COMPSCI-1DM3: Assignment #1

Author: Qusay Qadir
Instructor: Mahdee Jodayree
MacID: qadirq
Tut: T02

Due Date: July 7th, 2023

Contents

1 Question #1. [10 Marks]

Create a truth table for each of the following compound propositions.

a) ($q \rightarrow \neg p$) $\leftrightarrow$ ($p \leftrightarrow q$)

q	p	$\neg p$	$(q \rightarrow \neg p)$	$(p \leftrightarrow q)$	$(q \rightarrow \neg p) \leftrightarrow (p \leftrightarrow q)$
T	T	F	F	T	F
T	F	T	T	F	F
F	T	F	T	F	F
F	F	T	T	T	T

b) ($q \leftrightarrow p$) $\oplus$ ($p \leftrightarrow \neg q$)

p	q	$\neg q$	$(q \leftrightarrow p)$	$(p \leftrightarrow \neg q)$	$(q \leftrightarrow p) \oplus (p \leftrightarrow \neg q)$
T	T	F	T	F	T
T	F	T	F	T	T
F	T	F	F	T	T
F	F	T	T	F	T

2 Question #2. [10 Marks]

Express these system specifications using logical connectives (including negations) and the following propositions.

p: "The user enters a valid password,"
q: "Access is granted,"
r: "The user has paid the subscription fee."

a) "The user has paid the subscription fee but does not enter a valid password."

Solution: $\mathrm{r} \wedge \neg p$

b) "Access is granted whenever the user has paid the subscription fee and enters a valid password."

Solution: $(\mathrm{r} \wedge \mathrm{p}) \rightarrow q$

c) "Access is denied if the user has not paid the subscription fee."

Solution: $\neg r \rightarrow \neg q$

d) "If the user has not entered a valid password but has paid the subscription fee, then access is granted."

Solution: $(\neg p \wedge r) \rightarrow q$

3 Question #3. [10 Marks]

For each function, determine whether that function is $\Omega(x)$ and whether it is $\Theta(x)$

For the following questions, inequality 1 will be referred to the following: $|f(x)| \leq C \cdot |g(x)|$ to determine big-O and inequality 2 will be referred to: $|f(x)| \geq C \cdot |g(x)|$ to determine big-Omega

a) *f(x)* = 10

To find big-Omega for f(x) = 10, let's use inequality 2, where C = 9 and x > k such that k > 1, the inequality is satisfied as 10 ≥ 9 we can conclude that f(x) = 10 is $\Omega(1)$

Therefore, f(x) = 10 is $\Omega(1)$ and not $\Omega(x)$, concurrently f(x) cannot be $\Theta(x)$

b) *f(x)* = 3x + 7

To find big-O for f(x) = 3x + 7 lets use inequality 1, where C = 10 and x > k such that k > 1, the inequality is satisfied since 3x + 7 ≤ 10x, we can conclude that f(x) = 3x + 7 is O(x)

To find big-Omega for f(x), lets use inequality 2, where C = 3 and x > k such that k > 1, the inequality is satisfied since 3x + 7 ≥ 3x, we can conclude that f(x) = 3x + 7 is $\Omega(x)$

Therefore, since both O(x) and $\Omega(x)$ are for f(x), f(x) is $\Theta(x)$

c) *f(x)* = x^2 + x + 1

To find big-Omega for f(x) = x^2 + x + 1 lets use inequality 2, where C = 1 and x > k such that k > 1, the inequality is satisfied since x^2 + x + 1 ≥ x^2, we can conclude that f(x) = $3x^2$ + x + 1 is $\Omega(x^2)$.

