ASSIGNMENT 1
CS210
Dr. Daniel Page

Maksim Sharoika
200446763

Due: Feb. 4th 2022

Pages: 5

Page 1:

✱ Assumption: Array has at least 3 elements. ↳ if less needs to be changed.

1. $A = \{1, 2, 3, 4 \ldots x\}$ (index $0$ … $n-1$) Size of $n$ integers.

Problem: Determine if all the elements in the array are unique.
1. if all unique, return true.
2. if not unique, return false.

a) Algorithm: ElementDistinctness(L, n)
Input: Array A of size n.
Output: "true" if all elements are distinct.
"false" if there are duplicate elements.

```
for i <- 0 to n-2
   for j <- i+1 to n-1
      if A[i] == A[j]
         return false
return true
```

Lines 3–4: $C_1$
Lines 2–4: $C_1 \times$ # of iterations $(j)$
Lines 1–4: $C_1 \times$ # of iterations$^2$ $(i \times j)$
Line 5: $C_2$

b) finite time proof:

To prove the claim (of finite time) we see at the beginning of the algorithm we have an <u>outer loop</u> starting at i at 0 and a <u>inner loop</u> starting at j at 0. The inner loop is bounded by the array size (to the last index, n-1) therefore it will end in finite time because it increments each iteration. The outer loop is bounded by the array size (to the second last index, n-2) therefore it will end in finite time because it increments each iteration. In conclusion because both loops will end in finite time and all other steps are constant we can say the algorithm will end in finite time.

b) Correct output proof:

Before we begin with the proof of correct output I want to establish in a "complete" algorithm run, (the code does not return till final statement) all elements are compared to one another.

| | | | (n-2) | (n-1) |
|---|---|---|---|---|
| Element: | 6 | 9 | 4 | 16 |
| Index: | 0 | 1 | 2 | 3 |

$6 \to 9, 4, 16$, $9 \to 4, 16$, $4 \to 16$

The algorithm will iterate the outer loop until it gets to the second last element, each outer loop iteration will iterate the inner loop until it gets to the last element. The algorithm works its way from right to left. (index 0 to index n-2) and compares all elements to the right of it. It does not need to compare those to the left because they have been compared in a previous comparison. The second last element is compared to the last one, and the last one does not need to be compared further, all elements before it have already compared against it.

False: if the algorithm ever find an element at i and an element at j equal it returns false because i and j are never the same index ($j \leftarrow i+1$) therefore there must be two non-distinct elements.

True: if the algorithm has completed and excited both loops then all elements have been compared to all other elements and it will return true because the if statement was never triggered.

c) Worst Case Scenario: The array is distinct. Therefore all array elements must be compared to all other array elements, then the loop ends and returns.

Page 3:

Final - Initial
↳ for index numbers.

d) $((C_1 \cdot \#\text{iterations}(\text{for-}j)) \cdot \#\text{iterations}(\text{for-}i)) + C_2$

$$\sum_{i=0}^{n-2} (C_1 \cdot i) + C_2$$

$$C_1 \cdot \frac{(n-2)(n)}{2} + C_2$$

$$C_1\left(\tfrac{1}{2}n^2 - n\right) + C_2$$

The j loop is only peformed the averege beetween the shortest and logest runs. i.e $\frac{n! + 0}{\text{lenght} 2}$ = averege j loop

Therefore the algorithm is $O(n^2)$

line 5: takes constant time ($C_2$) and is out of any loops so we leave it on the side. Lines 3 and 4 take constant time and are $C_1$. They are iterated by the inner loop line 2 which is influvenced by the outer loop but is still $O(n)$ this is still nested in the outer loop which is also $O(n)$ outer loop. Therefore a loop nested within a loop is $n \times n$ or $O(n^2)$

-> another way is line 3 & 4 are iterated the most times in worst case and therefore are inside two loops with worst case of $n$, $n \times n = O(n^2)$

2. we need to find constant $C \in \mathbb{R}^+$ and $n_0 \in \mathbb{Z}^+$ such that for all $n \geq n_0$, $f(n) \leq C \cdot O(n)$ } statement.
↳ for a, b, d...

a) $(2022) \leq C(1)$ $C \geq 2022$ to make the equality true, $n_0$ can be anything $n_0 \geq 0$.

