Problem Set 2

> **Due dates:** Typeset your solution in LaTeX. Electronic submission of the resulting .pdf file of this homework is due on **Friday, September 17, 2021, before 11:59pm** on canvas.

Name: [redacted]

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: [redacted]

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.]

Solution. For the first problem, we want to prove that $\log_2(n!) \in \Theta(n \log n)$. We know that $\log(n!) \sim \log(1) + \log(2) + ... + \log(n)$. We can also say that $\log(1) + \log(2) + ... + \log(n) < \log(n) + \log(n) + ... + \log(n)$. Hence, the upper bound of this would be $n \log n$.
We must also find the lower bound, in order to satisfy big theta, so $\log(1) + \log(2) + ... + \log(n) > \log(\frac{n}{2}) + \log(\frac{n}{2} + 1) + \log(\frac{n}{2} + 2) + ... + \log n$ and from there we can subtract $\log(\frac{n}{2}) + \log(\frac{n}{2} + 2) + ... + \log n$ from both sides, which then gives us $\log(1) + \log(2) + ... + \log(\frac{n}{2}) > 0$. Since we have $\frac{n}{2}$ number of $\log(\frac{n}{2})$, the lower bound is $\frac{n}{2} \log(\frac{n}{2})$. This then gives us $\frac{n}{2} \log(\frac{n}{2}) \sim C * \log n$. Thus, we can prove $\log_2(n!) \in \Theta(n \log n)$.

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. For problem 2 we must utilize the decision tree method to show that any algorithm has to go through all the pages to find the one with 😀 , in the worst case.
So using the decision tree method, we would first take page 1 to compare. If page 1 had a 😩 then the tree would branch off and have a set of all possible combinations where page 1 was the one with a 😀 . However, in the case that page one does not have 😀 , then the algorithm would move on to the next page. Then on page 2 it would once again check for the 😀 . If page 2 has the 😀 then the tree would branch off and have a set where all possible combinations of page 1 containing 😩 and page 2 containing 😀 . This algorithm would keep following these same steps above, until all pages are completed and all possible combinations are listed as elements on a node in this tree.

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 an smiley on it might have to look at all n pages.

Solution. For problem 3, similar to problem 2, we must show that any algorithm to find a page with on it has to look through all the pages, however this time we are using the adversary method.
Let's assume that any algorithm can check any number of pages x, where $0 < x < n$ In this case any of these algorithms can check any number of pages and find that they are all , and return that there was no page with a . These kinds of algorithms don't account for all the possibilities of there being a in any of the n-x pages. Thus, this ends up giving incorrect results.

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. For problem 4, we must find an algorithm that always looks at fewer than n pages while still finding the correct pattern when it exists.
We are hinted to look at all the even pages first. So looking at all the even pages first, we also know that the pattern goes like . Using this information, the algorithm can now find which pages had the , given it contained a in the first place, and then look at the previous page. However, if the even page had a then the algorithm will check the next page. Using this algorithm, there would be no checks on pages between the and , in that order.

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. For problem 5, we are given that there is a sorted array A[1...n] of n numbers, and we must find if the array has any duplicate elements.

a) Let A[1..n] be an array of n sorted elements. Let's say we utilized a loop where i=0 and for (n-1) times there's comparisons made for consecutive elements in an array. To check for duplicate elements, we must set conditions, where if the element at i is equal to the element at i+1 or (A[i] <=> A[i+1] == 0) then theres a duplicate element in the array. If the duplicate element is found then exit the loop, and if there's no duplicate elements found then i=n. The time complexity for this would be $O(n-1)$

b) For this part of the problem, we must use the adversarial argument. Let's assume that an algorithm x can take (n-2) comparisons for comparing two consecutive elements. This will help find duplicate elements in the sorted array of A[1..n]. If we let A[1..n] be A[i]=3i, with all unique elements, then while running algorithm x on A[1..n] there will be some element that's not compared to the next one, leaving it with n-2 comparisons. So to determine which element that is A[i+1] should be 3i where we can run algorithm x again and it will say no duplicates found, which would be incorrect. Hence, there can't be an algorithm with less than (n-1) comparisons, when looking for duplicate elements.

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?