Therefore we can conclude that f(x) = x^2 + x + 1 is not $\Omega(x)$ and neither can it be $\Theta(x)$

d) *f(x)*= 5log(x)

To find big-Omega for f(x) = 5log(x) lets use inequality 2, where C = 4 and x > k such that k > 1, the inequality is satisfied since 5log(x) ≥ 4log(x), we can conclude that f(x) = 5log(x) is $\Omega(log(x))$

Therefore we can conclude that f(x) = 5log(x) is not $\Omega(x)$ and thus neither can it be $\Theta(x)$

4 Question #4. [10 Marks]

Show that $\frac{x^3+2x}{2x+1}$ is $O(x^2)$.

let f(x) = $\frac{x^3+2x}{2x+1}$

$|f(x)| \leq$ C$\cdot$ $|g(x)|$ for C is a positive integer such that x $>$ k where k $>$ 1, the inequality can be satisfied by

$\frac{x^3+2x}{2x+1} \leq \frac{x^3+2x^3}{2x}$

$\frac{x^3+2x}{2x+1} \leq \frac{3x^3}{2x}$

$\frac{x^3+2x}{2x+1} \leq \frac{3}{2}$x

where C = 3/2 for k $>$ 1 we proved that f(x) is $O(x^2)$

5 Question #5. [10 Marks]

Show that each of these pairs of functions is of the same order.

To show that the following pairs of functions are of the same order we must determine whether the functions have the same big-O ($Og(x)$)

For the following questions, inequality 1 will be referred to the following: $|f(x)| \leq$ C$\cdot |g(x)|$ to determine big-O and inequality 2 will be referred to:$|f(x)| \geq$ C$\cdot|g(x)|$ to determine big-Omega

a) 3x + 7, x

Firstly, to find big-O of f(x) = 3x + 7, we can use inequality 1, where C = 10 and x > k such that k > 1, we get 3x + 7 $\leq$ 10x.

$Therefore$, f(x) = 3x + 7 is O(x)

To find big-O of g(x) = x , we can use inequality 1, where C = 2 and x > k such that k > 1, we get x $\leq$ 2x.

$Therefore$, g(x) = x is O(x)

Since, both f(x) = 3x + 7 and g(x) = x are $O(x)$ this is conclusive proof both pairs of functions are of the same order.

b) $2x^2$ + x - 7, x^2

To find big-O for f(x) = $2x^2$ + x - 7, we can use inequality 1, let C = 3 and x > k such that k > 1. Thus satisfying the inequality as $2x^2$ + x - 7 $\leq 3x^2$.

$Therefore$, for f(x) it is $O(x^2)$

To find big-O for g(x) = x^2, we can use inequality 1,let C =2 and x > k such that k > 0. Thus satisfying the inequality as $x^2 \leq 2x^2$.

$Therefore$, for g(x) it is $O(x^2)$

Since both f(x) = $2x^2$ + x - 7 and g(x) = x^2 are $O(x^2)$ this is conclusive proof both pairs of functions are of the same order.

c) $\log(x^2 + 1)$, $\log_2 x$

To find big-O for f(x) = $\log(x^2 + 1)$ we can use inequality 1, let C = 4 and x > k such that k > 1.

$$\log(x^2+1) \leq \log_2(x^2+1)$$
$$\log(x^2+1) \leq \log_2((2x)^2)$$
$$\log(x^2+1) \leq 2\log_2(x^2)$$
$$\log(x^2+1) \leq 4\log_2(x)$$

Therefore, for f(x) it is $O(\log_2(x))$

To find big-Omega for f(x) = $\log(x^2+1)$ we can use inequality 2, let C = $1/\log_2(10)$ and x > k such that k > 1.

$$\log(x^2+1) \geq \frac{\log_2(x^2+1)}{\log_2(10)}$$
$$\log(x^2+1) \geq \frac{\log_2(x)}{\log_2(10)}$$

Therefore, for f(x) it is $\Omega(\log_2(x))$

The average bound for f(x) is $\Theta(\log_2(x))$

Thus proving since f(x) is $\Theta(\log_2(x))$ and if we consider the order of g(x) = $\log_2(x)$ as $\log_2(x)$ then both f(x) and g(x) are of the same order.