$C \geq 2022$ $n_0 = N/A$

↳ if $C = 2022$, and $n_0$ is 0 they they are always equal.

b) $3(n+1)^2 \leq C(n^2)$ $C = 12$

$3n^2 + 6n + 3 \leq 12n^2 = 3n^2 + 6n^2 + 3n^2$

$3n^2 \leq 3n^2$
$6n \leq 6n^2$
$3 \leq 3n^2$
for all $n \geq 1$ squared > linear.

$C = 12$ $n_0 = 1$ Therefore by the definition of big-Oh $3(n+1)$

↳ we need to find a $C$ in the reals$^+$ and an $n_0$ in integers such that $n \geq n_0$ & $f(n) \leq C \cdot O(n)$

Page 4:

c) Proof by contradiction

we assume $2n^2(n-1)$ is $O(n^2)$ there must be constents $C \in \mathbb{R}^+$ and $n_0 \in \mathbb{Z}^+$ such that for all $n \geq n_0$, $2n^2(n-1) \leq Cn^2$ holds. Since $n \geq n_0$, it can be positive so we can divide both sides by $n^2$: $2n-2 \leq C$, this is a contraction because n is unbounded and C is bounded therefore:

$n = \max(\lceil C \rceil, n_0)+1$ is a contraction becase it is larger or equal to $n_0$, but larger than C violating $2n-2 \leq C$ for all $n \geq n_0$. The $\underbrace{\text{plus one}}_{+1}$ is to nudge it even slightly higher.

d) $f(n)$ is $O(g(n))$ Prove: $f(n)$ is $O(h(n))$
$g(n)$ is $O(h(n))$

① $f(n) \leq C_1 \cdot g(n)$, $n \geq n_1$
② $g(n) \leq C_2 \cdot h(n)$, $n \geq n_2$

By ① if the given is true there is a $C_1$ which holds ① for $n \geq n_1$, by similar logic if ② is true there excists a $C_2$ which holds ② for all $n \geq n_2$.

worst case $n > n_0 = n_1 + n_2$

$$\frac{f(n)}{C_1} \leq g(n) \leq C_2 \cdot h(n), \text{ for } n \geq n_0 \qquad C_0 = C_1 \cdot C_2$$

$$f(n) \leq C_1 \cdot C_2 \cdot h(n) \Rightarrow f(n) \leq C_0 h(n) \quad ③ \text{ for } n \geq n_0$$

Now we simplify if $\frac{f(n)}{C_1}$ is less than $g(n)$ which is less than $C_2 h(n)$ we end up with equalty "3." once simplified, (using $C_1 \cdot C_2 = C_0$) Therefore it become $f(n) \leq C_0 h(n)$ which holds for all $n \geq n_0$ as desired by the definition of Big-oh.

Optional Problem: (we will simplify $\log_2 = \log$) n for work.

$$\sum_{i=1}^{k} \log(n^i) = \sum_{i=1}^{k} i \log(n) = \frac{k(k+1)}{2} \log(n)$$

Firstly we simply the notation so we can apply big O to it, k is just the number of summals.

Positive Integer constant

$$\frac{k(k+1)}{2} \log n \leq C_0 \cdot \log n, \text{ for } n > n_0$$

$$\log n \leq \underbrace{\left(\frac{C_0}{\frac{k(k+1)}{2}}\right)}_{\text{This can be further simplified as a single constant } C_1} \log n$$

* we established a few things more on next page.

Page 5: $\log n \leq \frac{C_0}{\frac{k(k+1)}{2}} \log n.$

Once the value of K is chosen as a positive integer the term $C_0/(k(k+1)/2)$ can be simplified to a single constant because it is a constant divided by a constant.

$\log n \leq C_1 \log n.$

Therefore we can now rewrite in the base of 2 that it was

$\log_2 n \leq C_1 \log_2 n.$

Now we need to find a $C_1$ and $n_0$ such that the above is true for all $n \geq n_0$.

Any $C_1 \geq 1$ will work because it is the same function on both sides. i.e. $x = x$; and $n_0$ would be $n_0 = 2$

$C = 2$

$\therefore \log_2 n \leq \log_2 n$ for $n_0 = 2$

We can also now say that:

$2 = \frac{C_0}{\frac{k(k+1)}{2}}$ from the earlier simplification statements.