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: (put your name here)

Resources. (All people, books, articles, web pages, etc. that have been consulted when producing your answers to this homework)

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: _____

Problem A. Solve the following five subproblems.

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

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

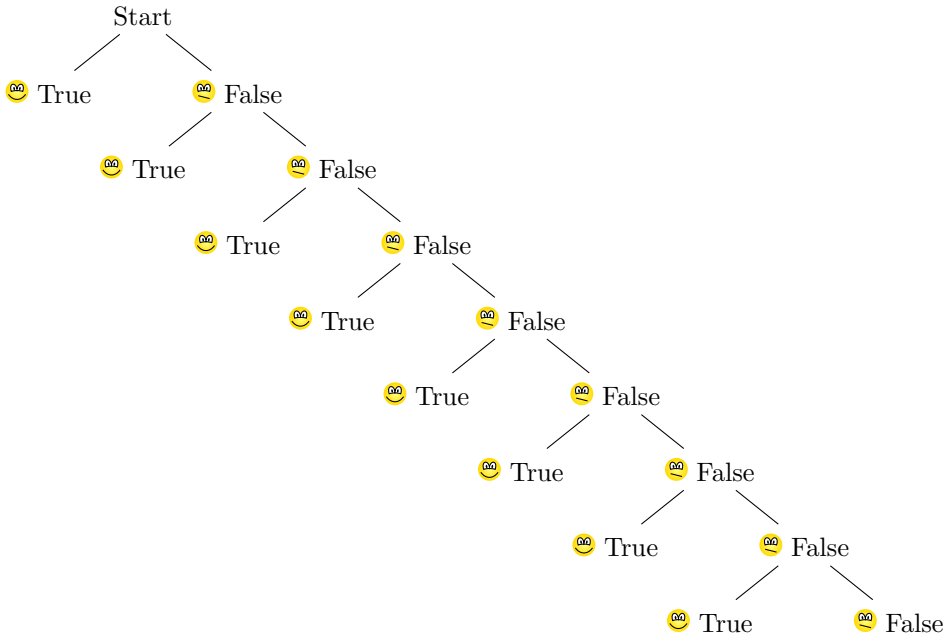
[For part of your argument, you can use results that were given in the lecture, but you should write up the proof in your own words. Make sure that you write it in complete sentences, even when the sentence contains formulas. A good check is to read out the entire solution aloud. It should read smoothly.]

Solution.

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. Amelia's journal pages can be searched through the use of a binary search tree. Each 'level' of the tree would correspond to an individual page, so the height of the tree h is equivalent to the n number of pages. Each node but the last should have one child node for 😊 if the problem was solved and the other for 😞 if she was unable to solve it. The 😊 nodes are leaf nodes, since the search may conclude once a solved page is found. 😞 nodes would continue the binary tree for as many levels as there are pages used. Consider the example provided, where Amelia has 8 pages she wishes to search through. The resulting decision tree would appear as follows:



In the worst case, one of two situations are possible; Amelia was only able to solve the *nth* problem, or none at all. The decision tree must then visit each of the n nodes down the tree until it reaches the *nth* node (page). The status (😊/😞) of this last page is inconsequential for this question, as we have already had to look at each of the n nodes anyway.

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. Under an adversarial method, our adversary is able to control the observation (input) that the algorithm sees and bases its actions on. The adversary is looking to slow the algorithm down as much as possible, and the algorithm cannot see what is actually written on the page(s). It must, unfortunately, rely only on what the adversary says is on the page. There is also nothing stopping the adversary from reordering the pages so that all of the 😒 unsolved pages are at the front, while all of the 😊 solved pages are pushed to the back. The absolute worst case for the algorithm is, as previously described, when all but the last problem are 😒 unsolved. Thus, as the algorithm asks the adversary if a page is solved, the adversary will naturally reply with a 'no'. Upon reaching the *nth* page, the adversary can answer either way as it has already succeeded in its goal of generating a worst case. The algorithm has had to observe each of the n pages.

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.]

Algorithm 1 Pattern Search

Solution.

```
for i ← 2 to n number of pages, step = 2 do
    if page[i] contains TRUE then
        if page[i-1] contains FALSE then return "pattern found at (i − 1)"
        end if
    else if page[i] contains FALSE then
        if page[i+1] contains TRUE then return "pattern found at i"
        end if
    end if
end for
if pattern not found then return "pattern doesn't exist"
end if
```

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. :

Algorithm 2 Finding duplicates in an array

---

for $i \leftarrow 0$ to $n-1$ **do**
  if $A[i] \leftrightarrow A[i+1] == 0$ **then return** "duplicate elements found at i and $i+1$"
    end
  end if
end for
else Output "duplicate elements do not exist"

(a)
In Problem 4, the binary format of the pages (or) allowed for a very efficient search that did not have to look at all n pages. The value of a page meant the algorithm only had to look to one direction from the current page for its operation. This problem is not as friendly, as equality could come from either side of a given point in the array. Since the array is already sorted, we know that any equal elements will be next to each other. Using only the spaceship operator, it would therefore seem that the most optimal solution is to iterate once through the first $n-1$ elements and compare to the next element.
(b)
Before, the worst case of the journal page search arose when all of Amelia's pages were unsolved or only the last page was solved. Similarly, the worst case here occurs when either none of the elements are equal or only the last two are. Say our adversary creates the following array for the algorithm:

2	4	6	8	8

Checklist:

- ☐ Did you add your name?
- ☐ Did you disclose all resources that you have used?
  (This includes all people, books, websites, etc. that you have consulted)
- ☐ Did you sign that you followed the Aggie honor code?
- ☐ Did you solve all problems?
- ☐ Did you write the solution in your own words?
- ☐ Did you submit the pdf file of your homework?