

01:198:344 - Homework I

Kev Sharma - kks107, Section 08

January 29, 2021

Partners collaborated with on this assignment: 1) Joel Martinez - Section 06

Problem 1.

1. $\mathcal{O}(g(n))$
2. $\mathcal{O}(g(n))$
3. $\Omega(g(n))$
4. $\Theta(g(n))$
5. $\mathcal{O}(g(n))$
6. $\mathcal{O}(g(n))$
7. $\Theta(g(n))$
8. $\mathcal{O}(g(n))$
9. $\Omega(g(n))$
10. $\mathcal{O}(g(n))$

Problem 2.

1. Prove by induction that

$$\sum_{i=0}^k i2^i = (k-1)2^{k+1} + 2$$

Proof. Base case: Let $k = 0$, then $\sum_{i=0}^0 i2^i = (0-1)2^{0+1} + 2$

$$\begin{aligned} 0 * 2^0 &= -1 * 2 + 2 \\ 0 &= 0 \end{aligned}$$

Inductive Step: For $k > 0$, we have:

$$\begin{aligned} \sum_{i=0}^k i2^i &= k2^k + \sum_{i=0}^{k-1} i2^i \\ &= k2^k + ((k-1)-1)2^{(k-1)+1} + 2 \\ &= k2^k + ((k-1)-1)2^{(k-1)+1} + 2 \\ &= k2^k + (k-2)2^k + 2 \\ &= k2^k + (k-2)2^k + 2 \\ &= k2^k + k2^k - 2 * 2^k + 2 \\ &= 2k2^k - 2^{k+1} + 2 \\ &= k2^{k+1} - 2^{k+1} + 2 \\ &= (k-1)2^{k+1} + 2 \\ &= \sum_{i=0}^k i2^i \end{aligned}$$

Thus for all k , we have shown that $\sum_{i=0}^k i2^i = (k-1)2^{k+1} + 2$.

□

2. Prove that $\sum_{i=1}^n \frac{i^4}{10} = \Theta(n^5)$

Proof. $\sum_{i=1}^n i^4/10 = \Theta(n^5)$

Let $f(n) = \sum_{i=1}^n i^4$

We must prove then that $\frac{1}{10}f(n) = \Theta(n^5)$.

It follows, by definition of Theta, that there must exist a c_1, c_2, n_1 such that:
 $0 \leq c_1(n^5) \leq \frac{1}{10}f(n) \leq c_2(n^5)$ for all $n \geq n_1$.

Equivalently we have: $0 \leq 10c_1(n^5) \leq f(n) \leq 10c_2(n^5)$ for all $n \geq n_1$.

Since $10c_1$ and $10c_2$ are constants, we have $f(n) = \Theta(n^5)$.

To prove $f(n) = \Theta(n^5)$ we must prove that $f(n) = \mathcal{O}(n^5)$ and $f(n) = \Omega(n^5)$

Case 1: Show $f(n) = \mathcal{O}(n^5)$

$$\begin{aligned} f(n) &= 1^4 + 2^4 + 3^4 + 4^4 + \dots + n^4 \\ &\leq n^4 + n^4 + n^4 + n^4 + \dots + n^4 \end{aligned}$$

Since $n^4 + n^4 + n^4 + n^4 + \dots + n^4 = n * n^4 = n^5$

It follows $1^4 + 2^4 + 3^4 + 4^4 + \dots + n^4 \leq n^5$

Accordingly $f(n) \leq n^5$

$\therefore f(n) = \mathcal{O}(n^5)$

Case 2: Show $f(n) = \Omega(n^5)$

$$\begin{aligned} f(n) &= 1^4 + \dots + \left(\left(\frac{n}{2}\right)^2\right) + \left(\left(\frac{n}{2}\right)^2 + 1\right) + \left(\left(\frac{n}{2}\right)^2 + 2\right) + \left(\left(\frac{n}{2}\right)^2 + 3\right) + \dots + n^4 \\ &\geq \left(\frac{n}{2}\right)^4 + \left(\frac{n}{2}\right)^4 + \left(\frac{n}{2}\right)^4 + \left(\frac{n}{2}\right)^4 + \dots + \left(\frac{n}{2}\right)^4 \end{aligned}$$

The RHS of inequality only has $\left(\frac{n}{2}\right)$ elements in the sequence.

Thus $\text{RHS} = \left(\frac{n}{2}\right)^4 + \left(\frac{n}{2}\right)^4 + \left(\frac{n}{2}\right)^4 + \left(\frac{n}{2}\right)^4 + \dots + \left(\frac{n}{2}\right)^4 = \left(\frac{n}{2}\right) * \left(\frac{n}{2}\right)^4 = \frac{n^5}{32}$

So we have: $1^4 + \dots + \left(\left(\frac{n}{2}\right)^2\right) + \left(\left(\frac{n}{2}\right)^2 + 1\right) + \left(\left(\frac{n}{2}\right)^2 + 2\right) + \left(\left(\frac{n}{2}\right)^2 + 3\right) + \dots + n^4 \geq \frac{n^5}{32}$

Accordingly $f(n) \geq \frac{n^5}{32}$;

$\therefore f(n) = \Omega(n^5)$

We have shown that $f(n)$ which is $\sum_{i=1}^n i^4$, is both $\mathcal{O}(n^5)$ and $\Omega(n^5)$.

