Algorithms HW1 Summer 2019 Tae Myles

1. From L2-Buffet slide, we know that this computer runs at $4 * 10^9$ clock cycles per second. It requires about 200 clock cycles to execute one computation step.
Therefore computer is running at, $(4x10^9)/200 = 2x10^7$ and the size is $2x10^8$.
So, A1: $O(n) = (2x10^8) / (2x10^7) = 10s$
A2: $O(nlogn) = ((2x10^8) * \log_2(2x10^8)) / (2x10^7)$
$= ((2x10^8) * 27.57542475)) / (2x10^7)$
$= 5515084950 / (2x10^7)$
$= 275.75s$
A3: $O(n^2) = ((2x10^8)^2) / (2x10^7)$
$= ((2x10^{16}) / (2x10^7)$
$= 2x10^9s$

2. Our goal is to get riders to find transportation with empty seats, this can be achieved by having drivers with empty seats allow the request made by the customers. Customers location is known by their longitude and latitude.

1) The inputs we need are list of requests made by the customers and directions between each pair of location which is start and customers destination.

2) The discrete structures/data representation is to store the location of the customers (latitude, longitude), their destination, and finding the shortest path given the input.

3) The output we are looking for is the shortest route, but only visiting each location once and returning to the next location.

3. <u>Steps for finding the i largest numbers in sorted order.</u>

1) First, check if array size is 1, if so, element in that array is the largest number.

2) Add the first element in the array into a variable, say var.

3) Iterate though the array and compare the values in the array with var. If element in the array is bigger than var then set var equal to that element.

4) Replace where largest element was with the next element in the array.

5) Set the corresponding next element with the largest var.

6) We will be given sorted array numbers from lowest to highest.

$T(n) = c1(n+1) + c2n(n+1) + c3n^2 + c4n^2 + c5n^2$
$= c1n+c1 + c2n^2+c2n + c3n^2 + c4n^2 + c5n^2$
$= n^2(c2+c3+c4+c5) + n(c1+c2) + c1$
$= c6n^2 + c7n + c1$; where $c6 = c2+c3+c4+c5$ and $c7 = c1+c2$
# of steps: 23

4. a) Output A: 55
b) Output B: 0
c) Output C: 0
d) Output D: 14
When input contains all negative integers: Returns 0.
When input contains all non-negative integers: Returns all of the elements added in the array.

5.

Step	Big-Oh complexity
1	O(1)
2	O(1)
3	O(n)
4	O(1)
5	O(n)
6	O(1)
7	O(1)
8	O(1)
9	O(1)
Complexity of the algorithm	$O(n^2)$

6.

Step	Cost of each execution	Total # of times executed
1	1	1
2	1	1
3	1	n+1
4	1	n
5	1	$\sum_{i=1}^{n} i+1$
6	6	$\sum_{i=1}^{n} i$
7	3	$\sum_{i=1}^{n} i$
8	2	$\sum_{i=1}^{n} i$
9	1	1

$T(n) = 1 + 1 + 1 + n + (n + 1) + \sum_{i=1}^{n}(i) + \sum_{i=1}^{n}(i) + \sum_{i=1}^{n}(i) + \sum_{i=1}^{n}(i + 1)$

$= 4 + 2n + \left(\frac{n(n+1)}{2} + n\right) + 6\left(\frac{n(n+1)}{2}\right) + 3\left(\frac{n(n+1)}{2}\right) + 2\left(\frac{n(n+1)}{2}\right)$

$= 4 + 3n + 12\left(\frac{n(n+1)}{2}\right)$

$= 4 + 3n + 6(n(n+1))$

$= 4 + 3n + 6n^2 + 6n$

$= 6n^2 + 9n + 4$

7. The algorithm is incorrect. It is supposed to print out identical and numbers that consecutively follow. However, it only prints numbers that are in pairs, for example.

f = 1 2 3 3 3 4 3 5 6 6 7 8 8 8 8
Index - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Iteration	i	j	Pair	Count
1	1	2	1,2	0
2	3	4	3,3	1
3	5	6	3,4	1
4	7	8	3,5	1
5	9	10	6,6	2
6	11	12	7,8	2
7	13	14	8,8	3
8	15	16	8, null	3

It will only count to 3, instead of 9. Hence, this algorithm is incorrect.

8. Proof by contradiction: Suppose the algorithm A is incorrect. Assuming array will be unsorted, it will not be able to partition the array. However, algorithm A will be able to find the kth largest number, if the algorithm keeps count of which elements are greater than the kth largest and which elements are less than it. With this information it is possible to partition the array. However, this is a contradiction because we shouldn't be able to partition such array because we assumed algorithm A is incorrect. Hence, this algorithm is correct.

9. Base case: n = 0: $3^n-2^n = 3^0-2^0 = 1-1 = 0$
n = 1: $3^n-2^n = 3^1-2^1 = 3-2 = 1$
Base case holds for all n >= 0.

General case: 3^n-2^n for n € positive integer we are trying to prove $5(g(n+1)-1) - 6(g(n+1)-2) = 3^{n+1} – 2^{n+1}$. We've checked the base cases, now to check n-1:

$5(g((n-1)-1)) – 6(g((n-1)-2)) = 3^{n-1}-2^{n-1}$
$\Leftrightarrow$ $5(g(n-2)) – 6(g(n-3)) = 3^{n-1}-2^{n-1}$

Hence, n >= 3, we know this will work because from the base case because as n gets larger so does the output. Hence, this algorithm will work for 3^n-2^n for all n>=0

10. Initialization: It holds before the start of first iteration of the loop with i = 2 because max = A[1].

Maintenance: Suppose that before the start of ith iteration max contains A[1..i-2]. If A[i] > max, then max = A[i-1]. Where max contains maximum element ranging A[1..i-1]. If A[i] <= max, then max is unchanged and maximum value is still in A[1..i-1]

Termination: Initialization and maintenance showed that it holds true, therefore will be true before every nth iteration and where max is the maximum value of A[1..n]. So this algorithm finds the maximum value in the array correctly.

11. Initialization: Is holds before the start of the loop. $t > 0$ and loop variant, $n = t2^k + m$. Since $t = n$, t is set to $t2^k + m$. Then $k = 0$, since k represents the binary array b[1..k] and $t = t2^k + m$, so m is 0. $t = t(2)^0+0 = t$ Hence it holds.

Maintenance: Suppose that before the start of iteration of the loop, the loop variant holds for $n = t2^k + m$. Since $t > 0$ so while loop is active. k becomes 1 from $k = k + 1$. b[k] = t mod 2, hence b[k] = 1. Then t is divided by 2 consecutively. So it holds for $t+1^{th}$ iteration.

Termination: Initialization and maintenance showed that it holds true. Therefore will be true for loop invariant ending at the nth iteration. We know it holds true before the t^{th} iteration. Before $n + 1^{th}$ iteration, b contains an array bits corresponding to the binary representation of n which contains t iterations of bits until the loop stops (e.g. t = 0). Hence, this algorithm correctly converts bits into b.

12. a) Algorithm ReverseString(A: Array of characters, p: Starting index, q: Ending index)

```
Begin
    If (q <= p)
        Return A
    Else
        Swap(A[p],A[q])
        p++
        q—
        ReverseString(A,p,q)
End
```

b) p=0, q=7 i<33270!
p=1, q=6 !<33270i
p=2, q=5 !03327<i
p=3, q=4 !07323<i
p=4, q=3 !07233<i (String changes here not because p = 4, q = 3, but this is after p = 3, q = 4 swap() method has been called on the string. The algorithm ends at p = 4, q= 3 because there is a condition stating if q <= p return the string)