**Name: Jason Truong**

For each of the below questions, write a short sentence or two to express (in your own words) your answer. Keep the answers short, but use complete, correct, English sentences.

If it helps to clarify the questions, feel free to mentally prefix all the questions with the phrase "According to the video…"

* After you’ve watched all the videos, please answer this question:  
  Of all the videos that you watched, if you could pick one video to be re-recorded by the instructor outside of class which would you choose? Why?  
  (Keep in mind the recording outside of class will omit any pauses from the instructor answering student questions, have less hemming and hawing, etc, and generally be more concise)

|  |
| --- |
| < Write your answer here > |

**VIDEO: Linear/Binary Search**

* Let’s say that you want to determine if a particular value is present in an array.  
  Briefly and intuitively, describe how a linear search operates.

|  |
| --- |
| It just basically walks through the array and searches in a straight line through as a search. |

* In the video, what does the variable **N** represent?

|  |
| --- |
| The number of elements inside the array. |

* Why is this search method called “linear” search?

|  |
| --- |
| 'Items in the area' to 'time to find the item in the array', when placed on a cartesian graph is visually linear. |

* What sort of arrays can you use a linear search on?

|  |
| --- |
| Any array you have, you just need some mechanism to say, "Is this what I'm looking for?" |

* In addition to the properties needed for a linear search, what must also be true of an array in order to use a binary search on it?

|  |
| --- |
| The array is sorted in some order. For example, ascending.  Any order works as long as the order is well defined. |

* How does a binary search begin?

|  |
| --- |
| In the middle. |

* If that array element is not the target value, then what will the binary search do next?

|  |
| --- |
| It compares its position in the sort, and go into the correct direction of the half the targeted element may be in. |

* Briefly describe how the binary search continues searching (after the initial check)

|  |
| --- |
| It will take half the of the desired array the target element is in, take that half and split it again, and repeat the first step again. Continuing to split again and again. |

* What is an important test case to check when testing a search algorithm (such as binary search)?

|  |
| --- |
| When the searches can't reduce to one value, out of bounds, and non-target value. |

**VIDEO: Binary Search on paper: Example Solution**Notice that in this lesson in the In-Class Exercises portion of the web page there are a number of documents linked to by **“Linear Search By Hand”** or **“Binary Search By Hand”** or “**Bubble Sort By Hand”**. This video explains how to do the “**Binary Search By Hand**” on-paper exercise.   
  
It is recommended that you try to do the exercise on your own before you watch this video, then check your work against the explanation in the video.

* During Iteration 0, what values are tracked, and what does each value mean?

|  |
| --- |
| The target value is 17: this is the value you're searching for.  Max index is 15: is the upper-limit the program will use to search with @ the array.  Min index is 0: is the lower-limit the program will use to search with @ the array.  Index 1/2 in between is the middle integer of the upper-limit/2, represented as an integer: 7.  "Value Found at the middle index?" (23!=17) = False. The description is on the tin.  Next iteration change \_\_\_\_\_ to:.. changes the upper-limit or lower-limit based on the index at the 1/2 position. Whether its going to be less than or more than the target index. |

* At the end of Iteration 0, why does max change to 6?  
  (Also - why does it not change to 7?)

|  |
| --- |
| Since the program already knows it isn't 7, it moves the entire program to check the indexes from the entire left hand side not including 7.  So the max would be 6.  If it were in the opposite direction, it would be 8, and not 7 as the minimum. |

* At the end of Iteration 1, what value is changed to 4, and why?

|  |
| --- |
| The target value: 17, is tested to be found inbetween the lower and upper limits of 0 and 6. The value inbetween 0 and 6 is 3, and the index holds the value of 3. And we find that 7 isn't 17.  So it checks the array index if 7 is greater or less than 17.  7 is less than, so anything before the 3rd index isn't to be concerned with, and so it changes the minimum index to everything after 3; which starts at the 4th index. |

Make sure that you check your work against the remainder of the video

* Given how awesomely efficient binary search is, why don’t we always use binary search on every array that we want to search?

|  |
| --- |
| Despite the binary search being amazing at what it does, the array has to be sorted for the search to even function - cumbersome. |

**VIDEO: Bubble Sort**

* About a minute into the video BubbleSort is quickly summarized. According to that summary, how does BubbleSort (basically) operate?

|  |
| --- |
| A bubble sort is a loop of going through the array to swapping the arrays-placements out of order.  Incrementing, so it would set from lowest to highest, left to right. |

* For BubbleSort, how are numbers swapped inside the computer?  
  In contrast, how will we represent numbers being swapped when doing the paper exercise?