d) $\log_{10} x$, $\log_2 x$

To find big-O for f(x) = $\log_{10} x$ we can use inequality 1, let C = 1 and x > k such that k > 1 we get $\log_{10} x \leq \log_2 x$ where the upper bound is $O(\log_2 x)$

Therefore, for f(x) it is $O(\log_2(x))$

To find big-Omega for f(x) = $\log_{10} x$ we can use inequality 2, let C = $1/(\log_2 10)$ and x > k such that k > 1 we get $\log_{10} x \geq \frac{\log_2(x)}{\log_2(10)}$ where the lower bound is $\Omega(\log_2 x)$

Therefore, for f(x) it is $\Omega(\log_2(x))$

The average bound for f(x) is $\Theta(\log_2(x))$

Thus proving since f(x) is $\Theta(\log_2(x))$ and if we consider the order of g(x) = $\log_2(x)$ as $\log_2(x)$ then both f(x) and g(x) are of the same order.

6 Question #6. [20 Marks]

Show that $x^5y^3 + x^4y^4 + x^3y^5$ is $\Omega(x^3y^3)$

Let f(x) be represented by $x^5y^3 + x^4y^4 + x^3y^5$

Let g(x) be represented by (x^3y^3)

For f(x) to have a $\Omega(x^3y^3)$, the inequality $|f(x)| \geq C{\cdot}|g(x)|$ must be satisfied for some positive integer C such that x > k and y > k

If such C does exist then this inequality $x^5y^3 + x^4y^4 + x^3y^5 \geq C{\cdot}(x^3y^3)$ must be satisfied. To determine this, let's first simplify f(x) by factoring out g(x). This will appear as $x^3y^3(x^2 + xy + y^2)$.

This demonstrates that for the following inequality to hold true, the positive integer C can be a constant as long as it is $< x^2 + xy + y^2$ for any given x > k and y > k such that k > 1.

As long as the value of the constant $C < x^2 + xy + y^2$, f(x) = $x^5y^3 + x^4y^4 + x^3y^5$ is $\Omega(x^3y^3)$

7 Question #7. [20 Marks]

Consider the following algorithm, which takes as input a sequence of n integers a_1,a_2,...., $a_n$and produces as output a matrix $\mathbf{M} = m_{ij}$ where m_{ij} is the minimum term in the sequence of integers a_i,a_{i+1},...., a_j for $j \geq i$ and $m_{ij} = 0$ otherwise.

initialize $\mathbf{M}$ so that $m_{ij} = a_i$, *if* $j \geq i$ and $m_{ij} = 0$ otherwise

for $i := 1$ **to** n
  for $j := i + 1$ **to** n
    for $k := i + 1$ **to** j
      $m_{ij} := \min(m_{ij}, ak)$

return $\mathbf{M} = (m_{ij})$ (m_{ij} is the minimum term of a_i,a_{i+1},...., a_j)

a) Show that this algorithm uses $O(n^3)$ comparisons to compute the matrix $\mathbf{M}$

The following pseudocode above has 3 nested for loops, the first loop runs a maximum of a total of n times, then the middle loop can run for another maximum of n times, and the final loop can also run for a maximum total of n times and has only 1 operation in the 3rd nested loop.Thus, the algorithm uses $O(n^3)$ comparisons to compute the matrix.

b) Show that this algorithm uses $\Omega(n^3)$ comparisons to compute the matrix $\mathbf{M}$. Using the fact and part (a), conclude that the algorithms uses (n^3) comparisons.

Hint: Only consider the cases where $i \leq n/4$ and $j \leq 3n/4$ in the two outer loops in the algorithm.]

