Worth: 8% **Due:** By 5:59pm on Tuesday 29 September

Remember to write the *full name* and *student number* of *every group member* prominently on your submission.

Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission ***every*** *source of information you used to complete this homework (other than your own lecture and tutorial notes). For example, indicate clearly the* ***name*** *of every student from another group with whom you had discussions, the* ***title and sections*** *of every textbook you consulted (including the course textbook), the* ***source*** *of every web document you used (including documents from the course webpage), etc.*

For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks ***will*** *be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.*

---

1. Considering the following algorithm which searches for the first appearance of value v in an array A of length $n > 1$. The index of the array starts from 1.

```
FINDFIRST($A, v$):
1   for $i \leftarrow 1, 2, \ldots, n$:
2       if $A[i] = v$:
3           return $i$
4   return $-1$
```

The input array A is generated in the following specific way: for $A[1]$ we pick an integer from $\{0, 1, \ldots, n\}$ uniformly at random; for $A[2]$ we pick an integer from $\{0, 1, \ldots, n-1\}$ uniformly at random; and more generally for $A[i]$ we pick an integer from $\{0, 1, \ldots, n-i+1\}$ uniformly at random. All choices are independent from each other. Now, let's analyse the complexity of FINDFIRST by answering the following questions. All your answers should be in **exact form**, i.e., **not** in asymptotic notations.

NOTE: For simplicity, we assume that v is an integer whose values satisfies $1 \leqslant v \leqslant n$.

(a) In the **best case**, for input (A, v), how many times is Line #2 ("**if** $A[i] = v$") executed? Justify your answer.

(b) What is the probability that the best case occurs? Justify carefully: show your work and explain your calculation.

(c) In the **worst case**, for input (A, v), how many times is Line #2 executed? Justify your answer.

(d) What is the probability that the worst case occurs? Justify carefully: show your work and explain your calculation.

(e) In the **average case**, for input (A, v), how many times is Line #2 expected to be executed? Justify your answer carefully: show your work and explain your calculation.

HINT: Your answers should be in terms of both n and v. Thinking about the number of comparisons needed to find a given v, which values are possible, which values are not?

2. This question consists of two problems that are seemingly unrelated to each other, but we will be able to solve both of them using a similar approach based on the heap data structure that we learned in class.

   (a) The input to this problem consists of k sorted lists $L_1, \ldots, L_k$, each one containing a list of n/k integers in non-descending order. We want to output a single list L that contains all n integers in $L_1, \ldots, L_k$, sorted in non-descending order. Devise an algorithm for solving the above problem with worst-case time complexity $\mathcal{O}(n \log k)$. You should use a heap that can hold **at most *k*** elements. Give a detailed description of your algorithm and explain why it works correctly and has the desired worst-case runtime.

   HINT: Start by thinking about how to merge two sorted arrays. Do **not** use an AVL tree (or other balanced binary search tree)!

   (b) Let a, b, c, d be four integers whose values are between 1 and n, inclusive. Devise an algorithm that finds all solutions to the following equation

   $$a^7 + b^7 = c^7 + d^7$$

   Your algorithm should have worst-case runtime $\mathcal{O}(n^2 \log n)$. You should use a heap of size **at most *n***. Give a detailed description of your algorithm, and argue why your algorithm works correctly and has the desired worst-case runtime.

   HINT: How is this algorithm related to the one in the previous part? What corresponds to the m sorted arrays? How can we obtain the solutions to the equation from the merged sorted array?

   NEW You may assume that the number of solutions (4-tuples which satisfy the equation) is $O(n^2 \log n)$.