

Problem 1. For the following functions, state either $f(n) \in o(g(n))$, $f(n) \in \omega(g(n))$, or $f(n) \in \Theta(g(n))$.

a. $f(n) = (n + \log n)(\sqrt{n} + 5)$, $g(n) = n \log n$

Solution: $f(n) = n^{1.5} + 5n + n^{0.5} \log n + 5 \log n$
 Because $5n$, $n^{0.5} \log n$, and $5 \log n$ are all in $o(n^{1.5})$, we can show that $f(n) \in \Theta(n^{1.5})$.
 Therefore $f(n) \in \omega(n \log n)$.

b. $f(n) = 2^{n \lg n}$, $g(n) = 3^n$

Solution: $f(n) = (2^{\lg n})^n = n^n \in \omega(3^n)$

c. $f(n) = \frac{n}{\lg n}$, $g(n) = \sqrt{n}$

Solution: In this case, $f(n)$ is less than n by a factor of $\lg n$; $g(n)$ is less than n by a factor of \sqrt{n} . A logarithm is asymptotically smaller than any positive power (or root) of n , so we expect that $f(n) \in \omega(g(n))$. We confirm this using the formal definition of ω .

For all $c > 0$ and all $n_0 > 0$, we construct a formula for n in terms of c and n_0 such that $n \geq n_0$ and $f(n) > cg(n)$. We proceed by starting from that inequality and solving for n .

$$\begin{array}{ll} \frac{n}{\lg n} & > c\sqrt{n} \\ n & > c\sqrt{n} \lg n \quad \text{by multiplying each side by } \lg n \\ \sqrt{n} & > c \lg n \quad \text{by dividing each side by } \sqrt{n} \\ \sqrt{n} & > c \sqrt[4]{n} \quad \text{is sufficient if we choose } n > 2^{16} \text{ so that } \sqrt{n} > \lg n \\ \sqrt[4]{n} & > c \quad \text{by dividing each side by } \sqrt[4]{n} \\ n & > c^4 \quad \text{by squaring both sides twice} \end{array}$$

Therefore we choose $n = n_0 + c^4 + 2^{16}$.

d. $f(n) = \sum_{i=1}^n (3i^2 \log i + 2i(\log i)^2)$, $g(n) = n^3 (\log n)^3$

Solution:

$$\begin{aligned} f(n) &= \sum_{i=1}^n (3i^2 \log i + 2i(\log i)^2) \\ &= 3 \sum_{i=1}^n (i^2 \log i) + 2 \sum_{i=1}^n (i(\log i)^2) \\ &\leq 3 \sum_{i=1}^n (i^2 \log n) + 2 \sum_{i=1}^n (i(\log n)^2) \\ &= 3(\sum_{i=1}^n i^2) \log n + 2(\sum_{i=1}^n i)(\log n)^2 \\ &= 3\left(\frac{2n^3 + 3n^2 + n}{6}\right) \log n + 2\left(\frac{n^2 + n}{2}\right) (\log n)^2 \\ &= n^3 \log n + \frac{3}{2} n^2 \log n + \frac{1}{2} n \log n + n^2 (\log n)^2 + n (\log n)^2 \\ &\in \Theta(n^3 \log n) \\ &\subseteq o(n^3 (\log n)^3) \end{aligned}$$

e. $f(n) = \frac{n}{n^{1/2}}$, $g(n) = \sqrt[4]{n^2}$

Solution: $f(n) = \frac{n}{n^{1/2}} = \sqrt{n} = \sqrt[4]{n^2} = g(n)$, therefore $f(n) \in \Theta(g(n))$.