Observe that $\frac{1}{10} \sum_{i=1}^n i^4 = \Theta(f(n))$ as $0 \leq \frac{1}{20}f(n) \leq \frac{1}{10} \sum_{i=1}^n i^4 \leq f(n)$ for all $n \geq 1$.

Accordingly $\frac{1}{10} \sum_{i=1}^n i^4 = \Theta(n^5)$ by the transitive property of Theta notation.

Hence we have $\sum_{i=1}^n \frac{i^4}{10} = \Theta(n^5)$

□

3. What is $\sum_{i=0}^{\log_2(n)} 4^i$ equal to in Θ -notation?

Employing $\sum_{i=0}^k r^i = (r^{k+1} - 1)/(r - 1)$ we have:

$$\begin{aligned}\sum_{i=0}^{\log_2(n)} 4^i &= (4^{\log_2(n)+1} - 1)/(4 - 1) \\ &= (4 * 4^{\log_2(n)} - 1)/3\end{aligned}$$

Applying log rule ($a^{\log_b(c)} = c^{\log_b(a)}$) we have:

$$\begin{aligned}&= (4 * n^{\log_2(4)} - 1)/3 \\ &= (4 * n^2 - 1)/3 \\ &= 4n^2/3 - 1/3 \\ &= \Theta(n^2)\end{aligned}$$

Justification:

$$\text{Let } f(n) = \sum_{i=0}^{\log_2(n)} 4^i = 4n^2/3 - 1/3$$

$$\text{Let } g(n) = n^2$$

Allowing $c_1 = 1$ and $c_2 = 5$, we have:

$$0 \leq c_1 * |g(n)| \leq |f(n)| \leq c_2 * |g(n)| \text{ for all } n > 2$$

Hence by definition, $f(n) = \Theta g(n)$

$$\text{So } \sum_{i=0}^{\log_2(n)} 4^i = \Theta(n^2)$$

Problem 3.

1. Simplify $8^{\log_4(n)}$:

Applying the log rule ($a^{\log_b(c)} = c^{\log_b(a)}$) we have:

$$\begin{aligned} n^{\log_4(8)} \\ &= n^{\log_4(4*2)} \\ &= n^{\log_4(4)+\log_4(2)} \\ &= n^{(1+0.5)} \\ &= n^{1.5} \end{aligned}$$

2. Simplify $3^{\log_2(n)}$:

Applying the log rule ($a^{\log_b(c)} = c^{\log_b(a)}$) we have:

$$\begin{aligned} n^{\log_2(3)} \\ &= n^{\log_2(2*1.5)} \\ &= n^{\log_2(2)+\log_2(1.5)} \\ &= n^{1+\log_2(1.5)} \end{aligned}$$

$\log_2(1.5)$ is irrational.

Directly computing $\log_2(1.5)$ gives approximately 0.5849625.

A more readable simplification gives $n^{1+0.5849625} = n^{1.5849625}$.

3. Prove that for any constants c, c' , $\log_c(n) = \Theta(\log_{c'}(n))$.

Proof. Note that c and c' will both be at least 2 (rules of log). I'll employ the Theta property of transitivity to demonstrate that $\log_c(n) = \Theta(\log_{c'}(n))$.

- Note that $\log_c(n) = \frac{\log_2(n)}{\log_2(c)}$ and $\log_{c'}(n) = \frac{\log_2(n)}{\log_2(c')}$ (by change of base formula)
- (1) Proof of the following $\log_2(n) = \Theta(\frac{\log_2(n)}{\log_2(c')})$

Proof. Let $p = \frac{\log_2(c')}{2}$ and $q = 2 \log_2(c')$

Then for all $n \geq 1$: $0 \leq p * \frac{\log_2(n)}{\log_2(c')} \leq \log_2(n) \leq q * \frac{\log_2(n)}{\log_2(c')}$

$$= 0 \leq \frac{\log_2(n)}{2} \leq \log_2(n) \leq 2 * \log_2(n)$$

By definition of Theta notation, we have proved that $\log_2(n) = \Theta(\frac{\log_2(n)}{\log_2(c')})$
 $\therefore \log_2(n) = \Theta(\log_{c'}(n))$ (by change of base formula)

