PSO 1

Introductions

← YOUR PSO GTA (Justin)

YOUR PSO UTAs
Srushti
Siddarth

Some other things about this PSO (and me)

- Usually on slides
- I try to cover everything
- Raise your hand whenever
- 3-time 251 TA, 10-time overall TA

I upload my slides on my website!
justin-zhang.com/teaching/CS251

Question 1

Let c be the cost of calling the function WORK. That is, the cost of the function is constant, regardless of the input value. Determine the respective closed-form $T(n)$ for the cost of calling WORK.

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i^2 do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

What is this question asking?

Question 1

Let c be the cost of calling the function WORK. That is, the cost of the function is constant, regardless of the input value. Determine the respective closed-form $T(n)$ for the cost of calling WORK.

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i^2 do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

What is this question asking? Determine the respective closed-form $T(n)$ for the cost of calling WORK.

What is this question **NOT** asking?

Something with a $\sum$ or $\prod$ in it

An asymptotic answer

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i^2 do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

(Let c be the cost of calling Work)

How to derive T(n) when there are weird loops that are nested

1. Write out the general form of T(n).

<u>General idea</u>

loop == sum

nested loop == nested sum

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i^2 do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

(Let c be the cost of calling Work)

How to derive T(n) when there are weird loops that are nested

1. Write out the general form of T(n).

$$\sum_{i=\_}^{\_} \sum_{j=\_}^{\_} c$$

i	1	3	9	…	n
j					

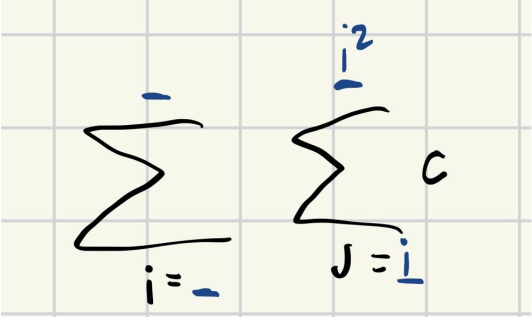
2. Write out the values of i and j as the loop iterates

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i² do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

(Let c be the cost of calling Work)

How to derive T(n) when there are weird loops that are nested

1. Write out the general form of T(n).



i	1	3	9	…	n
J (i to i²)	1 to 1	3 to 9	9 to 81		n to n^2

2. Write out the values of i and j as the loop iterates

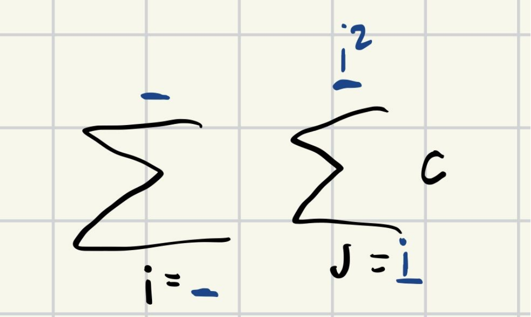
3. Plug in the start and ending values for i and j

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i^2 do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

(Let c be the cost of calling Work)

How to derive T(n) when there are weird loops that are nested

1. Write out the general form of T(n).



i	1	3	9	…	n
J (i to i²)	1 to 1	3 to 9	9 to 81		n to n^2

2. Write out the values of i and j as the loop iterates

3. Plug in the start and ending values for i and j   Problem: summations don't "multiply by 3"

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i^2 do
            val ← val + Work(n)
        end for
    end for
    return val
end function
```

Problem: summations don't "multiply by 3"

Solution: introduce a new variable k that iterates 'normally'

k	0	1	2	…	?
i	1	3	9	…	n
j (i to i^2)	1 to 1	3 to 9	9 to 81		n to n^2

Next step: write i in terms of k

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i^2 do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

Problem: summations don't "multiply by 3"

Solution: introduce a new variable k that iterates 'normally'

k	0	1	2	…	$\log_3 n$
i	3^0	3^1	3^2	…	3^k
j	1 to 1	3 to 9	9 to 81		n to n^2

Now write j in terms of k

```
function A1(n : Z+)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i^2 do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

Problem: summations don't "multiply by 3"

Solution: introduce a new variable k that iterates 'normally'

k	0	1	2	…	$\log_3 n$
i	**3^0**	**3^1**	**3^2**	**…**	**3^k**
j	**3^0** to **3^0**	**3^1** to **3^2**	**3^2** to **3^4**		**3^k** to **3^{2k}**

The corresponding sum is then..

```
function A1(n : Z⁺)
    val ← 0
    for i from 1 to n by multiplying by 3 do
        for j from i to i² do
            val ← val + WORK(n)
        end for
    end for
    return val
end function
```

Problem: summations don't "multiply by 3"

Solution: introduce a new variable k that iterates 'normally'

k	0	1	2	…	$\log_3 n$
i	3^0	3^1	3^2	…	3^k
j	3^0 to 3^0	3^1 to 3^2	3^2 to 3^4		3^k to 3^{2k}

The corresponding sum is then..