8 Question #8. [30 Marks]

What is the effect in the time required to solve a problem when you double the size of the input from n to $2n$, assuming that the number of milliseconds the algorithm uses to solve the problem with input size n is each of these functions? [Express your answer in the simplest form possible, either as a ratio or difference. Your answer may be a function of n or a constant.]

a) $\log \log n$

log log(2n) - log log(n)

= log (log (2) + log(n)) - log (log(n))

= log ($\frac{1+log(n)}{log(n)}$)

= log ($\frac{1}{log(n)} + 1$)

For a very large arbitrary value of n the fraction approaches zero, while log (1 + 0) is also very close to zero, thus demonstrating that the size of the input n to 2n for the following algorithm has near zero of an effect in the time.

b) $\log(n)$

log (2n) - log (n)

= $\log \frac{(2n)}{(n)}$

= log(2) = 1

Changing the size of the input requires does not have an effect on the time when you double the size of the input.

c) $100n$

$\frac{100(2n)}{100(n)}$

= 2

Changing the size of the input requires 2 times more amount of time than doubling the input size

d) $n \log n$

$\frac{2nlog(2n)}{nlog(n)}$

$= 2\ (\frac{log2+log(n)}{log(n)})$

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

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

For a very large arbitrary n, $\frac{1}{log(n)}$ approaches zero it will equal to 2(1), implying almost twice the amount of time is needed to execute the algorithm with twice the input.

e) n^2

$\frac{(2n)^2}{n^2}$

$\frac{(2n)^2}{n^2}$

$= 4$

It will take 4 times as much longer to solve the algorithm the double the size of the input

f) n^3

$\frac{(2n)^3}{n^3}$

$= \frac{8n^3}{n^3}$

$= 8$

It will take 8 times as much longer to solve the algorithm the double the size of the input

g) 2^n

$\frac{2^{2n}}{2^n}$

$= 2^n$, the amount of time required to process the algorithm is largely dependent on the size of the input, n.

9 Question #9. [30 Marks]

Give a big-O estimate for each of these functions.

For the function of g in your estimate *f(x)* is *O(g(x))*, use a simple function g of smallest order.

When referring to use of Theorem 2: suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 + f_2)$(x) is O(g(x)), where g(x) = (max $(g_1(x), g_2(x))$) for all x.

When referring to use of Theorem 3: suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$. Then $(f_1 f_2)$(x) is O$(g_1(x)g_2(x))$

a) $(n^3 + n^2 log$ n$)(\log n + 1) + (17 \log n + 19)\ (n^3 + 2)$

Let's first use Theorem 2 on $(n^3 + n^2 log$ n$)$, where big-O of n^3 is $O(n^3)$, and in addition use Theorem 3 on $(n^2 log$ n$)$ where big-O for n^2 is $O(n^2)$ and big-O for log(n) is $O(log(n))$, as such big-O for $(n^2 log$ n$)$ is $O(n^2 log(n))$. Since $O(n^3) > O(n^2 log(n))$ for $(n^3 + n^2 log$ n$)$ big-O is $O(n^3)$.

Use Theorem 2 on $(\log n + 1)$ where big-O for log(n) is $O(log(n))$, and big-O for 1 is $O(1)$. Big-O for $(\log n + 1)$ is $O(log(n))$ since it is the greater value compared to $O(1)$

Use Theorem 3 on $(n^3 + n^2 log$ n$)(\log n + 1)$ where big-O is $O(n^3 \cdot log(n))$

Use Theorem 2 on $(17 \log n + 19)$ where big-O for $17 \log n$ is $O(log(n))$ and big-O for 19 is $O(1)$. Since $O(log(n)) > O(1)$ for $(17 \log n + 19)$ big-O is $O(log(n))$

Use Theorem 2 on $(n^3 + 2)$ where big-O for n^3 is $O(n^3)$ and big-O for 2 is $O(1)$. Thus for $(n^3 + 2)$ big-O is $O(n^3)$

Using Theorem 3 on $(17 \log n + 19)\ (n^3 + 2)$ where big-O is $O(n^3 \cdot log(n))$

Use Theorem 2 on $(n^3 + n^2 log$ n$)(\log n + 1) + (17 \log n + 19)\ (n^3 + 2)$ where the final big-O estimate for the entire function is $O(n^3 \cdot log(n))$

b) $(2^n + n^2)(n^3 + 3^n)$

Use Theorem 2 on $(2^n + n^2)$ where big-O for 2^n is $O(2^n)$ and big-O for n^2 is $O(n^2)$. Since $O(2^n) > O(n^2)$ big-O estimate of $(2^n + n^2)$ is $O(2^n)$