|  |
| --- |
| The computer will check if the integer on the left placement is out of order from the right placement.  On-paper we could only compare one integer at a time. |

* (Roughly) how many elements does Pass #1 compare?

|  |
| --- |
| 22 checks on elements, but unique elements are 8. |

* In Pass #1, what happens if you find the largest element somewhere in the beginning/middle of the array?

|  |
| --- |
| It would always be out of order, so it will make the full pass through the array and swap it til' it reaches the end. |

* After one pass through the array, what is true about the values in the array?

|  |
| --- |
| The next largest value is passed and placed towards the end respective to the right value. |

* In the pseudocode, what is the outer loop used for?  
  Because of this, why is it ok that you never actually use the outer loop’s counter inside the inner loop?

|  |
| --- |
| The first loop would keep looping N number of times, where N is the number of elements in the array.  The second loop goes through every single element of the array and loops up to the last slot of the array to do the actual swap. However, this may not reach the literal end of the array. |

* Briefly, intuitively explain one (or more) optimization(s) there were briefly mentioned in the video.

|  |
| --- |
| It's very easy to code and only needs two or three variables to actually code, but is not exactly efficient. |

**VIDEO: Bubble Sort on paper: Example solution**

Notice that in this lesson in the In-Class Exercises portion of the web page there are a number of documents linked to by **“Linear Search By Hand”** or **“Binary Search By Hand”** or “**Bubble Sort By Hand”**.

This video explains how to do the “**BubbleSort By Hand**” on-paper exercise.   
It is recommended that you try to do the exercise on your own before you watch this video, then check your work against the explanation in the video.

* When do you copy the current array into a “fresh” (blank) array?  
  (Remember that you do NOT copy the array over every time that you do a comparison! You’ll get a hand cramp AND run out of paper trying to finish the exercise! )

|  |
| --- |
| You don't write something down unless something changes. |

* Between the “Starting Point” row and “Swap 0”, which two elements are swapped? Why?

|  |
| --- |
| 12 and 17.  17 > 12, so swap. |

* Between “Swap 0” and “Swap 1”, which two elements are swapped?

|  |
| --- |
| 21 and -3. |

* Between “Swap 1” and “Swap 2”, which two elements are swapped?

|  |
| --- |
| 21 and 0. |

* Between “Swap 2” and “Swap 3”, why do we “rewind, and go back to the beginning of the array”?

|  |
| --- |
| To make another sweep of variables that need to proceed down the line to be swapped and organized. |

* After which rows does Pass #2 end?

|  |
| --- |
| Swap 4. |

**VIDEO: Bubble Sort: Warning about the Null Reference test case in this week's tests**

* The BubbleSort method is public; what does this imply about the possible values of the parameters?

|  |
| --- |
| It will take a lot of values that may cause the program to crash. |

* Give one example of a line of C# code that would cause the program to crash if your BubbleSort method was given null as a parameter; make sure to explain why it causes the crash.

|  |
| --- |
| null, despite it being an extremely common practice, it will crash it.  Passing in a value of 'null' isn't exactly passing an array.  Method(null); |

* Give an example of C# code that will check if BubbleSort’s parameter is null or not:

|  |
| --- |
| if (nums != null) return; |

* Give the example from the video that checks if the array parameter is null or else if the length of the array is zero. Also explain why that line will not crash when the parameter is null.

|  |
| --- |
| if(nums == null || nums.Length == 0) return;  The reason why this happens cause it makes checks from left to right on the parameters. Coding OOP |

* Is it possible to create an array that is not null yet has a length of zero?  
  If so, then give an example of C# code that does this.

|  |
| --- |
| int[] unums = new int[0] |

**VIDEO: Big "Oh" notation**

* What can Big Oh notation be used to measure?

|  |
| --- |
| How much time an space an algorithm will consume. |

* Overall, what is the goal of using Big Oh notation? What can we use it to do?

|  |
| --- |
| Make decisions of the most efficient search and seek loop. |

* When in the software development lifecycle would you use Big Oh notation?

|  |
| --- |
| Before any software has been written. Early on, in a planning phase. |

* For the 'simple example' of printing out everything in array, what will control how many times the body of the loop runs?

|  |
| --- |
| The integer 'i' reaching the nums-array's length. |

* Is Big Oh notation good for analyzing small arrays? Why or why not?  
  What sized arrays is Big Oh notation good for analyzing

|  |
| --- |
| No, cause generally small arrays have about the same time to compile and perform - nothing distinct will occur.  Larger arrays will more desirable to measure. |

* Why/how do we ignore any coefficients on the leading term of the formula that exactly describes how much time the algorithm will need?