Last Steps: Solve the Sum (Carefully!!)

$$\sum_{k=0}^{\log_3 n} \sum_{j=3^k}^{3^{2k}} c$$

Pro technique:

Last Steps: Solve the Sum (Carefully!!)

$$\sum_{k=0}^{\log_3 n} \sum_{j=3^k}^{3^{2k}} c = \frac{9}{8}cn^2 - \frac{3}{2}cn + \left(\frac{11}{8} + \log_3 n\right)c$$

Pro technique: *Left as an exercise to the reader*

Question 2

Derive the closed-form $T(n)$ for the value returned by the following algorithm:

```
function A2(n : Z+)
    sum ← 0
    for i from 0 to n^4 − 1 do
        for j from i to n^3 − 1 do
            sum ← sum + 1
        end for
    end for
    return sum
end function
```

Let's follow the same steps

1. Write out the general form of T(n).
2. Write out the values of i and j as the loop iterates

Question 2

Derive the closed-form $T(n)$ for the value returned by the following algorithm:

```
function A2(n : Z+)
    sum ← 0
    for i from 0 to n^4 − 1 do
        for j from i to n^3 − 1 do
            sum ← sum + 1
        end for
    end for
    return sum
end function
```

1. Write out the general form of T(n).
2. Write out the values of i and j as the loop iterates

$$\sum_{i=0}^{n^4-1} \sum_{j=i}^{n^3-1} 1$$

Simple! (Fishy..)

Question 2

Derive the closed-form $T(n)$ for the value returned by the following algorithm:

```
function A2(n : ℤ⁺)
    sum ← 0
    for i from 0 to n⁴ − 1 do
        for j from i to n³ − 1 do
            sum ← sum + 1
        end for
    end for
    return sum
end function
```

1. Write out the general form of T(n).
2. Write out the values of i and j as the loop iterates

$$\sum_{i=0}^{n^4-1} \sum_{j=i}^{n^3-1} 1$$

When i > n^3-1, inner loop does not run so this sum is wrong!

Simple! (Fishy..)

Question 2

Derive the closed-form $T(n)$ for the value returned by the following algorithm:

```
function A2(n : Z+)
    sum ← 0
    for i from 0 to n^4 − 1 do
        for j from i to n^3 − 1 do
            sum ← sum + 1
        end for
    end for
    return sum
end function
```

1. Write out the general form of T(n).
2. Write out the values of i and j as the loop iterates

n^3-1 ~~n^4-1~~

$$\sum_{i=0}^{n^3-1} \sum_{j=i}^{n^3-1} 1$$

The right sum

Last Steps: Solve the Sum (Carefully!!)

$$\sum_{i=0}^{\overset{n^3-1}{\xcancel{n^4-1}}} \; \sum_{j=i}^{n^3-1} 1$$

Question 3

(a) The following statements are true or false?

1. $n^2 \in \mathcal{O}(5^{\log n})$
2. $\frac{\log n}{\log \log n} \in \mathcal{O}(\sqrt{\log n})$
3. $n^{\log n} \in \Omega(n!)$

(b) Sort the following functions in increasing order of asymptotic (big-O) complexity:

$$f_1(n) = n^{\sqrt{n}}, \quad f_2(n) = 2^n, \quad f_3(n) = n^{10} \cdot 2^{n/2}, \quad f_4(n) = \binom{n}{2}$$

What does f(n) = O(g(n)) mean (in words)?

Question 3

(a) The following statements are true or false?

1. $n^2 \in \mathcal{O}(5^{\log n})$
2. $\frac{\log n}{\log \log n} \in \mathcal{O}(\sqrt{\log n})$
3. $n^{\log n} \in \Omega(n!)$

(b) Sort the following functions in increasing order of asymptotic (big-O) complexity:

$$f_1(n) = n^{\sqrt{n}}, \quad f_2(n) = 2^n, \quad f_3(n) = n^{10} \cdot 2^{n/2}, \quad f_4(n) = \binom{n}{2}$$

What does f(n) = O(g(n)) mean (in words)? *f(n) is upper-bounded by g(n), asymptotically*

What does f(n) = O(g(n)) *actually* mean?

Question 3

(a) The following statements are true or false?

