**CS 58000\_01/02I Design, Analysis and Implementation Algorithms (3 cr.)**

**Assignment As\_03**

**Student Name: Harsh Sharma**

This assignment As 03 is due at 11:59 p.m., Wednesday, October 25, 2023. Please submit your assignment solution to Brightspace (purdue.brightspace.com). No late turn-in is accepted. Please write your name on the first page of your assignment. Your file name should be your last name such as NgP\_As03.docx. Please number your problem-answer clearly such as Problem I.a, I.b, I.c.i, I.c.ii, II, III.(a), III.(b), III.(c), III.(d), III.(e), III.(f). The problems’ answers must be arranged according to the order of the given problem. Please answer your questions using only a Word file (.docx file only). No pdf file will be accepted. Without using a Word file (.docx file) the submitted problems’ answers would not be graded.

The total number of points for this Assignment\_03 is 100 points.

1. [30 pts] Using definitions of Θ, O, and Ω to establish your claims:

a: Prove and disprove that for any real constants a and b, where b > 0,

(n + log2a)b = Θ(nb ).

**Solution:**

First let’s discuss the definition of Θ, O, and Ω.

O (big-Oh) represents the upper bound of f(n), where the function is of our interest.

Ω (big-Omega) represents the lower bound of f(n).

Θ (theta) represents the average bound of f(n).

Now in order to prove that, (n + log2a)b = Θ(nb ) we need to find the value of c1 and c2.

Now we know that, n + a ≤ 2n, when ∣a∣ ≤ n ∣a∣ ≤ n.

This also implies that,

n + log2a ≤ 2n {as log2a < a}

Also, we have, n + a ≥ n/2, when ∣a| ≤ n/2

This implies that,

n + log2a ≥ n/2 {as log2a < a}

Therefore, when n ≥ 2 ∣a∣, both of the above inequalities hold true, and we can write,

0 ≤ n / 2 ​≤ n + log2a ≤ 2n

As b > 0, we can raise all the terms of the previous inequality to the power of b without breaking the inequality:

0 ≤ (n/2)b ≤ (n + log2a)b ≤ (2n)b

On simplifying the above equation:

0 ≤ (1/2)b nb ≤ (n + log2a)b ≤ 2bnb

So, (n+log2a)b = Θ(nb) must be true because there exists c1 = 1/2b , c2 = 2b and n0 = 2∣a∣.

Hence **Proved**.

Now, for **disapproving**:

We have the following equations (n + log2a)b = Θ(nb ).

Let’s take b = 2 {as b > 0, according to the restriction of question.}

LHS:

(n + log2a)2

Taking n = 1 and a = 2 {n > 0 and ‘a’ can be any value, only b has a

restriction}

(1 + log22)2

(1 + 1)2

4

RHS:

= nb

= 12 {on substituting the value of n and b}

= 1

Clearly, LHS ≠ RHS. Hence **disapproved**.

b: Explain why the statement “The running time of algorithm A, T(A) 𝑖𝑠

𝑎𝑡 𝑙𝑒𝑎𝑠𝑡

O(n2 )” is meaningless.

**Solution:**

We know that big-oh notation represents **upper bound**. But this upper bound has to be the **nearest possible** to f(n) to actually make sense. Saying that the runtime is 'at least (some upper bound)' is **confusing** – as it means **upper bound is really a lower bound**.

Consider this example:

1 < log(n) < sqrt(n) < n < nlog(n) < n2 < n3 < … < 2n < 3n … nn

If f(n) = 5n + 3

Then,

f(n) = O(n)

f(n) = O(n2)

f(n) = O(n3)

f(n) = O(2n)

All the above statements are correct as all represent the upper bound for f(n). But saying that running time of f(n) is at least O(n3) is meaningless as the statement also holds true for O(n2), O(n), O(2n), etc.

The running time of the algorithm is at least O(n2) is pointless since it does not describe what could be neither the average case nor the worst one. If the above statement could be described in words it means like: ‘This tree is at least 30 feet tall or less.’ It tells nothing about the height of tree, it could be more than 30 feet or less. Replace 30 with any arbitrary number and still the statement.

Therefore, saying "T(A) is at least O(n^2)" creates confusion because Big O notation inherently represents an upper bound. And hence, the statement “The running time of algorithm A, T(A) 𝑖𝑠 𝑎𝑡 𝑙𝑒𝑎𝑠𝑡 O(n2 )” is meaningless.

c: Prove or disprove:

1. = O( )

**Solution**:

Let f(n) =

And g(n) =

We know that f(n) is O(g(n)) if there are positive constants C and k such that f(n) ≤ C \* g(n) whenever n>k.