Use Theorem 2 on $(n^3 + 3^n)$ where big-O for n^3 is $O(n^3)$ and big-O for 3^n is $O(3^n)$. Since $O(3^n) > O(n^3)$ big-O estimate of $(n^3 + 3^n)$ is $O(3^n)$

Use Theorem 3 on $(2^n + n^2)(n^3 + 3^n)$ where the final big-O estimate of the entire function is $O(6^n)$

c) ($n^n + n2^n + 5^n$) (n! + 5^n)

Use an extended version of Theorem 2 as there are 3 terms in this polynomial ($n^n + n2^n + 5^n$). Firstly, big-O for n^n is $O(n^n)$, for $n2^n$ after we use theorem 3 big-O is $O(n2^n)$, and finally big-O for 5^n is $O(5^n)$. Thus, big-O for($n^n + n2^n + 5^n$) is $O(n^n)$ as $O(n^n) > O(5^n) > O(n2^n)$.

Use Theorem 2 on (n! + 5^n), for n! big-O is $O(n!)$ and big-O for 5^n is $O(5^n)$. For (n! + 5^n) big-O estimate is $O(n!)$ as $O(n!) > O(5^n)$

Use Theorem 3 on ($n^n + n2^n + 5^n$) (n! + 5^n) the final big-O estimate for the entire function is $O(n!n^n)$

10 Question #10. [10 Marks]

Use the insertion sort to sort d, f, k, m, a, b, showing the lists obtained at each step.

Start with the original list:(d, f, k, m, a, b)

First step: (d, f, k, m, a, b)
The loop starts at the second index and compares it to all the values on its left and checks whether or not if it is larger or not if it is then the position swap if not then the letters stay in their respective index. Since f > d, there is no swap as the letters are in the right order so far.

Second step: (d, f, k, m, a, b)
The pointer now is on the 3rd index and compares itself to the values on its left which is f and d. Since k > f and f > d, k is in the correct position and no swapping is needed.

Third step: (d, f, k, m, a, b)
The pointer is now on the 4th index, m, which compared to the letters on its left is m > k, and k > f and f > d so the position of m is correct and requires no swapping.

Fourth step: (a, d, f, k, m, b)
Pointer is now at a, and since a < m a < k a < f and a < d, all the swapping of position happens in one loop and now the position of a is in the correct order of the array since a < d < f < k < m

Fifth step: (a, b, d, f, k, m)
Pointer is now on the final index b, since b < m there is a swap, and then b < k so a swap, b < f so a swap, and then b < d so a final swap because comparing b to a, b > a so there will be no swap of positions.

Thus having a finalized output of the sorted list in increasing order of the letters in the alphabet.

11 Question #11. [20 Marks]

Write the selection sort algorithm in pseudocode.
The selection sort beings by finding the least element in the list.
This element is moved to the front.
Then the least element among the remaining elements is found and put into the second position.
This procedure is repeated until the entire list has been sorted.

procedure ($a_1, a_2, ..., a_n$: real numbers with n $\geq$ 2)
for i := 1 to n
    min-num := a_i
    for j := i+1 to n
        if min-num > a_j
        min-num := a_j
    temp := min-num
    min-num := a_i
    a_i := temp
{$a_1, a_2, ..., a_n$ is in increasing order }

12 Questions #12. [20 Marks]

Devise an algorithm that finds all terms of a finite sequence of integers that are greater than the sum of all previous terms of the sequence.

procedure ($a_1, a_2, ..., a_n$: real numbers with n $\geq$ 2)
sum := 0
list := { empty }
for i := 1 to n
    if $a_i > sum$
    sum := sum + a_i
    list := append a_i
{$a_1, a_2, ..., a_n$, is a list of integers where a term is greater than the sum of all the previous integers}