Problem 2. Solve the following recurrences using summations, recursion trees, and/or the master method.

a. $T(n) = 3T(\frac{1}{4}n) + 1.5^n$

Solution: Solution via the master method where $a = 3$, $b = 4$, and $f(n) = 1.5^n$. In this case $f(n) \in \Omega(n^{\log_b a})$ because $f(n)$ is exponential, so we use case 3. We must choose $c < 1$ and $n_0 > 0$ such that for $n \geq n_0$, $af(\frac{n}{b}) \leq cf(n)$.

$$\begin{aligned} af(\frac{n}{b}) &\leq cf(n) \\ 3(1.5^{\frac{n}{4}}) &\leq c1.5^n && \text{by definition of } f(n) \\ 3 &\leq c1.5^{\frac{3n}{4}} && \text{by dividing both sides by } 1.5^{\frac{n}{4}} \\ 81 &\leq c^4 1.5^{3n} && \text{by squaring both sides twice} \\ 81 &\leq 1/3(1.5^{3n}) && \text{by choosing } c = \sqrt[4]{\frac{1}{3}} \\ 243 &\leq 1.5^{3n} && \text{by multiplying both sides by 3} \\ \log_{1.5} 243 &\leq 3n && \text{by taking the log base 1.5 of both sides} \\ \frac{\log_{1.5} 243}{3} &\leq n && \text{by dividing by 3} \end{aligned}$$

Therefore we choose $c = \sqrt[4]{\frac{1}{3}} \approx 0.760$ and $n_0 = \frac{\log_{1.5} 243}{3} \approx 4.52$, demonstrating that $T(n) = \Theta(1.5^n)$.

b. $T(n) = 2T(\frac{1}{2}n) + \log n$

Solution: Solution via the master method where $a = 2$, $b = 2$, and $f(n) = \log n$. In this case, $n^{\log_b a} = n^1$. We can choose any $0 < \epsilon < 1$ to make $f(n) = \log n \in O(n^{1-\epsilon})$, so case 1 applies, and $T(n) \in \Theta(n)$.

c. $T(n) = T(\frac{3}{5}n) + \sqrt[3]{n}$

Solution: Solution via the master method where $a = 1$, $b = \frac{5}{3}$, and $f(n) = \sqrt[3]{n}$; $n^{\log_b a} = n^0 = 1$. Therefore $f(n) = n^{\frac{1}{3}} \in \Omega(n^{0+\epsilon})$ for any $0 < \epsilon < \frac{1}{3}$, and we try case 3. We must choose $c < 1$ and $n_0 > 0$ such that for any $n \geq n_0$, $af(\frac{n}{b}) \leq cf(n)$.

$$\begin{aligned} af(\frac{n}{b}) &\leq cf(n) \\ \sqrt[3]{\frac{3}{5}n} &\leq c\sqrt[3]{n} && \text{by definition of } f(n) \\ \sqrt[3]{\frac{3}{5}}\sqrt[3]{n} &\leq c\sqrt[3]{n} \\ \sqrt[3]{\frac{3}{5}} &\leq c && \text{by dividing both sides by } \sqrt[3]{n} \end{aligned}$$

Therefore we choose $n_0 = 1$ and $c = \sqrt[3]{\frac{3}{5}} \approx 0.843$ to demonstrate that $T(n) \in \Theta(\sqrt[3]{n})$.

d. $T(n) = T(n-1) + n^2 + 3n + 2$

Solution: Solution via summation.

$$\begin{aligned} T(n) &= \sum_{i=1}^n i^2 + 3i + 2 \\ &= (\sum_{i=1}^n i^2) + 3(\sum_{i=1}^n i) + (\sum_{i=1}^n 2) \\ &= \frac{2n^3 + 3n^2 + n}{6} + 3(\frac{n^2 + n}{2}) + 2n \\ &= \frac{1}{3}n^3 + 2n^2 + \frac{11}{3}n \\ &\in \Theta(n^3) \end{aligned}$$

e. $T(n) = 2T(\frac{1}{4}n) + \sqrt{n}$

Solution: Solution via the master method where $a = 2$, $b = 4$, and $f(n) = \sqrt{n}$. Therefore $n^{\log_b a} = \sqrt{n} = f(n)$; case 2 applies and $T(n) \in \Theta(\sqrt{n} \log n)$.

f. $T(n) = T(\frac{1}{3}n) + T(\frac{2}{3}n) + n$

Solution: Solution by recursion trees. The first layer does n work. The second layer has two nodes of size $\frac{1}{3}n$ and $\frac{2}{3}n$, which adds up to n work again. The first node's children have size $\frac{1}{9}n$ and $\frac{2}{9}n$; the second node's children have size $\frac{2}{9}n$ and $\frac{4}{9}n$. The total is again n . Each layer splits up the work, but always totals n . The maximum height of the tree is $\log_{\frac{3}{2}} n$ at the leaf reached by taking $\frac{2}{3}$ of the input each time. The minimum height is $\log_3 n$ at the leaf reached by taking $\frac{1}{3}$ each time. The total work is therefore between $n \log_3 n$ and $n \log_{\frac{3}{2}} n$, and is in $\Theta(n \log n)$.

