01:198:344 - Homework I

Kev Sharma - kks107, Section 08

January 31, 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 $\sum_{i=0}^{0} i2^i = (0-1)2^{0+1} + 2$

$$0 * 2^0 = -1 * 2 + 2$$
$$0 = 0$$

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

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

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

$$\begin{gathered} f(n) = 1^4 + 2^4 + 3^4 + 4^4 + ... + n^4 \\ \leq n^4 + n^4 + n^4 + n^4 + ... + n^4 \end{gathered}$$

$$\begin{gathered} \text{Since } n^4 + n^4 + n^4 + n^4 + ... + n^4 = n * n^4 = n^5 \\ \text{It follows } 1^4 + 2^4 + 3^4 + 4^4 + ... + n^4 \leq n^5 \\ \text{Accordingly } f(n) \leq n^5 \\ \therefore f(n) = \mathcal{O}(n^5) \end{gathered}$$

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

$$\begin{gathered} f(n) = 1^4 + ... + ((\tfrac{n}{2})^2) + ((\tfrac{n}{2})^2 + 1) + ((\tfrac{n}{2})^2 + 2) + ((\tfrac{n}{2})^2 + 3) + ... + n^4 \\ \geq (\tfrac{n}{2})^4 + (\tfrac{n}{2})^4 + (\tfrac{n}{2})^4 + (\tfrac{n}{2})^4 + ... + (\tfrac{n}{2})^4 \end{gathered}$$

$$\begin{gathered} \text{The RHS of inequality only has } (\tfrac{n}{2}) \text{ elements in the sequence.} \\ \text{Thus RHS} = (\tfrac{n}{2})^4 + (\tfrac{n}{2})^4 + (\tfrac{n}{2})^4 + (\tfrac{n}{2})^4 + ... + (\tfrac{n}{2})^4 = (\tfrac{n}{2}) * (\tfrac{n}{2})^4 = \tfrac{n^5}{32} \end{gathered}$$

$$\begin{gathered} \text{So we have: } 1^4 + ... + ((\tfrac{n}{2})^2) + ((\tfrac{n}{2})^2 + 1) + ((\tfrac{n}{2})^2 + 2) + ((\tfrac{n}{2})^2 + 3) + ... + n^4 \geq \tfrac{n^5}{32} \\ \text{Accordingly } f(n) \geq \tfrac{n^5}{32}; \\ \therefore f(n) = \Omega(n^5) \end{gathered}$$

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

$$\sum_{i=0}^{\log_2(n)} 4^i = (4^{\log_2(n)+1} - 1)/(4 - 1)$$

$$= (4 * 4^{\log_2(n)} - 1)/3$$

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

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

Justification:

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

Let $g(n) = n^2$

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

$0 \le c_1 * |g(n)| \le |f(n)| \le c_2 * |g(n)|$ for all $n > 2$

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

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:

   $n^{\log_4(8)}$

   $= n^{\log_4(4*2)}$

   $= n^{\log_4(4)+\log_4(2)}$

   $= n^{(1+0.5)}$

   $= n^{1.5}$

2. Simplify $3^{\log_2(n)}$:
   Applying the log rule ($a^{\log_b(c)} = c^{\log_b(a)}$) we have:

   $n^{\log_2(3)}$

   $= n^{\log_2(2*1.5)}$

   $= n^{\log_2(2)+\log_2(1.5)}$

   $= n^{1+\log_2(1.5)}$

   $\log_2(1.5)$ 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 $\mathcal{O}(nlog(n))$ (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.

Thus T(n) = nlogn + (n-2 iterations)*(1)
T(n) = nlog(n) + n - 2
$= \mathcal{O}(n \log(n))$

Hence we have a worst case run-time of 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 $\mathcal{O}(nlog(n))$ (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) = nlogn + (n iterations)*(1 sorted-2-Sum call per iteration)
T(n) = nlog(n) + n*n

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

Problem Extra Credit. Prove that the total running time of the algorithm is $\Theta(n^3)$.

Suppose that A is of size seven, that is $n = 7$ (I'll use this example to derive a closed form equation for the number of computations required to solve the **Maximum Interval Value** algorithm):

i can thus take on the values from 0 to 6 inclusive.
j can take on the values from 1 to 6 inclusive.

