Jump Search

The jump search algorithm has characteristics in common with both linear and binary searches. Like a linear search, it moves in only one direction, from index 0 to the end of the array. Like a binary search, jump search works only with sorted arrays and checks values at array indexes to determine if the target could possibly exist in a subrange.

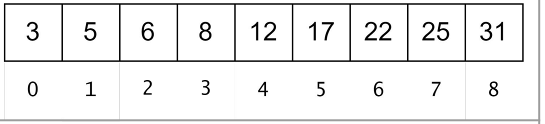
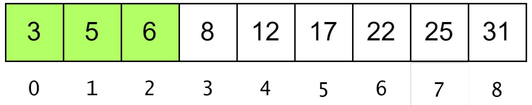
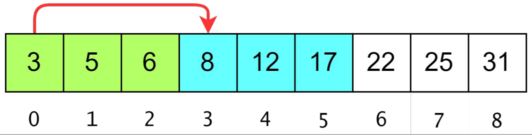
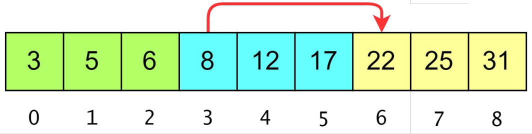
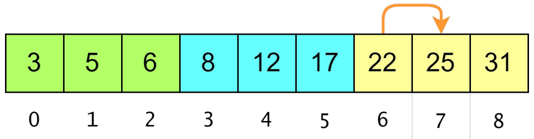
A jump search begins by computing a block size of n where n is the length of the array. The block size determines the number of indexes in a subrange. A block search is conducted to determine if a block *could possibly* contain the target value. If such a block is found, a linear search is performed on the indexes within the block.

For each block, the search checks the value at the last index in the block

- If the value is the target, it has been found and index is returned
- If the value is greater than the target, the target, if it exists, must be in this block
    - A linear search is then conducted from the starting index of the block to the next to last index in the block and returns the result of the linear search
- Otherwise, the value is less than the the target, the next block is searched

If the end of the array is reached and the target is not found, `None` is returned

Jump Search Example

Given the array to the right, let's search for a target of 25 The array's `length` is `9`, therefore the block size is √9 or `3`	
The first block to search is indexes `[0]` - `[2]` (inclusive)	
The last index in the block, `[2]`, has a value of `6`. `6` is not our target, `25`, and is also less than the target, so we jump to the next block of 3 indexes, `[3]` - `[5]` (inclusive)	
The last index in the block, `[5]`, has a value of `17`. `17` is not our target, `25`, and is also less than the target, so we jump to the next, and last, block of 3 indexes, `[6]` - `[8]` (inclusive)	
The value at the last index, [8], in the block, `31`, is not our target, `25`, but it is greater than the target, so *if* the target exists in the array, it must be in this block, indexes `[6]` - `[8]` (inclusive) A linear search begins from the start index of the block, `[6]` to the next to last index in the block, `[7]` The value at index `[6]`, `22`, does not match our target, so we check the next index The value at index `[7]`, `25` is our target, so it is found and we return our index `[7]`!!!	

Problem 1A

Given the following array and target, fill in the block size, block search values, and linear search values for each iteration

1	3	4	11	17	21	26	31	32	37	40	43	58	62	65	77	91
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Block Size: sqrt(17)

target = 91

Block Search					
iteration	0	1	2	3	4
block start	0	8	12	14	15
block end	16	16	16	16	16
value	32	58	65	77	91

Linear Search					
iteration	0	1	2	3	4
index	0	4	8	12	16
value	1	17	32	58	91

Problem 1B

Given the following array and target, fill in the block size, block search values, and linear search values for each iteration

1	3	4	11	17	21	26	31	32	37	40	43	58	62	65	77
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Block Size: sqrt(16)

target = 21

Block Search					
iteration	0	1	2	3	4
block start	0	0	4	4	
block end	15	8	8	6	
value	32	17	26	21	

Linear Search					
iteration	0	1	2	3	4
index	0	1	3	5	
value	1	3	11	21	

Problem 1C

Given the following array and target, fill in the block size, block search values, and linear search values for each iteration

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	3	4	11	17	21	26	31	32	37	40	43	58	62

Block Size: sqrt(14)

target = 5

Block Search					
iteration	0	1	2	3	4
block start	0	0	0	2	N/A
block end	13	7	4	3	N/A
value	31	17	4	11	N/A

Linear Search					
iteration	0	1	2	3	4
index	0	1	2	3	N/A
value	1	3	4	11	N/A

Problem 2

Together with your team, write the pseudocode (high-level steps) for the Jump Search Algorithm.

```
def jump_search(an_array, target):

    length = len(an_array)
    for index in range(length):
        value = an_array[index]
        block_size = math.sqrt(len(an_array))
        linear_search(an_array, target, start=None, stop=None)
        if value == target:
            return index
        if value != target:
            return None
        else:
            return "FAIL"
```

Problem 3

Referencing your pseudocode from the previous problem, calculate the expected time complexity of the Jump Search Algorithm.

Hint: break the analysis into two parts, Block Search and Linear Search, and combine the complexity of each.

The Expected Time Complexity Of The Jump Search Algorithm Is $O(\sqrt{n})$ And/Or $O(n^{1/2})$.

Problem 4

A test that is written *after the fact* to test code that is already working is called a *characterization test*. We use the test to characterize the current behavior of the code before making modifications so that we can make the modifications safely.

For the linear search that we wrote in class, list four test conditions that you would write characterization tests for. Do not write the actual tests.

Example: Empty array
1. Test Linear Search Correct
2. Test Linear Search Incorrect
3. Test Linear Search Start/Stop
4. Test Linear Search Array/Target

Problem Solving 5

More Binary Search practice. Given the following array and targets, fill in the start, end, mid, and values for each iteration

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	8	12	14	17	45	46	55	57	59	60	61	70	79	92	97	99

target = 99							
iteration	0	1	2	3	4	5	6
start	0	8	12	14	15		
end	16	16	16	16	16		
mid	8	12	14	15	16		
value	57	70	92	97	99		

target = 2							
iteration	0	1	2	3	4	5	6
start	0	0	0	0	0	N/A	
end	16	8	4	2	1	N/A	
mid	8	4	2	1	0	N/A	
value	57	17	12	8	3	N/A	