Problem 3. Here we examine a sorting algorithm for vectors based on swapping elements that are out of order. The first function, naturally enough, swaps elements that are out of order.

```
(define (sort!-swap-if-needed v i j)
  (define v.i (vector-ref v i))
  (define v.j (vector-ref v j))
  (cond
    [(<= v.i v.j) (void)]
    [else
     (vector-set! v i v.j)
     (vector-set! v j v.i)]))
```

a. State the running time of `sort!-swap-if-needed` as a recurrence. Solve the recurrence using summations, recursion trees, or the master method.

Solution: $T(n) = 1 \in \Theta(1)$

The `sort!-from/to` function sorts elements that are swapped from position `i` to position `j`, incrementing `j` until the end of the vector.

```
(define (sort!-from/to v i j)
  (cond
    [(>= j (vector-length v)) (void)]
    [else
     (sort!-swap-if-needed v i j)
     (sort!-from/to v i (add1 j))]))
```

b. State the running time of `sort!-from/to` as a recurrence. Solve the recurrence using summations, recursion trees, or the master method.

Solution: Solution by summation. $T(n) = T(n-1) + 1$, where $n = j - i$. Therefore $T(n) = \sum_{i=1}^n 1 = n \in \Theta(n)$.

The `sort!-from` function sorts elements that are swapped from position `i` to each position at a greater index; `i` increments until the end of the vector. Finally, `sort!` calls `sort!-from` starting at index 0.

```
(define (sort! v)
  (sort!-from v 0))

(define (sort!-from v i)
  (cond
    [(>= i (vector-length v)) (void)]
    [else
     (sort!-from/to v i (add1 i))
     (sort!-from v (add1 i))]))
```

c. State the running time of `sort!` as a recurrence. Solve the recurrence using summations, recursion trees, or the master method.

Solution: Solution by summation. $T(n) = T(n-1) + n$, where $n = |v| - i$. $T(n) = \sum_{j=1}^n j = \frac{1}{2}(n^2 + n) \in \Theta(n^2)$.

Problem 4. The *median of medians* algorithm selects the i th smallest element of a list xs , much like quickselect. The implementation starts with partitioning functions.

```
(define (all-less-than pivot xs)
  (cond
    [(empty? xs) empty]
    [(< (first xs) pivot) (cons (first xs) (all-less-than pivot (rest xs)))]
    [else (all-less-than pivot (rest xs))]))

(define (all-greater-than pivot xs)
  (cond
    [(empty? xs) empty]
    [(> (first xs) pivot) (cons (first xs) (all-greater-than pivot (rest xs)))]
    [else (all-greater-than pivot (rest xs))]))
```

a. State the running time of `all-less-than` and `all-greater-than` as recurrences. Solve the recurrences using summations, recursion trees, or the master method.

Solution: In both cases, $T(n) = T(n-1) + 1 \in \Theta(1)$.

The `groups-of-five` function splits a list into groups of five (or fewer) elements.

```
(define (groups-of-five xs)
  (cond
    [(empty? xs) empty]
    [(empty? (rest xs)) (list (list (first xs)))]
    [(empty? (rest (rest xs))) (list (list (first xs) (second xs)))]
    [(empty? (rest (rest (rest xs))))]
      (list (list (first xs) (second xs) (third xs)))]
    [(empty? (rest (rest (rest (rest xs)))))]
      (list (list (first xs) (second xs) (third xs) (fourth xs)))]
    [else (cons (list (first xs) (second xs) (third xs) (fourth xs) (fifth xs))
      (groups-of-five (rest (rest (rest (rest (rest xs)))))))]))
```

b. State the running time of `groups-of-five` as a recurrence. Solve the recurrence using summations, recursion trees, or the master method.

