CS 5012 Module 1 ICA

Alanna Hazlett

uwa6xv

September 4th, 2024

In class problems:

In [ ]:

```
def findMax(arr):                   1 opertation
    max_val = arr[0]                run len(arr) times
    for num in arr:                 1 operation
        if num > max_val:           1 operation
            max_val = num           1 operation
        return max_val              1 operation

1 + len(arr) * (1+1) + 1 -> 2*len(arr) + 2.
This simplifies to len(arr) = n, so in big O notation it is O(n).


def findMax(arr):
    max_val = arr[0]                1 operation
    for num in range(5):            run 5 times
        if num > max_val:           1 operation
            max_val = num           1 operation
    return max_val                  1 operation

1 + 5 * (1+1) + 1 = 12
This is O(1).


def findMax(arr):
    max_val = arr[0]                            1 operation
    for num in arr:                             run len(arr) times
        max_val = complex_function(arr)         complex_function(arr)
    return max_val                              1 operation

Unknown number of operations/kinds in complex_function,
so we just include it in our answer,
whether it is multiplication or addition.
1 + len(arr) * complex_function(arr) + 1
len(arr) * complex_function(arr) + 2
len(arr) * complex_function(arr)
n = len(arr).
This simplifies to O(n * complex_function(arr)).


def findMax(arr):
    max_val = arr[0]                              1 operation
    b = complex_function(arr)                     complex_function(arr)
```

```
    for num in arr:                             run len(arr) times
        max_val = num                           1 operation
    return max_val                              1 operation
1 + complex_function(arr) + len(arr) * 1 + 1
len(arr) + complex_function(arr) + 2
len(arr) + complex_function(arr)
n = len(arr).
This simplifies to O(n + complex_function(arr)).


def findMax(arr1, arr2):
    max_val = arr[0]                1 operation
    for num in arr1:                run len(arr1) times
        max_val = num               1 operation
    for num in arr2:                run len(arr2) times
        max_val = num               1 operation
    return max_val                  1 operation
We do not know which array is longer or shorter and they could change,
so we include both in our big O notation.
1 + len(arr1) * 1 + len(arr2) * 1 + 1
len(arr1) + len(arr2)
n = len(arr1)
m = len(arr2)
This simplifies to O(n+m).


def findMax(arr):
    max_val = arr[0]                 1 operation
    for ii in arr:                   run len(arr) times
        for jj in arr:               run len(arr) times
            if num > max_val:        1 operation
                max_val = num        1 operation
    return max_val                   1 operation

1 + len(arr) * len(arr) * (1+1) + 1
len(arr) * len(arr) * 2 + 2
len(arr) * len(arr)
n = len(arr)
This simplifies to O(n^2).


What circumstances do you come across an exponential complexity?
Optimization problems.

When do you come across O(log n)?
Binary search, the search space is halving in each step.
```

Breakout group problems

Problem 6: Calculating the Factorial of a Number

In [ ]:

```
def factorial(n):
    if n == 0:
        return 1
    else:
```

```
        return n * factorial (n-1)

This function is a O(n) function.
n = the number of times the function runs,
since it is a recursive function.
```

Problem 7: Checking for Duplicates in an Array

In [ ]:
```
def checkDuplicates(arr):
    for i in range(len(arr)):              run len(arr) times
        for j in range(i + 1, len(arr)):   run len(arr) times
            if arr[i] == arr[j]:           1 operation
                return True                1 operation
    return False                           1 operation

len(arr) * len(arr) * 1 * 1 + 1
len(arr) * len(arr)
n = len(arr)
This simplifies to O(n^2).
```

Problem 8: Linear Search Algorithm

In [ ]:
```
def linearSearch(arr, x):
    for i in range(len(arr)):               run len(arr) times
        if arr[i] == x:                     1 operation
            return i                        1 operation
    return -1                               1 operation

len(arr) * 1 * 1 + 1
len(arr)
n = len(arr)
This simplifies to O(n).
```

Problem 9: Merging Two Sorted Arrays

In [ ]:
```
def mergeArrays(arr1, arr2):
    result = []
    i, j = 0, 0
    while i < len(arr1) and j<len(arr2):     run len(shorter array) times
        if arr1[i]<arr2[j]:                  1 operation
            result.append(arr1[i])           1 operation
            i += 1                           1 operation
        else:                                1 operation
            result.append(arr2[j])           1 operation
            j += 1                           1 operation
    result += arr1[i:] + arr2[j:]            1 operation
    return result                            1 operation

len(shorter array) * 1 * (1+1) + 1 * (1+1) + 1 + 1
len(shorter array)
We do not know which array will be shorter,
so we must include both in our answer.
n = len(arr1)
```

```
m = len(arr2)
This simplifies to O(n+m).
```

Problem 10: Finding Unique Elements in an Array

In [ ]:
```
def findUnique(arr):
    unique_elements = set()             1 operation
    for num in arr:                     run len(arr) times
        unique_elements.add(num)        1 operation
    return list(unique_elements)        1 operation

1 operation + len(arr) * 1 + 1
len(arr)
n = len(arr)
This simplifies to O(n).
```

I pledge that I have neither given nor received help on this assignment. : Alanna Hazlett