In our example of $n = 7$, we would thus have the following loop iterations with their sum(i,j) computational time:

i=0, j=1; takes $\Theta(1)$
i=0, j=2; takes $\Theta(2)$
i=0, j=3; takes $\Theta(3)$
i=0, j=4; takes $\Theta(4)$
i=0, j=5; takes $\Theta(5)$
i=0, j=6; takes $\Theta(6)$

i=1, j=2; takes $\Theta(1)$
i=1, j=3; takes $\Theta(2)$
i=1, j=4; takes $\Theta(3)$
i=1, j=5; takes $\Theta(4)$
i=1, j=6; takes $\Theta(5)$

i=2, j=3; takes $\Theta(1)$
i=2, j=4; takes $\Theta(2)$
i=2, j=5; takes $\Theta(3)$
i=2, j=6; takes $\Theta(4)$

i=3, j=4; takes $\Theta(1)$
i=3, j=5; takes $\Theta(2)$
i=3, j=6; takes $\Theta(3)$

i=4, j=5; takes $\Theta(1)$
i=4, j=6; takes $\Theta(2)$

i=5, j=6; takes $\Theta(1)$

Total running time is thus equivalent to:
$= 6 * \Theta(1) + 5 * \Theta(2) + 4 * \Theta(3) + 3 * \Theta(4) + 2 * \Theta(5) + 1 * \Theta(6)$

$= 6 * \Theta(1) + 5 * \Theta(2) + 4 * \Theta(3) + 3 * \Theta(4) + 2 * \Theta(5) + 1 * \Theta(6)$

Rewriting this we have: $6 * 1 + 5 * 2 + 4 * 3 + 3 * 4 + 2 * 5 + 1 * 6 = 56$

Hence the total running time of the algorithm, equivalent to the sum of sum(i,j) for all possible combinations i,j s.t. $j > i$, is 56 when $n = 7$. While 56 doesn't immediately resemble 7^3, we'll continue to use it to determine a summation formula to compute the total running time of this algorithm for larger n.

Recall that since $n = 7$, we can rewrite this as:
$(n-1) * \Theta(1) + (n-2) * \Theta(2) + (n-3) * \Theta(3) + (n-4) * \Theta(4) + (n-5) * \Theta(5) + (n-6) * \Theta(6)$

Rewriting this we have (loosely converting $\Theta(x)$ into x computations):
$(n-1)(1) + (n-2)(2) + (n-3)(3) + (n-4)(4) + (n-5)(5) + (n-6)(6)$

$= n - 1$
$+2n - 4$
$+3n - 9$
$+4n - 16$
$+5n - 25$
$+6n - 36$
———-
Plugging in $n = 7$, we get 56.

$\therefore$ We know that this form of summation is a correct indication of the total running time.

In general this formula is of the form:
$= (n-1)(1) + (n-2)(2) + ... + (n-(n-1))(n-1)$

This summation can be equivalently written as:

$= \sum_{i=1}^{n-1} (n-i)(i)$

$= \sum_{i=1}^{n-1} in - i^2$

$= \sum_{i=1}^{n-1} in$ - $\sum_{i=1}^{n-1} i^2$

$= n \sum_{i=1}^{n-1} i$ - $\sum_{i=1}^{n-1} i^2$

$\therefore$ for any n, the total running time of this algorithm is $n \sum_{i=1}^{n-1} i$ - $\sum_{i=1}^{n-1} i^2$

Recap: we computed the total running time that the **Maximum Interval Value** algorithm takes when $n = 7$. Refactoring a little bit, we arrived at a summation formula to compute the total running time of this algorithm for any n:= $n \sum_{i=1}^{n-1} i$ - $\sum_{i=1}^{n-1} i^2$.

In order to prove that the running time of the algorithm is $\Theta(n^3)$, we will prove that $n\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 = \Theta(n^3)$.

Proof. $n\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 = \Theta(n^3)$

$n * [\frac{n(n-1)}{2}] - \sum_{i=1}^{n-1} i^2 = \Theta(n^3)$ (by summation formula)

$n * [\frac{n(n-1)}{2}] - [\sum_{i=1}^{n-1} i^2 + n^2] = \Theta(n^3) - n^2$ (subtracting n^2 from both sides)

$n * [\frac{n(n-1)}{2}] - [\sum_{i=1}^{n} i^2] = \Theta(n^3) - n^2$

$n * [\frac{n(n-1)}{2}] - \Theta(n^3) = \Theta(n^3) - n^2$ (as demonstrated in class)

$n * [\frac{n(n-1)}{2}] + n^2 = 2 * \Theta(n^3)$

$\frac{n^3-n^2}{2} + n^2 = 2 * \Theta(n^3)$

$\frac{n^3+n^2}{2} = 2 * \Theta(n^3)$

$\frac{n^3+n^2}{4} = \Theta(n^3)$

$0 \leq \frac{1}{4} * n^3 \leq \frac{n^3+n^2}{4} \leq n^3$ for all $n \geq 10$

$\therefore$ by definition of Theta notation we have $n\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 = \Theta(n^3)$

$\square$

Hence the **Maximum Interval Value** algorithm as presented to us is $\Theta(n^3)$.