Solution: In this case, $T(n) = T(n-1) + 1$; we consider n to be $\lceil |xs| \rceil$. Again, $T(n) = \Theta(n)$.

The `median` function uses `slow-select` to find the median element of a list. Assume that `slow-select` runs in $n \log n$ time. The `medians-of-five` function takes a list of lists, where each inner list has at most five elements. It produces a list of numbers, where each number is the median of the corresponding list in the input.

```
(define (median five-or-less)
  (slow-select (quotient (length five-or-less) 2) five-or-less))

(define (medians-of-five fives)
  (cond
    [(empty? fives) empty]
    [else (cons (median (first fives))
      (medians-of-five (rest fives)))]))
```

c. State the running time of `medians-of-five` as a recurrence. Solve the recurrence using summations, recursion trees, or the master method.

Solution: $T(n) = T(n-1) + 1$ once again, where n is the length of fives. Here, the call to median takes $\Theta(1)$ time because the input is of constant size. The running time for medians-of-five is $\Theta(n)$.

Finally, the select function itself operates like quickselect, subdividing the input by partitioning. In order to guarantee a good pivot, the algorithm splits the input into groups of five elements, finds the median of each group of five, then uses a recursive call to select to find the median of all of those medians. That *median of medians* serves as the pivot for partitioning the input.

```
(define (select i xs)
  (define n (length xs))
  (cond
    [(<= n 5) (slow-select i xs)]
    [else
     (define fives (groups-of-five xs))
     (define medians (medians-of-five fives))
     (define pivot (select (quotient n 10) medians))
     (define left (all-less-than pivot xs))
     (define right (all-greater-than pivot xs))
     (define n1 (length left))
     (define n2 (length right))
     (cond
       [(< i n1) (select i left)]
       [(>= i (- n n2)) (select (- i (- n n2)) right)]
       [else pivot])]))
```

d. Argue that the length of (medians-of-five (groups-of-five xs)) is at most $\lceil \frac{1}{5}n \rceil$.

Solution: The groups-of-five function returns a list of exactly $\lceil \frac{1}{5}n \rceil$ groups, and medians-of-five produces the median of each one. Therefore the length of (medians-of-five (groups-of-five xs)) is precisely $\lceil \frac{1}{5}n \rceil$.

e. Argue that there are no more than $\frac{7}{10}n$ elements in left.

Solution: If pivot is the median of medians, then at least half the elements of medians are greater than or equal to pivot. In the corresponding groups of five elements in fives, there must therefore be at least three elements greater than or equal to pivot. Three out of five, out of half of xs, is three tenths of xs that is greater than or equal to pivot, and thus not in left. Therefore left can have at most $\frac{7}{10}n$ elements.

f. Argue that there are no more than $\frac{7}{10}n$ elements in right.

Solution: This follows symmetrically from the argument for left above; if there are three tenths of the list that must be greater than or equal to pivot, there are likewise three tenths of the list that must be less than or equal to pivot, and therefore not in right.

g. State the running time of select as a recurrence. **Note:** this recurrence may have an unusual form, as not every recursion in select has the same worst-case input size.

Solution: $T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + n$

h. Solve the recurrence for the running time of select using summations, recursion trees, or the master method.

Solution: Solution by recursion trees. The first layer does n work. The second layer contributes at most $\frac{n}{5} + \frac{7n}{10} = \frac{9n}{10}$ work. The third layer, $\frac{1}{5} \frac{1}{5} n + \frac{1}{5} \frac{7}{10} n + \frac{7}{10} \frac{1}{5} n + \frac{7}{10} \frac{7}{10} n = \frac{81}{100} n$. In each layer, the work is split into $\frac{1}{5}$ and $\frac{7}{10}$ of the total, giving a consistent reduction by a factor of $\frac{9}{10}$. The size of the tree is therefore bounded by $\sum_{i=0}^{\infty} (\frac{9}{10})^i n = \frac{1}{1-\frac{9}{10}} n = 10n \in \Theta(n)$.