Nlam57
251164861
Asn1 Electronic Typed Answers
September 25, 2021

1.
We must find constants $c > 0$ and $n >= 1$ such that

$1/n^2 <= c\ 1/n$

$0 <= c\ 1/n - 1/n^2$ for all $n >= n0$

$0 <= 1/n(c - 1/n)$ for all $n >= n0$

$1/n <= c$ for all $n >= n0$

Therefore when c is 1 and n0 is 1 this equality holds true for all $n >= n0$

2.
We must find constants $c > 0$ and $n >= 1$ such that

$f(n)$ is $O(g(n))$

This means that there exist constants $c > 0$ nd $n >= 1$ such that

$f(n) <= cg(n)$ for all $n >= n0$

both are positive, so we can multiply both sides by 1/f(n)*g*(n)

$f(n)/f(n)*g(n) <= g(n) / f(n)*g(n)$ for all $n >= n0$

$1/g(n) <= c\ 1/f(n)$ for all $n >= n0$

Therefore, when $c = 1$ and $n0 = 1$ the inequality holds true for all $n >= n0$

3.

We must find constants $c > 0$ and $n >= 1$ such that $n - 1$ is $O(1)$

To prove by contradiction we will assume that $n - 1$ is $O(1)$ is true and derive the contradiction

If $n - 1$ is $O(1)$ then there are constants $c > 0$ and $n0 >= 1$ such that:

n - 1 <= c(1) for all n >= n0

n <= c - 1 for all n >= n0

Therefore, this cannot be true because c is a constant whereas n can grow without bounds, so n-1 is not O(n).

4.
Done → MostTimes.java submission

5.
The algorithm does not terminate. The example I will show is in the case we have an array L with n % 2 = 0 with x = L[-1]. The problem with the algorithm is the second if statement. The reason we increment i by 2 in the else statement is to check through all the even indexes in the array for x, then in the second if statement we set i to 1 when we are “done with all the even indexes” to then start checking the odd ones in hopes to find x. The problem with this is that if the element is at the last slot in the array if n is even then i will be set back to 1 because it is equal to n - 1 and we compare L[i] which is L[1] with x on the current loop/value of i. In the case where n is odd, if it is the last element in the array the same issue occurs, i gets set to 1 and then we compare L[i] which is L[1] with x.

Lets say x that we are looking for is the last element in the array of and even number of elements, take the array: {4, 5, 6, 7, 8, 9}, look at the table below if we are looking for x = 9:

Iteration	**Statement**	**i**	**L[i] in this iteration**
1	if L[i] = x	0	4
	else do i += 2	2	4
	if i >= n - 1	Not Yet	4
2	if L[i] = x	2	6
	else do i += 2	4	6
	if i >= n - 1	Not Yet	6
3	if L[i] = x	4	8
	else do i += 2	6	8

	if i >= n - 1	True	8
4	if L[i] = x	1	5
	else do i += 2	3	5
	if i >= n - 1	Not Yet	5
5	if L[i] = x	3	7
	else do i += 2	5	7
	if i >= n - 1	Yes and x is currently L[i] after the incrementation in the else statement but is now set to 1 because i is n - 1 currently, one more loop just to make sure	7
6	if L[i] = x	1	5
	else do i += 2	3	5
	if i >= n - 1		5
7	if L[i] = x	3	7
	else do i += 2	5	7
	if i >= n - 1	Yes and x is currently in index but this will repeat infinitely	7

What I am trying to explain is that i is never allowed to get to the last element in the array because every time it does, the loop is started again with i = 1. To fix this if the second if statement was if i > n. Also in the scenario that x is not in the array, this will cause an infinite loop because i is always being incremented and reset to 1 when we reach the end until x is found, and if x cannot be found in the first and second loop, each loop after that will only check the odd indexes of the array which did not find x in the first place, thus the loop will be infinite unless there is a condition or a counter that stops i from incrementing and exits the loop.

I coded this out in the IDE and have the solution with my suggested correction, happy to provide it as context. However, my response is that the algorithm does not ALWAYS terminate in the case where x is at the last element in the array L or x is not in the array.

6.
This algorithm always terminates because we use variable j to be incremented and reset to try to find x, however, each time we loop through the array i is incremented by 1, and so when we loop through the array n - 1 times, the loop will terminate and return -1 because x was not found.

This algorithm does not produce the correct output for any L with x as its last element. Like the algorithm in question 5, we are still checking the even indexes for x first then we are checking the odd ones. However, when j gets incremented to the last element in the

array (j = n - 1) where x lies after the else statement, the last if statement does not allow us to check that position because it sets j to 1 starting the loop again from one, incrementing i and therefore this will repeat until i = n and the loop terminates returning -1 because we never checked the last element so we could not find x.

These are proved using the example I gave in question 5. The loop terminates when i is incremented to 6 after the 5th loop → always terminates.

Also when x is the last element in the array, when j goes from 0 → 2 → 4 → 1 → 3 → 5 we do not start a loop with j = 5 because 5 is n - 1 so j becomes 1 and this repeats until i >= n.

Let me display this in a table like the one above:
Here is the input we will use:

```
Run | Debug
public static void main(String[] args) {

    Question_6 q6 = new Question_6();

    int[] L = {4, 5, 6, 7, 8, 9};

    int x = 9;
    int y = L.length;

    System.out.println(q6.search(L, y, x));
}
```

Iteration	Statement	i, j	L[j] in this iteration
1	if L[j] == x	0, 0	4
	else j += 2	0, 2	4
	if j >= n - 1, j = 1, i += 1	Not yet	
2	if L[j] == x	0, 2	6
	else j += 2	0, 4	6
	if j >= n - 1, j = 1, i += 1	Not yet	
3	if L[j] == x	0, 4	8

	else j += 2	0, 6	8
	if j >= n - 1, j = 1, i += 1	Yes, j > n so j = 1 and loop is restarted	
4	if L[j] == x	1, 1	5
	else j += 2	1, 3	5
	if j >= n - 1, j = 1, i += 1	Not Yet	
5	if L[j] == x	1, 3	7
	else j += 2	1, 5	7
	if j >= n - 1, j = 1, i += 1	Not Yet	
6	if L[j] == x	2, 5	9
	else j += 2	2, 5	9
	if j >= n - 1, j = 1, i += 1	Yes, we found x, however, j > n so j = 1 and we restart the loop, however, since i = n - 1 and n is 6 so the loop is restarted with j = 1. This process is repeated until i = 5 which is n - 1 which is when the while loop terminates and it returns -1 because the last element was reached through loops 2 - 5 however was never searched. This proves that the algorithm does not always return the correct output.	

7.

n	Linear Search
5	111 ns
10	179 ns
100	556 ns
1000	4748 ns
10000	7471 ns
100000	25140 ns

n	Quadratic Search
5	259 ns
10	482 ns
100	7966 ns
1000	117000 ns
10000	9594228 ns

n	Factorial Search
7	2122357 ns
8	14425270 ns
9	62173658 ns
10	642973269 ns
11	6610770884 ns
12	121421444291 ns