CS3230 Semester 2 2024/2025

Assignment 04
D&C, Sorting, and Average-Case Analysis

Due: Sunday, 23rd February 2025, 11:59 pm SGT.

Instructions:

- Canvas Assignment Submission page: `Assignments/Assignment 4`.
- Please upload PDFs containing your solutions (hand-written & scanned, or typed) by the due date.
- Name the file **Assignment4_SID.pdf**, where SID should be replaced by your student ID.
- You may discuss the problems with your classmates at a high level only. You should write up your solutions on your own (any copying from your co-students or usage of Internet or AI tools is not allowed). Please note the names of your collaborators or any other sources in your submission; failure to do so would be considered plagiarism.
- Question listed as "graded for correctness" (worth 6 points) require complete answers. Other questions (worth 1 point each) will be graded only based on reasonable attempts. However, you should still do these questions, as they are practice questions, which would be useful for exams as well as for your knowledge.

1. (6 points; graded for correctness)

Consider the following algorithm to find the two largest numbers from a length-n array $A[1..n]$ of n distinct numbers.

Input: Given n distinct numbers, $A[1..n]$ contains a uniform random permutation of them.

```
Begin
        1. If $A[1] > A[2]$,
                Then let $a = A[1]$ and $b = A[2]$,
                Else let $a = A[2]$ and $b = A[1]$
        Endif
        2. For $i = 3$ to $n$ do:
            2.1. If $b < A[i]$ Then
              2.2. If $a < A[i]$,
                    Then let $b = a$ and $a = A[i]$
                    Else let $b = A[i]$
              Endif
            Endif
        (* Comment: $a$ and $b$ store the two largest elements of the subarray $A[1..i]$. *)
        End For
        3. Output $a, b$.
End
```

(a) (2 points) For a fixed i, in the For loop iteration with index i, at step 2.1: what is the probability that $b < A[i]$? Give proof/justification for your answer.

(Your answer can be in terms of i. **Hint:** Observe the comment in the pseudocode.)

(b) (4 points) What is the expected number of comparisons made by the above algorithm? (Here, we count only the comparisons made between a, b and the array elements in steps numbered 1, 2.1, and 2.2 and *not* any comparisons done in the control statement of the For statement).

Give the bound in the form $n + O(f(n))$, where $f(n)$ is as tight as possible.

Give proof/justification for your answer.

2. (1 point). This question is mainly for you to understand some other sorting algorithms, which are not comparison-based.

   Recall that the lower bound of $\Omega(n \log n)$ for sorting applies if the only operations on elements allowed is comparison among them.

   Counting Sort:

   Let us consider a linear-time algorithm for a special case, when the numbers are in the range $\{1, 2, \ldots, k\}$ for some k, and we are allowed to 'see' the elements of the array.

```
Begin Counting Sort
(* Input Array $A[1..n]$. The values $A[i]$ are in $\{1, 2, \ldots, k\}$. *)
1. For $i = 1$ to $k$ Do
          $Count[i] = 0$
   EndFor
2. For $j = 1$ to $n$ Do
          $Count[A[j]] = Count[A[j]] + 1$
   EndFor
   (* Above is counting how many times each element in $\{1, 2, \ldots, k\}$ appears in the array *)
3. For $i = 2$ to $k$ Do
          $Count[i] = Count[i] + Count[i-1]$
   EndFor
   (* Above accumulates the count: that is, $Count[i]$ is the number of times that elements
      $\leq i$ appear in the array. *)
4. For $j = n$ down to 1 Do
   4.1. $B[Count[A[j]]] = A[j]$;
   4.2. $Count[A[j]] = Count[A[j]] - 1$;
          (* Intuitively, above copies all $A[j] = i$ for each $i$ to locations in $B$ starting from
             $B[C[i]]$ down to $B[C[i-1]+1]$, where $C[i]$ denotes the value of $Count[i]$ at the
             end of step 3. *)
   EndFor
End
```

   Show that B is the sorted array. Moreover, the sorting is stable: Suppose $A[j_1] = A[j_2]$ and $j_1 < j_2$, and in step 5 above, $A[j_1]$ gets copied to $B[j'_1]$ and $A[j_2]$ gets copied to $B[j'_2]$; then $j'_1 < j'_2$. What is the time complexity of the above algorithm (in terms of n and k)?

3. (1 point) Recall the question from Tutorial 4 where you had to find one heavier ball from a group of 243 balls. This question generalizes the problem.

   Suppose we have n balls, and one of them is defective (either lighter or heavier than other $n-1$ balls, but you don't know which ball it is or whether it is lighter or heavier than the other balls). Assume $n \geq 10$.

   You have a balance, which allows you to check when given two sets A, B of balls (placed on either side of the balance respectively), whether: A and B weigh equally, or if A is heavier than B or A is lighter than B.

   Using the above balance, you need to devise an algorithm to detect which ball is defective, and also whether it is heavier or lighter than the other balls. As each weighing is charged, your algorithm needs to do it in as small a number of weighings as possible (in the worst case, in terms of n). For this problem, you need to give an exact answer rather than in Θ notation. You can also assume that the balls are numbered (to be able to easily distinguish the balls).

   Though you are allowed to use any method to solve the above question, the following parts will guide you towards the answer in a structured/simpler way.

   (a) Suppose you are given m balls, each colored either red or green. Among these balls one is defective. Furthermore, you are also given one additional non-defective ball. Suppose you already know that (i) if the defective ball is red, then it must be heavier than the other balls, and (ii) if the defective ball is green, then it must be lighter than the other balls.

   What is the largest value of m, such that you can determine the defective ball among the m colored balls, in the worst case, using at most i weighings?

   Hint: In each round, think about how many (and which) balls you want to put on the balance (and on which side of the balance). The technique is similar to the tutorial question, but you need to be careful in deciding which ball goes where.

   (b) Suppose you have n balls, but no other information (i.e, they are not colored as in part (a)). Among these balls, one is defective. Furthermore, you are also given one additional non-defective ball.

   What is the largest value of n, such that you can determine the defective ball among the n balls (as well as whether it is heavier or lighter than the other balls), in the worst case, using at most i weighings?

   Hint: Again, in each round, think how many balls you want to put on the balance (and on which side). Let $T(i)$ denote the largest number of balls from which you can find the defective ball using i weighings (in the worst case). Then try to express $T(i)$ in terms of $T(i-1)$ and the result from part (a), and then solve for $T(i)$.

   (c) Modify your algorithm and analysis for part (b) when you are not given the additional non-defective ball, to determine the number of weighings needed to find the defective ball (as well as whether it is heavier or lighter than the other balls) among the n balls (where $n \geq 10$).

   Hint: Note that in part (b), the additional non-defective ball is needed only for the first weighing. Afterwards, based on the result of the first weighing, one can determine a non-defective ball among the n balls given. So, think how you want to modify the first weighing — and then get the answer for this part using part (b).

Remark: You should clearly specify your algorithm for how you are selecting the balls to be put on the balance (and which side of the balance you place them on). However, you are not required to give any pseudocode.