Problem Set 2

Due dates: Typeset your solution in LaTeX. Electronic submission of the resulting .pdf file of this homework is due on **Friday, Feb 3, before 11:59pm** on canvas. If your submission cannot be checked by turnitin, then it will not be graded.

Name: Huy Quang Lai

Resources. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. *Introduction to Algorithms*. The MIT Press.

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework.

Signature: Huy Lai

Problem A. Solve the following five subproblems.

Problem 1 (20 points)**.** Give a self-contained proof of the fact that

$$\log_2(n!) \in \Theta(n \log n).$$

[For part of your argument, you can use results that were given in the lecture, but you should write up the proof in your own words. Make sure that you write it in complete sentences, even when the sentence contains formulas. A good check is to read out the entire solution aloud. It should read smoothly.]

Solution.

$$\begin{aligned}\log_2(n!) &= \log_2(1) + \log_2(2) + \log_2(3) + \cdots + \log_2(n) \\ &\leq \log_2(n) + \log_2(n) + \log_2(n) + \cdots + \log_2(n) \\ &= n \log_2(n)\end{aligned}$$

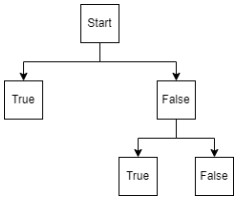
$$\begin{aligned}\log_2(n!) &= \log_2(1) + \log_2(2) + \cdots \log_2\left(\frac{n}{2}\right) + \cdots + \log_2(n) \\ &\geq \log_2\left(\frac{n}{2}\right) + \log_2\left(\frac{n}{2}+1\right) + \cdots + \log_2(n-1) + \log_2(n) \\ &= \frac{n}{2}\log_2\left(\frac{n}{2}\right) = \frac{n}{4}\log_2 n\end{aligned}$$

$\because c_1 n \log n \leq \log_2(n!) \leq c_2 n \log n, \forall n \geq 1$
$\therefore \log_2(n!) \in \Theta(n \log n)$

Problem 2 (20 points)**.** Amelia attempted to solve n algorithmic problems. She wrote down one problem per page in her journal and marked the page with 😩 when she was unable to solve the problem and with 😀 when she was able to solve it. So the pages of her journal look like this:

Use the decision tree method to show that any algorithm to find a page with an 😀 smiley on has to look at all n pages in the worst case.

Solution.



In the worst case, Amelia could not solve any problems and all pages are 😩. However, the algorithm must first search through all pages of her journal to determine if this is true.

Problem 3 (20 points)**.** Amelia attempted to solve n algorithmic problems. She wrote down one problem per page in her journal and marked the page with ☹ when she was unable to solve the problem and with ☺ when she was able to solve it. So the pages of her journal look like this:

Use an adversary method to show that any method to find a page with a ☺ smiley on it might have to look at all n pages.

Solution.

No matter what algorithm is used, it is possible that all $n-1$ pages checked are ☹ and the algorithm must check the n-th page to determine if it is ☺ or ☹ .
Therefore, the algorithm must check all n pages to guarantee if the journal contains a ☺ .

Problem 4 (20 points)**.** Amelia attempted to solve n algorithmic problems, where n is an odd number. She wrote down one problem per page in her journal and marked the page with ☹ when she was unable to solve the problem and with ☺ when she was able to solve it. Suppose that we want to find the pattern ☹☺, where she was unable to solve a problem, but was able to solve the subsequent problem.
Find an algorithm that always looks at fewer than n pages but is able to correctly find the pattern when it exists. [Hint: First look at all even pages.]

Solution.

Loop through the journal looking at only the even-numbered pages.
If the page is ☺ , then the algorithm needs to check the previous page for ☹ .
If the page is ☹ , then the algorithm needs to check the next page for ☺ .
This algorithm guarantees that fewer than n pages are checked. This is because no matter the arrangement of ☺ and ☹ , there is always at least one page that is not checked.

Problem 5 (20 points)**.** Suppose that we are given a sorted array $A[1..n]$ of n numbers. Our goal is to determine whether or not the array A contains duplicate elements. We will limit ourselves to algorithms that use only the spaceship operator `<=>` for comparisons, where

```
a <=> b :=
  if a < b then return -1
  if a = b then return  0
  if a > b then return  1
  if a and b are not comparable then return nil
```

No other methods will be used to compare or inspect elements of the array.

(a) Give an efficient (optimal) comparison-based algorithm that decides whether $A[1..n]$ contains duplicates using the spaceship operator for comparisons.

(b) Use an adversarial argument to show that no algorithm can solve the problem with fewer calls to the comparison operator `<=>` than the algorithm that you gave in (a).

Solution.

Assuming the array is zero indexed.

```
for (size_t i = 0; i < A.size() - 1; ++i) {
    if ((A[i] <=> A[i + 1]) == 0)
        return true;
    return false;
}
```

In the worst case, this algorithm need $n - 1$ comparisons.
The algorithm must check all $n - 1$ pairs of consecutive elements to guarantee that a duplicate element exists. If the algorithm does not check all pairs, it is possible that the two unchecked elements are equal to each other.