Stephen Sallas
02/01/2021
COMP-3720

HW1

1. $\frac{4*10^9}{200} = 2 * 10^7$ steps per second, so…

   O(n) = $\frac{20*10^7}{2*10^7} = 10$ seconds

   O(nlogn) = $\frac{\frac{(20*10^7)\log(20*10^7)}{log2}}{2*10^7}$ = ~275.75 seconds

   O(n^2) = $\frac{(20*10^7)^2}{2*10^7} = 2 * 10^9$ seconds = ~63.42 years

2. Inputs and their data representations would be source location of user (latitude, longitude), direction of user (degrees from 0° North), selected building (latitude, longitude), and the queried parking lot (latitude, longitude).

   Desired outputs would be visual representation of the path from the source location to the parking lot. Both point would be highlighted, and the path would shrink and adapt as the user approaches. Directions would constantly be given an updated as the user proceeds with the directions. The number of vacant spots and their locations in the parking lot would also be shown and constantly updated.

3. I would first find the median of the sorted sequence (n/2) and compare it to a_i. If they are equal, the job is finished. If not, we compare i to this new median number and determine if it larger or smaller. If larger, we find the median of the half of the sequence above the current median and vice versa if it is smaller. We then compare this to i and repeat the process until it the job is completed.

4. Array (a) output = 55

   Array (b) output = 0

   Array (c) output = 0

   Array (d) output = 11

   When the array is all negative numbers, the output will always be 0 because the sum will never be higher than the max of 0.

   When the array is all non-negative numbers, the output is the sum of all the numbers in the array.

Stephen Sallas
02/01/2021
COMP-3720

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	2	$\frac{n^2 + 2n + 4}{2}$
6	6	$\frac{n * (n + 1)}{2}$
7	4	$\frac{n * (n + 1)}{2}$
8	2	$\frac{n * (n + 1)}{2}$
9	2	1

T(n) = $n + 1 + n^2 + 2n + 4 + 3\big(n * n(n + 1)\big) + 2\big(n * n(n + 1)\big) + n * n(n + 1) + 2$

= $6(n + 1)n^2 + n^2 + 3n + 7$

Stephen Sallas
02/01/2021
COMP-3720

7. This algorithm is incorrect, and is proved by Counterexample. This is easily seen when the algorithm is iterated through and the file of integers contains “1 2 3 3 3 4 3 5 6 6 7 8 8 8 8".

   Iteration #1: i = 1, j = 2, count = 0

   Iteration #2: i = 3, j = 3, count = 1

   Iteration #3: i = 3, j = 4, count = 1

   Iteration #4: i = 3, j = 5, count = 1

   Iteration #5: i = 6, j = 6, count = 2

   Iteration #6: i = 7, j = 8, count = 2

   Iteration #7: i = 8, j = 8, count = 3

   Iteration #8: i = 8, j = END OF FILE, count = 3

   This algorithm returns 3 when the correct answer is actually 9, therefore this algorithm is proved incorrect.

Stephen Sallas
02/01/2021
COMP-3720

8. Base case:

Assume $n = 1$

$3^n - 2^n = 3^1 - 2^1 = 3 - 2 = 1$

This is true.

Induction Hypothesis:

Assume $n = k$

$3^k - 2^k$

Inductive Step:

Assume $n = k + 1$

$3^n - 2^n =$

$3^{k+1} - 2^{k+1}$

$5 * g(n-1) - 6 * g(n-2) =$

$5 * g(k) - 6 * g(k-1) =$

$5 * (3^k - 2^k) - 6(3^{k-1} - 2^{k-1}) =$

$15 * 3^{k-1} - 10 * 2^{k-1} - 6 * 3^{k-1} + 6 * 2^{k-1} =$

$9 * 3^{k-1} - 4 * 2^{k-1} =$

$3^{k+1} - 2^{k+1}$

This is proved true.

Stephen Sallas
02/01/2021
COMP-3720

9. Initialization:

The variable “max” is correctly set to the first integer in the input array’s sequence. It does not matter what order the array is in, so the algorithm holds prior to the first iteration of the loop.

Maintenance:

The algorithm iterates through every integer in the array after A[1] (A[2], A[2+1], A[2+2]…) until it gets to the end at A[n]. It compares each integer to the one before it. If the integer being tested, A[i], is greater than the current max then A[i] replaces and becomes the new max. If not, nothing happens and the iterations continue with no problems. This proves the algorithm holds through all iterations of the loop.

Termination:

After the loop is finished, all integers will have been compared and the variable “max” will be holding the greatest integer from the array (max = Max(A[1…n]). This is the value that is returned from the algorithm. This concludes that the algorithm holds all the way through and is proved correct.

Stephen Sallas
02/01/2021
COMP-3720

10.

Initialization:

The output array “b” begins completely empty. The values t = n, k = 0, and m (the integer represented by the binary array) is also 0.

($n = n * 2^0 + 0$) = (n = n)

The algorithm holds prior to the first iteration of the loop.

Maintenance:

To prove this algorithm works through each iteration of the loop, we have to prove it works for iteration j and then it will also be true for iteration j+1. Assuming $n = t * 2^k + m$…

If t is an even number, t % 2 = 0, t = t/2, k = k + 1, and m does not change. $n = \frac{t}{2} * 2^{k+1} + m$.

If t is an odd number, t %2 = 1, t = (t-1)/2, k = k + 1, m = m + 2^k, b[k+1] = 1. $n = \frac{t-1}{2} * 2^{k+1} + m + 2\hat{}k$

This proves the algorithm holds through all iterations of the loop.

Termination:

The loop terminates when t = 0, because$n = 0 * 2^k + m = m$. This results in $n = m$; proving that when the loop is terminated the algorithm correctly outputs an array of bits corresponding to the binary representation of n. This concludes that the algorithm holds all the way through and is proved correct.

Stephen Sallas
02/01/2021
COMP-3720

11.

a. Pseudocode:

```
Reverse(A[p…q])
    If p<=q Then
        swap(A[p], A[q])
        reverse(A[p+1…q-1])
```

b. p = 0, q = 7

Executions	**Outputs**
Reverse(A[0…7])	i<33270!
Swap(A[0], A[7])	i<33270!
Reverse(A[1…6])	i<33270!
Swap(A[0], A[7])	i03327<!
Reverse(A[2…5])	i03327<!
Swap(A[0], A[7])	i07323<!
Reverse(A[3…4])	i07323<!
Swap(A[0], A[7])	i07233<!
Reverse(A[4…3])	i<33270!