1. $n^2 \in \mathcal{O}(5^{\log n})$
2. $\frac{\log n}{\log \log n} \in \mathcal{O}(\sqrt{\log n})$
3. $n^{\log n} \in \Omega(n!)$

(b) Sort the following functions in increasing order of asymptotic (big-O) complexity:

$$f_1(n) = n^{\sqrt{n}}, \quad f_2(n) = 2^n, \quad f_3(n) = n^{10} \cdot 2^{n/2}, \quad f_4(n) = \binom{n}{2}$$

What does f(n) = O(g(n)) mean (in words)? *f(n) is upper-bounded by g(n), asymptotically*

What does f(n) = O(g(n)) *actually* mean?

There exists constants $c > 0, n_0 \in \mathbb{N}$ such that

$c f(n) \leq g(n)$

for all $n \geq n_0$

(a) The following statements are true or false?

1. $n^2 \in \mathcal{O}(5^{\log n})$

(Suppose log is base 2)

(For general log base b, this is only true when..)

(a) The following statements are true or false?

2. $\frac{\log n}{\log \log n} \in \mathcal{O}(\sqrt{\log n})$

(a) The following statements are true or false?

3. $n^{\log n} \in \Omega(n!)$

What does f(n) = O(g(n)) mean (in words)? *f(n) is upper-bounded by g(n), asymptotically*

What does f(n) = O(g(n)) *actually* mean?

There exists constants $c > 0, n_0 \in \mathbb{N}$ such that

$$cf(n) \leq g(n)$$

for all $n \geq n_0$

Now…

What does f(n) = Ω(g(n)) mean (in words)?

What does f(n) = Ω(g(n)) *actually* mean?

(a) The following statements are true or false?

3. $n^{\log n} \in \Omega(n!)$

(b) Sort the following functions in increasing order of asymptotic (big-O) complexity:

$$f_1(n) = n^{\sqrt{n}}, \quad f_2(n) = 2^n, \quad f_3(n) = n^{10} \cdot 2^{n/2}, \quad f_4(n) = \binom{n}{2}$$

Intuition: Exponentials dominate

Any clear relationships?

Question 4

(a) Show that $\max\{f(n), g(n)\} \in \Theta(f(n) + g(n))$ for any $f(n)$ and $g(n)$ that eventually become and stay positive.

(b) Give an example of f and g such that f is not $O(g)$ and g is also not $O(f)$.

f(n) = Θ(g(n)) is defined as f(n) = O(g(n)) and f(n) = Ω(g(n)) (simultaneously) i.e.

There exists constants $c > 0, n_0 \in \mathbb{N}$ such that

$$cf(n) \leq g(n)$$

for all $n \geq n_0$

AND

There exists constants $c' > 0, n_1 \in \mathbb{N}$ such that

$$c'f(n) \geq g(n)$$

for all $n \geq n_1$

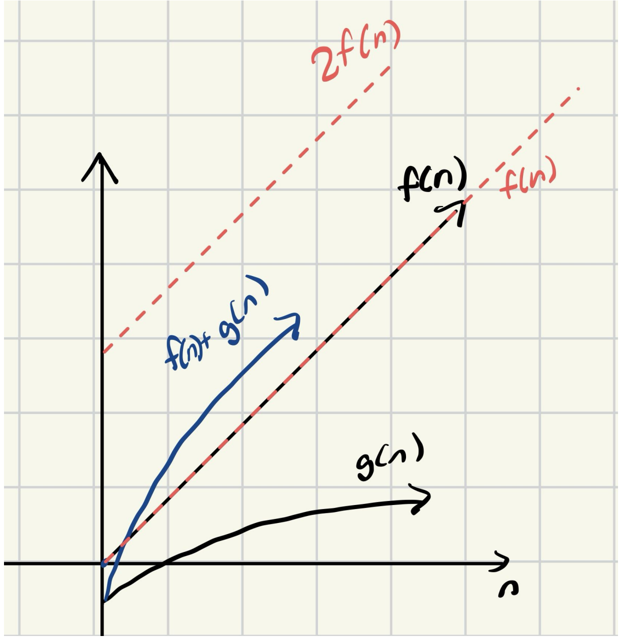
We can rewrite this as..

(a) Show that $\max\{f(n), g(n)\} \in \Theta(f(n) + g(n))$ for any $f(n)$ and $g(n)$ that eventually become and stay positive.

To convince ourselves, let's look at f(n) = n and g(n) = log n

(a) Show that $\max\{f(n), g(n)\} \in \Theta(f(n)+g(n))$ for any $f(n)$ and $g(n)$ that eventually become and stay positive.

Ex. f(n) = n and g(n) = log n



Prove formally using definition

There exist constants $c, c' > 0$, and $n_0 \in \mathbb{N}$ such that

$$c f(n) \le g(n) \quad \text{and} \quad c' f(n) \ge g(n)$$

for all $n \ge n_0$

1. max(f,g) = O(f + g)

2. max(f,g) = *Ω*(f + g)

(b) Give an example of f and g such that f is not $O(g)$ and g is also not $O(f)$.

Idea: crazy oscillating behavior