
DSC 40B - Midterm Review

Problem 1.

What is the time complexity of this piece of code?

```
def factorial(n):  
    r = 1  
    i = n  
    while i > 0:  
        r = r*i  
        i -= 1  
    return r
```

- ☐ $\theta(n^2)$
- ☐ $\theta(n^3)$
- ☐ $\theta(n \log n)$
- ☐ $\theta(n)$

Solution: $\theta(n)$

Problem 2.

What is the time complexity of this piece of code?

```
def foo(n):  
    for i in range(5, 10):  
        for j in range(i):  
            for k in range(n):  
                print(i, j, k)
```

- ☐ $\theta(n^2)$
- ☐ $\theta(n^3)$
- ☐ $\theta(n \log n)$
- ☐ $\theta(n)$

Solution: $\theta(n)$

The two outer loops run for only a constant number of iterations and the inner most loop runs for $\theta(n)$ iterations. Therefore, the total time complexity is $\theta(n)$.

Problem 3.

This piece of code returns the number of "pairs" of the form (x, -x) in a collection of numbers. What is the time complexity of this piece of code if the input is a Python list of size n?

```
def count_pairs(numbers):  
    count = 0  
    for x in numbers:  
        if -x in numbers:
```

```
        count += 1/2
    return count
```

- ☐ $\theta(n^2)$
- ☐ $\theta(n^3)$
- ☐ $\theta(n \log n)$
- ☐ $\theta(n)$

Solution: $\theta(n^2)$

Checking if an element is in the list using the Python **in** operator takes time linear in the size of the list. Therefore, the if statement in the code takes $\theta(n)$ time. The for loop executes for $\theta(n)$ iterations. Hence, the total time complexity is $\theta(n^2)$.

Problem 4.

The below code shows the iterative version of binary search.

```
def binary_search(arr, t, start, stop):
    while start < stop:
        middle = start + (stop - start) // 2
        if arr[middle] == t:
            return middle
        if arr[middle] < t:
            start = middle + 1
        else:
            stop = middle
```

Let $n = \text{stop} - \text{start}$. What is the worst-case time complexity of this version of binary search?

- ☐ $\theta(n^2)$
- ☐ $\theta(\log n)$
- ☐ $\theta(n \log n)$
- ☐ $\theta(n)$

Solution: $\theta(\log n)$

Problem 5.

Here is a recursive algorithm for computing the factorial of n :

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

What is the recurrence relation describing this function's run time?

Solution: $T(n) = 1 + T(n-1)$

Problem 6.

Suppose $f_1(n) = \Theta(n^3)$ and $f_2(n) = \Omega(n)$. Which is true about the upper bound of $f_1 + f_2$?

- ☐ $f_1(n) + f_2(n) = O(n)$
- ☐ It cannot be determined
- ☐ $f_1(n) + f_2(n) = O(n^3)$

Solution: It cannot be determined

Problem 7.

Suppose $f_1(n) = \Omega(n^2)$ and $O(n^5)$ and $f_2(n) = \Omega(n^3)$ and $O(n^6)$. Which is true about $f_1 + f_2$?

- ☐ It is $O(n^5)$ and $\Omega(n^2)$
- ☐ It is $O(n^6)$ and $\Omega(n^3)$
- ☐ It is $O(n^6)$ and $\Omega(n^2)$
- ☐ It is $O(n^5)$ and $\Omega(n^3)$

Solution: It is $O(n^6)$ and $\Omega(n^3)$

Problem 8.

If $f(n) = O(n^2)$, then $f(n) = \Omega(n^2)$

- ☐ True
- ☐ False

Solution: False

Assume $f(n) = n$. In this case, $f(n) = O(n^2)$, but $f(n) \neq \Omega(n^2)$

Problem 9.

If $f(n) = O(n^5)$, and $g(n) = O(n^2)$ then $f(n)/g(n) = O(n^3)$

- ☐ True
- ☐ False

Solution: False

Assume $f(n) = n^5$ and $g(n) = 1$. In this case, $f(n) = O(n^5)$, and $g(n) = O(n^2)$.
But $f(n)/g(n) = n^5 \neq O(n^3)$.

Problem 10.

The best case and worst case time complexity of merge sort is $\theta(n \log n)$

- ☐ True
- ☐ False

Solution: True

Problem 11.

The recursive calls made by mergesort are always on arrays of strictly smaller size than the input array.

- ☐ True
- ☐ False

Solution: True

Problem 12.

Consider the modified mergesort given below:

```
def mergesort(arr):
    if len(arr)>1 :
        middle=math.floor(len(arr)/2)
        left=arr[:middle]
        right=arr[middle:]
        for i in range(len(arr)):
            for j in range(len(arr)):
                print("Mergesort")
        mergesort(left)
        mergesort(right)
        merge(left, right, arr)
```

What is the time complexity of this modified mergesort?

- ☐ $\theta(n^2)$
- ☐ $\theta(n^3)$
- ☐ $\theta(n \log n)$
- ☐ $\theta(n)$

Solution: $\theta(n^2)$

The recurrence relation for the modified mergesort is $T(n) = n^2 + 2T(n/2)$. Let's try to solve the recurrence by unrolling the loop.

$$\begin{aligned}
 T(n) &= 2T(n/2) + n^2 \\
 &= 2[2T(n/4) + (\frac{n}{2})^2] + n^2 \\
 &= 4T(n/4) + 2(\frac{n^2}{2^2}) + n^2 \\
 &= 4T(n/4) + n^2(1 + \frac{1}{2}) \\
 &= 4[2T(n/8) + (\frac{n}{4})^2] + n^2(1 + \frac{1}{2}) \\
 &= 8T(n/8) + 4(\frac{n^2}{4^2}) + n^2(1 + \frac{1}{2}) \\
 &= 8T(n/8) + n^2(1 + \frac{1}{2} + \frac{1}{4})
 \end{aligned}$$

At k-th step, we have $T(n) = 2^k T(n/2^k) + n^2(1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^{k-1}})$

Base case is $T(1)$.

$$n = 2^k$$

$$\Rightarrow \log n = k$$

Substituting the value of k, we get

$$T(n) = 2^{\log n} T(1) + n^2(1 + 1/2 + \dots)$$

$$= \theta(n^2)$$

Problem 13.

Given n point in R^d , what is the time complexity for computing the angle between each of the vectors using dot product?

- ☐ $\theta(n^2)$
- ☐ $\theta(nd)$
- ☐ $\theta(n^2d)$
- ☐ $\theta(d)$

Solution: $\theta(n^2d)$

Computing the dot product of two d dimensional vectors take $\theta(d)$ time. Given n points, we have $\binom{n}{2} = \theta(n^2)$ pairs. Therefore, total time complexity is $\theta(n^2d)$.

Problem 14.

Suppose you are given n numbers in a Python list in sorted order. Describe an efficient algorithm for checking to see if there is any number in the list which occurs 42 times. Do not use a dictionary/hash map.

Solution: Iterate over the elements in the list at indices from 0 to len(list) - 42 using a for loop. For each index i, check if the element at i+41 is equal to the element at i. If yes, return the element.