|  |
| --- |
| O(N) |

* If we say that an algorithm takes O( N ) time (where N is the length of the array), what are we actually saying about the exact time that it takes to run the algorithm.

|  |
| --- |
| There exists a function that has a coefficient \* the length of the array + some constants. |

* Is Big Oh notation an upper or lower bound on the amount of time that the algorithm will take?

|  |
| --- |
| Upper bound, it uses a separate system to measure lower bound. |

**VIDEO: Big "Oh" notation & linear search**

* What is one complication that arises when analyzing the run-time of linear search?

|  |
| --- |
| What if you find it in the first space in the array? |

* In analyzing the run-time of any algorithm (say, linear search), what assumption are we going to make about the values contained in the array?

|  |
| --- |
| The array contains a collection of essentially random numbers. |

* For linear search, on average, how far do we expect to go through the array before finding the target value (or determining that the value is not present)

|  |
| --- |
| On average it's evenly distributed in the array, but overall we should expect to go over halfway. |

* When writing out the running time of linear search, why is it that we do not see the coefficient listed in the video?

|  |
| --- |
| The whole point of the BIG(O) notation is that we should hide the coefficient no matter what. 'O' hides the extra coefficient. |

* When examining the Excel spreadsheet in the video, what do the values in the “Array Size” column mean?

|  |
| --- |
| Its a literal title as it is the size of the array. |

* When examining the Excel spreadsheet in the video, what do the values in the “Found” column mean?

|  |
| --- |
| The on-average end-point you end on. |

* When examining the Excel spreadsheet in the video, what do the values in the “Not Found” column mean?

|  |
| --- |
| How many elements in the array we go through to find the items in the array. |

* **What is the running time of the linear search algorithm, in the Big Oh notation?  
  (Make sure that you (briefly) note what N is!)**

|  |
| --- |
| The algorithmic runtime of N; the size of the array, is correlated directly to the number of comparisons made and how long it takes. It's essentially linear, as the name implies. |

**VIDEO: Big "Oh" notation & binary search**

* Why is it good to compare algorithms based on looking at the ‘Found’ column?

|  |
| --- |
| In the scenario you do find the array, how often; on-average does it take to find an array. |

* Why is it good to compare algorithms based on looking at the ‘Not Found’ column?

|  |
| --- |
| In wort-case scenario you don't find the array, you can measure its comparisons to the other algorithms. |

* Why don’t we use binary search for every single search we want to do on every single array we want to search?

|  |
| --- |
| Because not every array is set-up properly for binary search - as the array needs to be sorted sequentially. |

* **What is the running time of the binary search algorithm, in the Big Oh notation?  
  (Make sure that you (briefly) note what N is!)**

|  |
| --- |
| For the beginning of the Binary algorithm's runetime, it will spike in comparisons depending on 'N' number of elements in the array. However, the algorithm will slowly plataeu and reach and upper-bound.  nlog[2](n). |

**VIDEO: Big "Oh" notation & bubble sort**

* In the video we examine the behavior of the BubbleSort algorithm. Is the implementation that we’re examining optimized in any way?  
  (Although it was not covered in this video, make sure that you remember what some possible optimizations are)

|  |
| --- |
| This is an unoptimized bubble sort. 9.6E+07 isn't exactly efficient. |

* When examining the Excel spreadsheet in the video, what do the values in the “Sorting Swaps” column mean?

|  |
| --- |
| The number of times an element is moved through the array. |

* If you were to graph the linear search on the same graph as BubbleSort, how would the linear search appear?

|  |
| --- |
| If the variable 'N' is a large pool of information to wade through, the Linear search would appear to be a less efficient form of the BubbleSort up til an 'N' point. |

* Given a 20,000 element array that you want to find a single, specific value in would it be better to do a BubbleSort followed by a Binary Search, or would it be better to skip the sorting and go straight to a linear search?   
  Explain (briefly and intuitively) your reasoning.  
  Note: There is a correct answer to this question, but because the video does not say it you will not lose points for failing to list it here. You should think about the question, reason your way to a conclusion, then concisely explain your answer (your reasoning) here.

|  |
| --- |
| In a 20,000 element array, you can find a single element in a linear amount of time on the 'Linear Search Algorithm'.  With the Bubble Sort to Binary Search, the array woud have a one-time constant added to the equation as it needs to sort through the array to organize it. Then from then on, up until a new element is added to the array, the Binary Search will have a consistently efficient time with the sorted array. |

* **What is the running time of the bubble sort algorithm, in the Big Oh notation?**(You may need to look this up online)

|  |
| --- |
| As the 'N' increases, the sort takes an exponential amount of time. O(n^2) in both average and worst-case. |