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: Ayushri Jain

Resources. Below resources were used for reference -

- https://latex-cookbook.net/contents/chapter9-creating-graphics/9-3-growing-a-tree/
- https://app.perusall.com/courses/23-spring-csce-411-503-design-analy-algorithms/dm_ch11-270748931
- https://www.youtube.com/watch?v=3F3EKJNGNSQ
- https://www.geeksforgeeks.org/find-repeating-element-sorted-array-size-n/

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: 

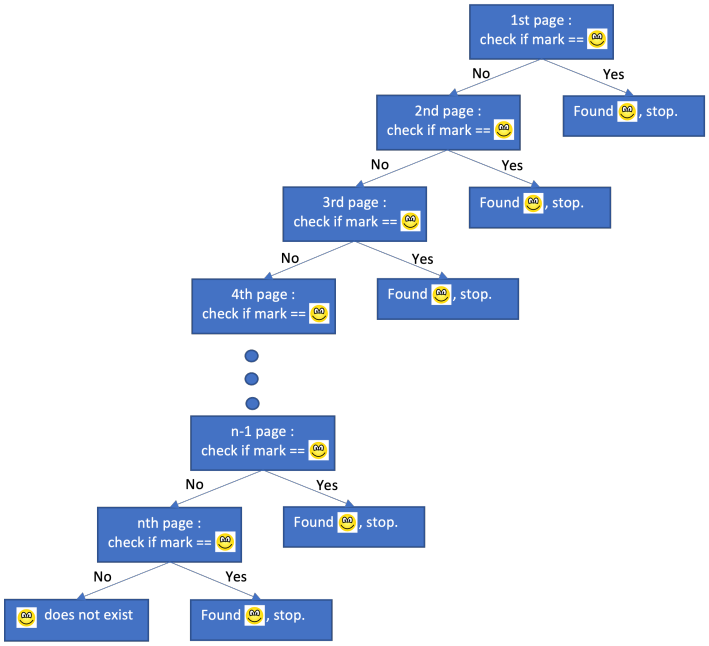
Problem A. Solve the following five subproblems.

Problem 1 (20 points)**.**

Solution. If we are somehow able to get the lower Ω and upper O bounds for $f(n) = \log_2(n!)$, then we can combine them to get the asymptotically tight bound Θ. To get the lower bound Ω of $f(n)$, we can use the inequality of exponential which claims that exponential increases at a faster rate compared to $x^n/n!$, in other words, it says that the exponential functions for any real number x is always greater than the value obtained by taking fraction of x raised to the power of n and n factorial, i.e. $e^x > x^n/n!$ for all $x \geq 0$. If we take x equal to n, then the inequality becomes $e^n > n^n/n!$. Since $n!$ and e^n are always positive, we can move them to the other sides in the inequality which gives us $n! > n^n/e^n$. We can simplify it as $n! > (n/e)^n$. If we take log on both sides, we get $\log(n!) > \log(n/e)^n$ and now if we use exponential rule of logarithm we get $\log(n!) > n\log(n/e)$. Using the division rule of logarithm, we can further simplify as $\log(n!) > n(\log n - \log e) \implies \log(n!) > n\log n - n\log e$. Since we are interested in the lower bound, we can ignore the second term which gives us $\log(n!) > n\log n$. Clearly, this gives us a lower bound $\log(n!) = \Omega(n\log n)$. One important point to note here is that the base of log does not matter, so we can also write $\log_2(n!) = \Omega(n\log n)$.
Let's use definition of n factorial to get the upper bound O of $f(n)$. According to the definition, a factorial of a number n is the product of all positive integers upto n, i.e. $n! = n*(n-1)*(n-2)*\cdots*3*2*1$. If we take log on both sides, we get $\log(n!) = \log(n*(n-1)*(n-2)*\cdots*3*2*1)$. Using the product rule of logarithm, we can simplify this as $\log(n!) = \log n + \log(n-1) + \log(n-2) + \cdots + \log 3 + \log 2 + \log 1$. On the right side of this equality, if we replace each number with n (n times $\log n$), then it will become greater than left hand side. So we will get an inequality $\log(n!) \leq \log n + \log n + \log n + \cdots + \log n + \log n + \log n$ which on simplifying gives us $\log(n!) \leq n\log n$. Clearly, this gives us upper bound $\log(n!) = O(n\log n)$. Since base of log does not matter here also, we can write $\log_2(n!) = O(n\log n)$.
Finally, we can combine lower bound $\log_2(n!) = \Omega(n\log n)$ and upper bound $\log_2(n!) = O(n\log n)$ to get the asymptotically tight bound $\log_2(n!) \in \Theta(n\log n)$.

