Started on	Wednesday, 4 December 2024, 2:15 PM
State	Finished
Completed on	Wednesday, 4 December 2024, 2:49 PM
Time taken	34 mins 10 secs
Marks	5.00/5.00
Grade	**100.00** out of 100.00

Information

Consider the following algorithm:

```
procedure Algorithm1(list)
    let n be the length of the list
    for i = 0 to n-1 do
        for j = 0 to n-1 do
            if i != j and list[i] == list[j] then
                return true
            end if
        end for
    end for
    return false
end procedure
```

Question 1

Correct

Mark 1.00 out of 1.00

What task does the algorithm perform?

Select one:

[✓ a. Returns true if the list contains any duplicate elements, false otherwise] ✓

b. Returns an element that appears in both of two lists

c. Returns an element that appears more than once in the list

d. Returns true if two lists are equal, false otherwise

Your answer is correct.

The correct answer is: Returns true if the list contains any duplicate elements, false otherwise

Question 2

Complete

Not graded

Explain why the worst-case running time of the algorithm is quadratic, i.e. O(n^2).

this algorithm is quadratic because for every time the outer loop (for i = 0 to n-1) runs the inner loop (for j = 0 to n-1) runs as well. this is n * n = n^2

Information

Now suppose the algorithm is changed as follows:

```
procedure Algorithm2(list)
    let n be the length of the list
    for i = 0 to n-1 do
        for j = 0 to i-1 do  // n-1 has been changed to i-1 here
            if i != j and list[i] == list[j] then
                return true
            end if
        end for
    end for
    return false
end procedure
```

Question 3

Complete

Not graded

Explain why the algorithm is still correct (i.e. still carries out the task you identified in question 1).

it still checks for duplicate pairs but in this instance it does it with out checking the same pair twice. it still loops through every pair and returns true if there is a pair.

Question 4

Complete

Not graded

Explain why this algorithm runs approximately twice as fast (in the worst case) as the previous algorithm.

in this instance it doesn't check the pairs twice. in the last algorithm the pairs for example could be i=1 and j= 3 but then it would also check i=3 and j=1. however in this instance of the algorithm it doesn't do that as it only check each pair once.

Question 5

Correct

Mark 1.00 out of 1.00

Is the time complexity of this algorithm still quadratic?

Select one:

[✓ a. Yes] ✓

b. No

Your answer is correct.

The correct answer is: Yes

Question 6

Complete

Not graded

Explain why the time complexity is / isn't quadratic.

it is still quadratic because for every loop of the outer loop the inner loop also does a loop. this means that it is still O(N^2)

Information

Now let's assume the list contains items that can be sorted in order (e.g. numbers), and consider the following algorithm which performs the same task again:

```
procedure Algorithm3(list)
    let n be the length of the list
    let sortedList = Sort(list)
    for i = 1 to n-1 do
        if sortedList[i-1] == sortedList[i] then
            return true
        end if
    end for
    return false
end procedure
```

Question 7

Complete

Not graded

Explain why this algorithm only needs to check consecutive elements of the list, rather than checking every possible pair.

as the list is sorted you only have check consecutive pairs because if there was a duplicate it would appear right next to itself in a sorted list.

Question 8

Correct

Mark 1.00 out of 1.00

What is the time complexity of Python's built-in **sort** function?

Hint: you will need to do some online research to answer this question.

Select one:

- a. O(n^2)
- b. O(1)
- c. O(n)
- [✓ d. O(n log n)] ✓
- e. O(log n)

Your answer is correct.

The correct answer is: O(n log n)

Question 9

Correct

Mark 1.00 out of 1.00

Therefore what is the overall time complexity of the algorithm above, if it is implemented in Python using the built-in **sort** function?

Select one:

- a. O(1)
- [✓ b. O(*n* log *n*)] ✓
- c. O(log *n*)
- d. O(n^2)
- e. O(*n*)

Your answer is correct.

The correct answer is: O(*n* log *n*)

Question 10

Complete

Not graded

Explain your answer to question 9.

well the built in sort function has a time complexity of O(N log N) and the time complexity of the iteration appears to be O(N) thus as O(N log N) is bigger i believe it would be O(N log N).

Information

Recall the two algorithms we have seen so far:

```
procedure Algorithm2(list)
    let n be the length of the list
    for i = 0 to n-1 do
        for j = 0 to i-1 do  // n-1 has been changed to i-1 here
            if i != j and list[i] == list[j] then
                return true
            end if
        end for
    end for
    return false
end procedure
```

```
procedure Algorithm3(list)
    let n be the length of the list
    let sortedList = Sort(list)
    for i = 1 to n-1 do
        if sortedList[i-1] == sortedList[i] then
            return true
        end if
    end for
    return false
end procedure
```

Question 11

Correct

Mark 1.00 out of 1.00

If the size of the input list is very large, which algorithm is likely to run faster?

Select one:

a. Algorithm2

[✓ b. Algorithm3] ✓

Your answer is correct.

The correct answer is: Algorithm3

Question 12

Complete

Not graded

Explain your answer to question 11.

algorithm3 with a time complexity of O(N log N) will perform the sort faster than algorithm2 with a time complexity of O(n^2) on larger list as O(N log N) is smaller of a time complexity than O(N^2)

Question 13

Complete

Not graded

Suggest **one** reason why a programmer might choose the "slower" algorithm (i.e. the algorithm you **did not** choose in question 11) over the "faster" one.

they might chose to use the slower algorithm2 because they it doesn't rely on the inbuilt sort function and thus does not need extra memory to perform the given task and they may be low on memory.

◄ Worksheet 3 Brief

Jump to...

Worksheet 3 Template Repo ►