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 2, before 11:59pm** on canvas. If your submission cannot be checked by turnitin, then it will not be graded.

Name: Jawahar Sai Nathani

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

https://ctan.math.washington.edu/tex-archive/macros/latex/contrib/algpseudocodex/algpseudocodex.pdf

https://www.ling.upenn.edu/advice/latex/qtree/qtreenotes.pdf

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: Jawahar Sai Nathani

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. Let $f(x) = \log_2(n!)$ and $g(x) = n \log n$

Step 1

$$n! = 1 * 2 * 3 * \cdots * (n-1) * n$$

$$\Rightarrow n! \leq n * n * n * \cdots * n * n = n^n$$

Apply **log** on both sides.

$$\Rightarrow \log_2(n!) \leq \log_2(n^n) = \frac{n \log n}{\log 2}$$

As $f(x) \leq Cg(x) \ \ \forall n > n_0$, where C = $1/\log 2$. Hence $f(x) = O(g(x))$

Step 2

$$n! = 1 * 2 * 3 * \cdots * (n-1) * n$$

From the above expression ignore the first n/2 terms and all the terms after n/2 are greater than n/2. Hence above expression can be written as follows

$$\Rightarrow n! \geq \frac{n}{2} * \frac{n}{2} * \frac{n}{2} \cdots * \frac{n}{2} \quad \longrightarrow \frac{n}{2} \ terms$$

$$\Rightarrow n! \geq (\frac{n}{2})^{\frac{n}{2}}$$

Apply **log** on both sides

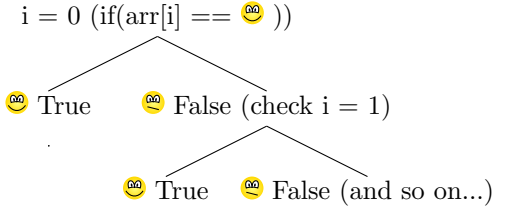
$$\Rightarrow \log_2(n!) \geq \frac{n}{2} \log_2 \frac{n}{2} = \frac{n}{2 \log 2} \log \frac{n}{2}$$

Hence $f(x) = \Omega(g(x))$ for large values of n.
Therefore, as $\log_2(n!) = O(n \log n)$ and $\log_2(n!) = \Omega(n \log n)$ it implies that $\log_2(n!) = \Theta(n \log n)$

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 Decision tree method, at each node we check if the current page of the journal is marked with 😊 . If the page is marked 😊 we will end the algorithm and if the page is not marked 😒 then we move forward. For Example



The algorithm continues until a page with 😊 is found. Therefore, the worst case for this algorithm is when there exists only one page with 😊 symbol at the end of the array and all other pages have 😒 symbol. In this case the algorithm checks all n pages in the array to find a page with 😊 symbol. Hence, the worst case to find a page with 😊 smiley on for any algorithm is to look at all n pages.

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. Let us consider an adversary method which manipulates the input data in such a way that, all the false cases (pages with 😒 smiley) are provided first to the algorithm and true cases (pages with 😊 smiley) later. This input data format generates the worst case or the longest run-time to find the result
Assumption - There exists only one page with 😊 smiley on and rest of the pages are marked with 😒 smiley.

- For the above input data, since all the pages with 😒 smiley are tackled first, any algorithm should look at all n pages to find the page with 😊 smiley marked on it.

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. To find the pattern 😠 😀 , let's iterate through all the even indexes of the array and check the adjacent elements to determine whether the pattern exists. Ideally if the smiley on page "i" is 😀 we will check if page "i - 1" contains 😠 smiley or if the smiley on page "i" is 😠 we will check if page "i + 1" contains 😀 smiley.

Sudo Code -

Let us assume the indexes of the array start from 0.

$i \leftarrow 1$
while $i < length(arr)$ **do**
  if $arr[i] ==$ 😀 $\ and\ \ arr[i-1] ==$ 😠 **then**
    return $i - 1$
  end if
  if $arr[i] ==$ 😠 $\ and\ \ arr[i+1] ==$ 😀 **then**
    return i
  end if
  $i \leftarrow i + 2$
end while
return nil

The worst case for the above algorithm is when all the smileys in the array are 😠 , in this case, the above algorithm looks at n - 1 pages to find a page with 😀 smiley. Hence the algorithm always looks at fewer than n pages and finds the pattern if exists.

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. (a) The given array A is sorted and contains some random numbers. Hence if the array has duplicates they exist adjacent to each other, the optimal solution to find the duplicate would be

Sudo Code -

Let us assume the indexes of the array start from 0.

$i \leftarrow 1$
while $i < length(A)$ **do**
  if $A[i-1] <=> A[i] == 0$ **then**
    return duplicate exits at $i-1$
  end if
  $i \leftarrow i+1$
end while
return duplicate doesn't exist

(b) As the array is sorted, the adversarial argument is having the input data with duplicates present at "n - 2" and "n - 1" indexes of the array (last 2 elements) or an array without duplicates. For Example

10	13	15	17	20	23	24	24

To find if the duplicate exists, every number should be compared with its adjacent number and as the duplicates are present at the last 2 locations of the array, the algorithm has to make n - 1 comparisons to find the duplicates. Therefore, the worst-case run-time for the above algorithm is - "n - 1".

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?