Requirements

- When proofs are required, do your best to make them both clear and rigorous. Even when proofs are not required, you should justify your answers and explain your work.
- When asked to present a construction, you should show the correctness of the construction.

Expectations for Runtime Analysis in Algorithm Problems When a problem explicitly asks for one or more of the following components, these are the expectations for a correct solution.

- **Provide runtime:** You are only required to state the final runtime of the algorithm, and full credit will be awarded if your answer is correct. However, if your answer is incorrect, no partial credit will be given unless you also provide a derivation.
- **Provide runtime and derivation:** Show step-by-step reasoning, including the number of iterations of loops and the runtime of each line of pseudocode, leading to the total runtime.
- **Provide exact number of iterations:** Specify the precise number of times each loop executes for the given input size.
- **Provide upper bound on the number of iterations:** State a guaranteed maximum number of iterations for each loop; this may be larger than the exact count.
- **Provide asymptotic bound on the number of iterations:** Describe the growth rate of number of iterations using big-O notation.

Some Useful Equalities

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $2^{\log_2 n} = n$, $a^{\log_b n} = n^{\log_b a}$, $n^{n/2} \leq n! \leq n^n$, $\log x^a = a \log x$
- $\log(a \times b) = \log a + \log b$, $\log(a/b) = \log a - \log b$
- $a + ar + ar^2 + ... + ar^{n-1} = \frac{a(r^n-1)}{r-1}$
- $1 + \frac{1}{2} + \frac{1}{2^2} + ... + \frac{1}{2^n} = 2(1 - \frac{1}{2^{n+1}})$
- $1 + 2 + 4 + ... + 2^n = 2^{n+1} - 1$

1. **(10 pts)** Given a dynamic array `A` as input, our goal is to construct a heap `H`, which is also stored as a dynamic array.

   Your task is to analyze the time complexity of two different algorithms for building `H`. In each algorithm, the variable `n` represents the number of elements in `A`, i.e., `n = A.length`.

   Note: For both parts (a) and (b), you need to explicitly write down the summation that represents the total runtime of each loop. You may cite the result from class for the final evaluated value of the summation.

   (a) **(5 pts)** Determine the runtime of `BuildHeap1` below and provide the derivation of the runtime. For each line in the pseudocode, clearly specify its **asymptotic (Big-O) runtime bound**. Then, express the **total runtime of the loop** as a summation that accounts for the runtime of each iteration, before evaluating the summation to determine the overall time complexity.

```
BuildHeap1(A){
  Initialize empty array H
  for i = 1 to n {
    H[i] = A[i]
    Heapify-Up(H, i)
  }
}
```

(b) **(5 pts)** Perform the same runtime analysis as in part (a) for `BuildHeap2` below. Clearly specify the **asymptotic (Big-O) runtime bound** for each line of the pseudocode and express the **total runtime of each loop** as a summation that accounts for the runtime of each iteration, before evaluating the summation to determine the overall time complexity.

```
BuildHeap2(A) {
  Initialize empty array H
  for i = 1 to n {
    H[i] = A[i]
  }
  for i = floor(n/2) down to 1 {
    Heapify-Down(H, i)
  }
}
```

2. **(20 pts)** You are given a dynamic array A of n integers. The unique elements of A are those that appear only once in A.

   (a) **(10 pts)** Design an efficient algorithm to return a new dynamic array that contains **all and only the unique elements of** A.

   Your score will be evaluated based on the efficiency of your algorithm. More efficient algorithms will receive higher scores.

   Provide the **pseudocode** for your algorithm as well as its **runtime complexity**. Explicitly state whether the runtime is **expected** or **worst-case**.

(b) **(10 pts)** Now suppose you are also given an integer k, along with the same dynamic array A of n integers.

Design an efficient algorithm to return a dynamic array that contains the k **smallest unique elements** of A. If fewer than k unique elements exist, return all of them.

Your score will be evaluated based on the efficiency of your algorithm. More efficient algorithms will receive higher scores.

Provide the **pseudocode** for your algorithm as well as its **runtime complexity**. Explicitly state whether the runtime is **expected** or **worst-case**.

3. **(10 pts)** Given an array A of n distinct integers and a target integer k, write an efficient algorithm to determine whether there exist 2 distinct elements x, y such that $|x - y| = k$. Your algorithm should have **expected** runtime of $O(n)$ or better. Provide the **expected** runtime of your algorithm.

4. **(10 pts)** Given **sorted** arrays `L[1]`, `L[2]`, ..., `L[K]`, where $K \geq 1$. Each array `L[i]`, `i` $\in \{1, \ldots, K\}$ contains $N \geq 1$ elements. Write an **efficient** algorithm to merge `L[1]`, `L[2]`, ..., `L[K]` into a single sorted array. Give the runtime of your algorithm **as well as its derivation**.

   Your score will be evaluated based on the efficiency of your algorithm. More efficient algorithms will receive higher scores.

   Hint: A naive brute-force approach that repeatedly selects the smallest available element from the arrays results in a runtime of $O(NK^2)$. Consider using a heap, exactly like what we did in recitation, to achieve a better runtime.