Algorithm Performed By Hand:

Target: 13

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	13	18	20	25	30	34	38	**[44]**	47	51	57	63	68	72	79	85	91	97

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	13	18	**[20]**	25	30	34	38	44	47	51	57	63	68	72	79	85	91	97

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	**[6]**	13	18	20	25	30	34	38	44	47	51	57	63	68	72	79	85	91	97

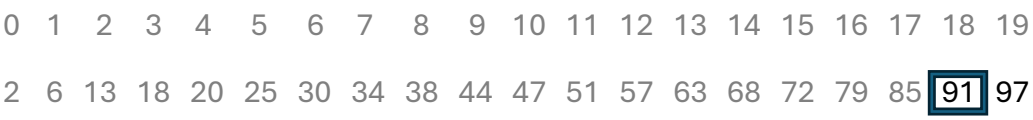
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	**[13]**	18	20	25	30	34	38	44	47	51	57	63	68	72	79	85	91	97

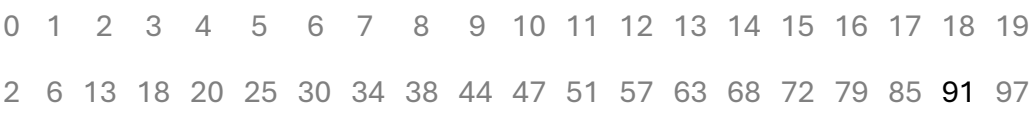
Target: 95

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	13	18	20	25	30	34	38	**[44]**	47	51	57	63	68	72	79	85	91	97

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	13	18	20	25	30	34	38	44	47	51	57	63	**[68]**	72	79	85	91	97

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	13	18	20	25	30	34	38	44	47	51	57	63	68	72	79	**[85]**	91	97





Problem Approach:

For this algorithm and assignment, we need to find an item in a list faster than we would by simply iterating through each item in the list. In order to do this, I will cut the list of potential items by half with each search through the list. I will start by setting variables to contain the starting index of the list and the end index of the list. From there, I will find the middle index by divide the distance between the start and end variables by 2 and save the value in a variable. I will then compare the value at that middle index to the value of the given target. If it matches, I will output the result. If the middle index is less than the target, the variable for the start index will be reset to the value of the middle index +1 while the variable for the end index will stay the same. If the middle index is greater than the target, the variable for the end index will be reset to the value of the middle index - 1 while the variable for the start index will stay the same. I will then divide the distance between them by two again to find the middle and repeat the process until the target is found or all possibilities are eliminated. If advanced search function does find the value, it will return true since the assignment does not have us list the index it was found at. If it does not find it, it will return false.

Pseudocode Attempt 1:

```
advanced_search( target, list)
        SET start <- 0
        SET end <- length(list)
        SET found <- False
        WHILE not found
                SET middle <- (start + end) / 2
```

```
        IF list[middle] == target
            SET found <- True
            RETURN True
        ELSE IF list[middle] < target
            SET start <- middle + 1
        ELSE
            SET end <- middle – 1
    RETURN False


main()
    GET list
    GET target
    SET result <- advanced_search(target, list)
    IF result
        PUT “We found {target} in the list”
    ELSE
        PUT “Target was not found”
```

Microsoft Copilot Pseudocode:

Me:

create a pseudocode solution to the following:

For this algorithm and assignment, we need to find an item in a list faster than we would by simply iterating through each item in the list. In order to do this, I will cut the list of potential items by half with each search through the list. I will start by setting variables to contain the starting index of the list and the end index of the list. From there, I will find the middle index by divide the distance between the start and end variables by 2 and save the value in a variable. I will then compare the value at that middle index to the value of the given target. If it matches, I will output the result. If the middle index is less than the target, the variable for

the start index will be reset to the value of the middle index +1 while the variable for the end index will stay the same. If the middle index is greater than the target, the variable for the end index will be reset to the value of the middle index - 1 while the variable for the start index will stay the same. I will then divide the distance between them by two again to find the middle and repeat the process until the target is found or all possibilities are eliminated. If advanced search function does find the value, it will return true since the assignment does not have us list the index it was found at. If it does not find it, it will return false.

Copilot:

Certainly! Let's create a pseudocode solution for a binary search algorithm. Binary search is an efficient way to find an item in a sorted list by repeatedly dividing the search space in half.

Here's the pseudocode for a basic binary search algorithm:

```
function binarySearch(arr, target):
    // Initialize the start and end indices
    start = 0
    end = length(arr) - 1

    while start <= end:
        // Calculate the middle index
        mid = (start + end) / 2

        // Compare the value at the middle index with the target
        if arr[mid] == target:
            return true // Target found
        elif arr[mid] < target:
            start = mid + 1 // Adjust the start index
        else:
            end = mid - 1 // Adjust the end index

    return false // Target not found
```

Explanation:

1. We start with the entire array (from index 0 to length(arr) - 1).
2. In each iteration, we calculate the middle index (mid) and compare the value at that index with the target.
3. If the value at mid is equal to the target, we return true.
4. If the value at mid is less than the target, we adjust the start index (start = mid + 1).
5. If the value at mid is greater than the target, we adjust the end index (end = mid - 1).
6. Repeat the process until start is greater than end.

Remember to adapt this pseudocode to the specific programming language you're using (e.g., C#, JavaScript, etc.). You'll need to implement the actual array and handle any edge cases. Let me know if you need further clarification or additional examples! 😊

Compare and Contrast:

1. Both of the pseudocode solutions had extremely similar ideas about how the logic of the algorithm should be executed. The one I created has more descriptive and better names for some of the variables, alongside using pseudocode keywords. My code also has a main function to get the list and target from the user. However, it does appear that I had a redundant variable ("found") which I have noticed could also lead to an infinite loop if something is not found to set the condition to true. My "end" variable would also initially be off by one because I forgot to consider that the list starts at index 0, so I should have subtracted 1 from the value for the length of the list. The Copilot pseudocode has the "end" variable properly calculated and it also has more simple logic for determining when the loop should end rather than trying to use multiple Booleans. Where it falls short is the more generic names.
2. I can improve my solution by taking what the Copilot pseudocode did correctly with the loop logic and variable calculations and implementing those into my pseudocode.
3. The Copilot pseudocode could be better improved based on what I know by using more keywords and more descriptive names for some of the variables and functions. It could also use a main function to get the list and target from the user.
4. The pseudocode from both previous steps do accomplish the task I performed by hand at the beginning of the assignment, although Copilot was more correct than my first attempt at writing the pseudocode for the algorithm. I have also noticed that the problem description for the next week includes getting the list from a JSON file, so I will include that logic in my updated pseudocode on the next page. I will also try and use more descriptive variable names and keywords than last time. In addition, a call to the main function and a specified end of program will be included.

Updated Pseudocode (Attempt 2):

```
advanced_search( target, list)

    SET start <- 0

    SET end <- length(list) - 1
```

```
        WHILE start <= end
                SET middle <- (start + end) / 2
                IF list[middle] == target
                        RETURN True
                ELSE IF list[middle] < target
                        SET start <- middle + 1
                ELSE
                        SET end <- middle – 1
        RETURN False


main()
        PROMPT for file_name
        OPEN file_name for reading
        READ search_array as a list of items from the file
        CLOSE file_name
        PROMPT for target
        SET result <- advanced_search(target, search_array)
        IF result
                PUT “We found {target} in the list”
        ELSE
                PUT “Target was not found”
main()
END
```