Problem 2 (20 points)**.**

Solution. The challenge here is to locate a page in Amelia's journal that has a 😀 on it which means that if a 😀 is encountered, the problem is solved. We can simply do a linear search to find 😀 in her journal. Assuming we start searching from the first page, the root of the decision tree will represent that initial page. We will compare the mark on the page with 😀 . If the mark is 😀 , then we have found the 😀 marked page and we stop searching further. But if the mark is 😐 , we need to keep moving to the next pages and search for 😀 .



Height of the decision tree will be equal to the number of pages that we check. Evidently, the worst case will occur when we don't find in initial pages and there is only one in the last page which is the n^{th} page. Since we are doing a linear search, this means that we need to look into all n pages in the worst case. Also, it is possible that does not exist at all, in that case also, we will need to look at all n pages.

Problem 3 (20 points)**.**

Solution. The challenge here is to locate a page in Amelia's journal that has a on it which means that if a is encountered, the problem is solved. Let there be an adversary who is in charge of telling where is placed and we check with him to see if the mark is or not. The adversary is smart so whenever we ask him if a particular page mark is or not, he keeps saying no. Finally when we ask about the last remaining page, he says yes, the mark is . This way we have to ask about all the pages to the adversary (no matter what order). Thus, it means that we might have to look at all n pages to find a page with on it.

Problem 4 (20 points)**.**

Solution. We need to find the pattern in 2 consecutive pages of Amelia's journal. Let's start looking at all even numbered pages. If is present on an even numbered page, we check if its previous page has . Otherwise, if is present on an even page, we check if its next page has on it. Let page numbers start from 1 to n, journal represents the collection of pages and journal[p] represents the page p of journal then algorithm can be stated as :

```
for (journal p = 1 ; p < n ; p+=2) :
        if journal[p] == 😐 :
                if journal[p - 1] == 😐 :
                        return "Found pattern 😐 😐 "
        else if journal[p] == 😐 :
                if journal[p + 1] == 😐 :
                        return "Found pattern 😐 😐 "
if pattern not found:
        return "Pattern not found"
```

This way we will check all even numbered pages ($\lceil n/2 \rceil$) plus their previous or next page which is also ($\lceil n/2 \rceil$). In worst case, we will not find the pattern. So total checks that we will do is equal to $\lceil n/2 \rceil + \lceil n/2 \rceil = 2\lceil n/2 \rceil$. Since n is an odd number, $2\lceil n/2 \rceil < n$ (in particular $2\lceil n/2 \rceil = n - 1$ if n is odd). For example, if $n = 5$, then $2\lceil 5/2 \rceil = 4 \implies 4 < 5$. Hence, to find the pattern, our algorithm looks at fewer than n pages if it exists.

Problem 5 (20 points)**.**

Solution. Below -

(a) Since this is a sorted array, an efficient algorithm can give results in linear time complexity. If there are duplicates in the array, they will be adjacent to each other. So, we can loop over the array and check if the current element is equal to its next element or not using the spaceship operator. In the worst case, we will be checking the penultimate (2^{nd} last) page, so it will be $n - 1$ checks. Hence, algorithm's time complexity is $\Theta(n)$.

```
for i in range(1, n-1):
        if A[i] <=> A[i + 1] == 0: # elements are equal
                return "Duplicates exist."
# did not find duplicates till now
return "Duplicates do not exist."
```

(b) The challenge here is to find duplicates in an array. Since the array is sorted, the duplicate elements will be adjacent to each other. So, if two same elements are found next to each other, then the problem is solved. The only operation allowed is the spaceship operator. Let there be an adversary who is in charge of giving the results of spaceship operator when we give 2 elements to him. The adversary is smart so whenever we ask him the result of spaceship operator on 2 elements, he keeps telling -1 or 1. In this way, at the very end we will be left with only 1 pair of adjacent location where the duplicates exist and the adversary will have to say 0 as output. It means that we will need n-1 comparisons no matter what order we follow. Hence, the algorithm in (a) is the best algorithm to find duplicates in a sorted array and no other algorithm is better than it.