□

- (2) Proof of the following $\frac{\log_2(n)}{\log_2(c)} = \Theta(\log_2(n))$

Proof. Let $p = \frac{1}{2 * \log_2(c)}$ and $q = \frac{2}{\log_2(c)}$

Then for all $n \geq 1$:

$$0 \leq p * \log_2(n) \leq \frac{\log_2(n)}{\log_2(c)} \leq q * \log_2(n)$$

$$= 0 \leq \frac{\log_2(n)}{2 * \log_2 c} \leq \frac{\log_2(n)}{\log_2(c)} \leq \frac{2 * \log_2(n)}{\log_2 c}$$

$$= 0 \leq 0.5 \log_2(n) \leq \log_2(n) \leq 2 \log_2(n)$$

By definition of Theta notation, we have proved that $\frac{\log_2(n)}{\log_2(c)} = \Theta(\log_2(n))$
 $\therefore \log_c(n) = \Theta(\log_2(n))$ (by change of base formula)

□

We demonstrated that $\log_c(n) = \Theta(\log_2(n))$ and that $\log_2(n) = \Theta(\log_{c'}(n))$.

$\therefore \log_c(n) = \Theta(\log_{c'}(n))$ (by the transitivity property of Theta notation)

□

Problem 4.

1. Closest Pair with a run-time better than $\mathcal{O}(n^2)$

*// @param A -> An array with n distinct elements, n >= 2
 // @return x,y -> s.t that |x-y| is as small as it could be*

```
PROCEDURE ClosestPair(A):

    sort(A)

    minDistance <- |A[0] - A[1]|
    x <- A[0]
    y <- A[1]

    i <- 2
    while i < A.length:

        if minDistance > |A[i-1] - A[i]|:
            minDistance <- |A[i-1] - A[i]|
            x <- A[i-1]
            y <- A[i]

        i <- i+1

    return x,y

END PROCEDURE
```

Analysis of run-time:

We sort A which is $\mathcal{O}(n \log(n))$ (discussed in class).

Computing the initial closest pair takes constant time.

The while loop takes $\mathcal{O}(n)$ time to complete since we iterate over A.length-2 number of elements. In each iteration we do a constant number of operations, hence each iteration takes $\mathcal{O}(1)$ time.

$$\begin{aligned} \text{Thus } T(n) &= n \log n + (n-1 \text{ iterations}) * (1) \\ T(n) &= n \log(n) + n - 1 \\ &= \mathcal{O}(n \log(n) + n) = \mathcal{O}(n \log(n)) \end{aligned}$$

Hence we have a worst case run-time of $\text{ClosestPair}(A) = \mathcal{O}(n \log(n))$.

2.

```
/**
  @param An array A of comparable elements
  @return The maximum value of A or null if none exist
 */
PROCEDURE FindMax(A):

    // An empty array implies no max.
    if A.length == 0:
        return null

    max <- A[0]
    count <- 1

    while count < A.length:

        if A[count] > max:
            max <- A[count]

        count <- count + 1

    return max
```

Analysis of run-time:

Our loop is guaranteed to iterate $n-1$ times, where $n = A.length$.
For each iteration, we do a constant number of operations.

Hence our procedure takes $(n-1)*1$ time to complete.
Accordingly FindMax(A)'s run-time is $\mathcal{O}(n)$.

3. Find a triplet of indices problem with a run-time of $\mathcal{O}(n^2)$

```

// @param T is the target sum
// @returns (true, I,J) where I=A[i], J=A[j], such that I+J=T
// @returns (false,-1,-1) if no A[i]+A[j] sum to T.
// note that i,j do not have to be unique.
PROCEDURE sorted-2-sum(A, T):
    i <- 0
    j <- A.length - 1
    while(i <= j):
        sum <- A[i] + A[j]

        if sum == T:      return (true,A[i],A[j])
        else if sum < T:  i <- i + 1
        else:             j <- j - 1

    return (false,-1,-1)

END PROCEDURE

/**
    @param A -> array of length n
    @return I,J,K such that I+J=K;
*/
PROCEDURE FindTriplets(A):

    sort(A)

    k <- A.length-1
    while k >= 0:

        K <- A[k]

        bool,I,J <- sorted-2-sum(A, K)

        if (bool,I,J) != (false,-1,-1):
            return I,J,K

        k <- k - 1

    return "no such triplet exists"

END PROCEDURE

```

Analysis of run-time:

We sort A which is $\mathcal{O}(n \log(n))$ (discussed in class)

The while loop takes $\mathcal{O}(n)$ time to complete since we iterate over A.length number of elements. Per iteration, the sorted-2-Sum method takes $\mathcal{O}(n)$ time to return (discussed in class).

Thus $T(n) = n \log n + (n \text{ iterations}) * (1 \text{ sorted-2-Sum call per iteration})$

$$T(n) = n \log(n) + n * n$$

Hence we have a worst case run-time of: $\mathcal{O}(n \log(n) + n * n) = \mathcal{O}(n * n) = \mathcal{O}(n^2)$.