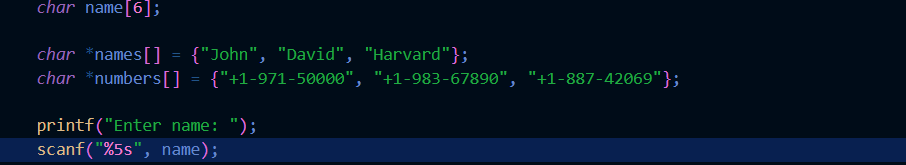
1. Array – contiguous memory locations that store the same type of data.
2. Memoization is a technique used to improve the performance of your algorithm by “remembering” previous results.
3. Hashing is a technique used to store and retrieve data efficiently from a hash table using a hashing algorithm.
4. Linear search – Moving from left to right, through each element. O(N) complexity.
5. Binary search – Only possible for sorted arrays. We move to the middle, then go to the left or right half depending on what we’re looking for. O(logn) complexity.
6. The reason we say that a program with complexity O(N) and another with O(N/2) are the same in terms of performance is because when N gets extremely large, both the algorithms on a bar graph would start to resemble the same line. They aren’t the same thing, but they perform similarly as N gets larger and larger.
7. O notation is to calculate the worst case scenario.
8. Omega (Ω) notation is to calculate the best case scenario.
9. So graphically speaking, O would be the upper bound, and Ω would be the lower bound. The average performance of your algorithm would lie between these two.
10. Example: Linear search has a worst case scenario of n steps, and a best case of just 1 step. Therefore, it has O(N) (worst case) and Ω(1) (best case)
11. Certain algorithms have the same best and worst case scenario performance. That is called a Θ performance. (Theta)

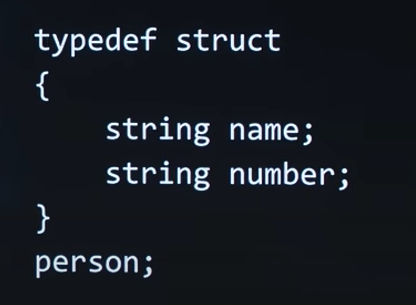
Strings:

1. Suppose we are trying to declare an array, with multiple strings within it. We might go about declaring it the following ways:
2. char strings[][] = {"battleship", "boot", "hat"};
   1. The syntax above is incorrect, because the second square bracket would require the size of the longest string in that array (+1 for the \0). So if we were to write 11 in the second square bracket, then it would initialize fine.
3. char \*\*strings = {"battleship", "boot", "hat"};
   1. The above syntax is wrong. We are declaring a variable called strings, which is a pointer to a pointer of type character, and then we are trying to put multiple characters in it without mallocing the appropriate amount of space.
4. char \*strings[] = {"battleship", "boot", "hat"};
   1. The above syntax is correct. (But why tho?)
5. We cannot compare two strings using the == operator.
6. This is because when we declare a char \* variable to hold a string, we are actually storing the address of the first letter of that string.
7. So technically when we use the == operator, we are comparing the addresses of the first letters of both those strings.
8. In order to check equality, we need to use the strcmp function.

A general rule of thumb in programming is, if you need to store a number, but you will not be doing any math on it, then it is best to store it as a string.

1. 
2. The scanf function has the format specifier %5s. This means no matter how many letters the user inputs, only 5 or below will be accepted.
3. “Code Smell” – When something’s a little “off” about how the code is implemented.

Structures:

1. 
2. The above is the syntax for a structure.
3. The typedef keyword allows you to essentially create a custom data type.
4. So the struct keyword allows you to cluster all the data you want together, and the typedef keyword allows you to call that data cluster you just created with another name.

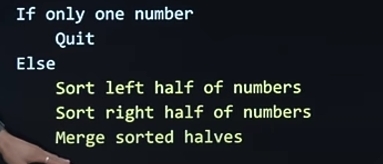
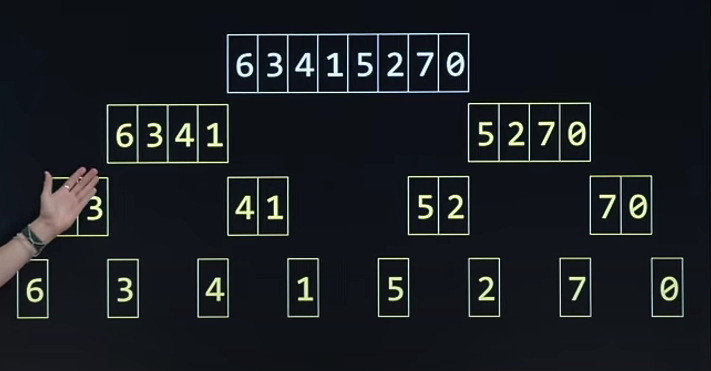
Sorting:

1. Selection sort:
   1. Find the smallest element in the array
   2. Put that in the 0th location, and put the element we originally had in the 0th location to where we found the smallest element.
   3. Since we found the smallest element, we no longer need to consider the 0th location, so increment the index by 1.
   4. TL;DR – Find the smallest element on each iteration, and put it at the ith location. i is initially 0, and we keep incrementing it till we reach array length.
   5. The worst case T.C is O(N^2). The best case is Ω(N^2), and theta (Θ) is also N^2
   6. We can kinda modify the selection sort algorithm depending on our needs. If it’s basic sorting look for the smallest element through each iteration. Or we can look for the biggest too. The point being, it is more flexible.
2. Bubble sort:
   1. Start at the 0th element (Lets call it i)
   2. Swap the ith and the (i + 1)th element in case you have to, otherwise move forward.
   3. Keep repeating this process till you reach the last element.
   4. Restart from the first element, and go till the second last element this time
   5. T.C = O(N^2).
   6. Best case = Ω(N) (In case of sorted list. In order to achieve this, we need to add a condition like if no swaps done, abort the algo.)

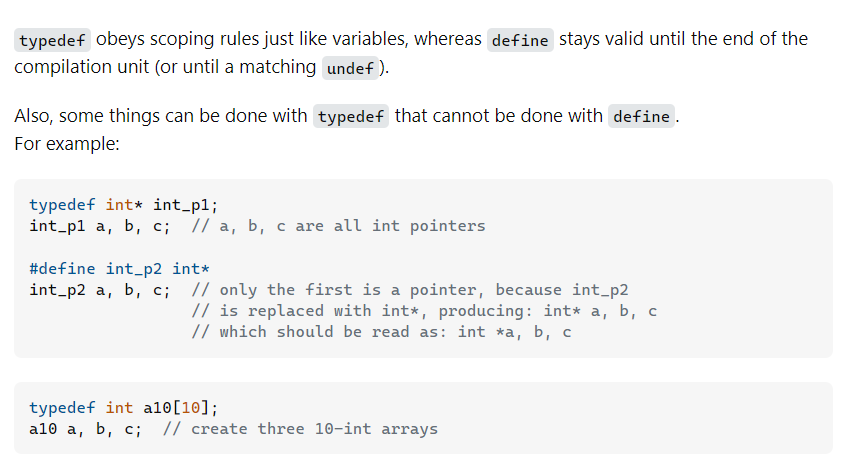
Recursion:

1. A function that calls itself is said to be recursive.
2. Recursive functions need a base case in order to break out of it. Otherwise it’ll go on till infinity.
3. The base case is hardcoded.
4. Recursion allows us to solve problems in a fundamentally new way, by allowing us to use the computers memory in a new way.

Merge Sort:

1. Selection sort and bubble sort are useful if we are allowed to use only constant amount of memory.
2. Merge sort performs better if we can use extra memory.
3. 
4. The above is the pseudocode for merge sort.
5. The time complexity is O(NlogN)
6. 
7. There are logN levels to this tree, and there are N steps at each level, hence, O(NlogN)
8. It is also Ω(NlogN)

Problem Set Notes:

1. You may find it helpful to time your sorts. To do so, run time ./[sort\_file] [text\_file.txt].
2. Sort1 is probably bubble sort, Sort2 is merge, and Sort3 is selection
3. The reason for the above claims are:
   1. Sort1 performance when we time it with random50000 is 6.542 seconds. However, when we time it with sorted50000, we get 2.015 seconds. No algorithm apart from this shows such a drastic difference in an average case scenario and the best case scenario. This could imply that Sort1 is bubble sort, since it’s best case is Ω(N) and average/worst case is O(N^2)
   2. Sort2 performance when we time it with random50000 is 1.434 seconds, and when we time it with sorted50000, we get 1.182 seconds. This is the fastest algorithm we’ve seen so far as the input gets bigger. And the best case performance and average case performance are fairly similar. For merge sort, best case is Ω(NlogN) and average/worst case is also O(NlogN)
4. typedef is used to create different TYPES (hence TYPE define), whereas define is used to define
5. #define is a preprocessor token: the compiler itself will never see it.
6. typedef is a compiler token: the preprocessor does not care about it.
7. 
8. The macro in C language is known as the piece of code which can be replaced by the macro value.
9. The macro is defined with the help of #define preprocessor directive and the macro doesn’t end with a semicolon(;).
10. Macro is just a name given to certain values or expressions it doesn't point to any memory location.
11. Whenever the compiler encounters the macro, it replaces the macro name with the macro value. Two macros could not have the same name.
12. The syntax of the macro is as shown in the following figure. Here, we will have the three components:
13. 