Let’s try to find the value of constant C.

Calculating: f(n)/g(n)

f(n) / g(n) = / = \* 2 / = 2

therefore, f(n) = 2. = 2.g(n)

Or, f(n) = O( )

Hence, **proved.**

1. = O( )

**Solution:**

can be written as .

Now, we know that = n.

Therefore, is equal to (n+1)2

Also, on simplifying left side of equation, = n. We have,

(n+1)2 = O(n)

So, we need to either prove or disapprove this:

(n+1)2 = O(n)

We know by the definition of Big O notation a function f(n) is said to be O(g(n)) if there exist constants C and n0 such that for all n ≥ n0, |f(n)| ≤ C \* |g(n)|.

In our case, we have:

f(n) = (n + 1)2

g(n) = n

Now, let's analyze the limit of the ratio (n + 1)2 / n as n approaches infinity:

lim (n -> ∞) [(n + 1)^2 / n] = lim (n -> ∞) (n^2 + 2n + 1) / n

= lim (n -> ∞) (n^2/n + 2n/n + 1/n)

= lim (n -> ∞) (n + 2 + 1/n)

= ∞ + 2 + 0

= ∞

The limit is not bounded by a constant, which means (n + 1)2 is not O(n). Therefore, the statement 2(2 \* log2(n + 1)) = O(2log2(n)) is false.

In conclusion, 2(2 \* log2(n + 1)) is not O(2log2(n)). Hence, **disapproved**.

II.[10 pts]

Is growing faster than the exponential function ?

Use to show your claim.

What is the order of growth if the closed form formula is ?

**Solution:**

We have the following limit:

We can simplify this limit,

Now, as n tends to infinity the limit 1/n will tend to 0.

This limit demonstrates that the fraction approaches 0 as n becomes larger. This means that as n grows, n⋅2n grows faster than 2n because the fraction gets smaller and approaches zero.

**So, n⋅2n indeed grows faster than the exponential function 2n, and the limit confirms this conclusion.**

To find the order of growth of n.2n we can use this equality equation:

1 <= n <= 2n {for every n > 0 this statement holds true}

Multiplying both sides of the first inequality by 2^n, we get:

2n <= n2n

Multiplying both sides of the second inequality by c1, we get:

c1 \* 2n <= c1 \* n \* 2n

Combining these two inequalities, we get:

c1 \* 2n <= n \* 2n <= 2n

for all n >= n0 for some constant n0. This means that n2n is sandwiched between two constant multiples of 2n for all sufficiently large values of n.

Therefore, the order of growth of n2n is Θ(2n) in Big Theta notation.

III.[60 points] Given the following recurrence relation

T(n) = T(n -1) + f(n),

find a bound for each of them.

1. if f(n) = c, where c is a constant.

**Solution:**

I’ll use backward substitution to solve the above question.

We have, T(n) = T(n -1) + f(n)

T(n) = T(n -1) + c {as f(n) = c}

Let’s now calculate T(n-1)

T(n-1) = T((n-1) -1) + c

T(n-1) = T(n-2) + c

Similarly,

T(n-2) = T(n-3) + c

On backward substituting value of T(n-1) and T(n-2) in T(n) we get:

T(n) = T(n-1) + c

T(n) = T(n-2) + c + c

T(n) = T(n -2) + 2c

T(n) = T(n-3) + c + 2c

T(n) = T(n-3) + 3c

For the kth iteration

T(n) = T(n-k) + kc

If we keep on repeating this, then we will reach a point such that T(0) for some iteration.

Then, n – k = 0

Implies that, n = k

Therefore, we’ll get T(0) when n = k. On substituting the values,

T(n) = T(n-n) + nc

T(n) = T(0) + nc

T(n) = 1 + nc {T(0) is constant, and so assuming T(0) = 1}

Therefore, the bound for T(n) = T(n -1) + c is

**And so, the bound for f(n) = c is**

1. if f(n) = log2 n.

**Solution:**

We have, T(n) = T(n -1) + f(n)

T(n) = T(n -1) + log2 n {as f(n) = log2 n.}

Let’s now calculate T(n-1)

T(n-1) = T((n-1) -1) + log2 (n-1)

T(n-1) = T(n-2) + log2 (n-1)

Similarly,

T(n-2) = T(n-3) + log2 (n-2)

On backward substituting value of T(n-1) and T(n-2) in T(n) we get:

T(n) = T(n-1) + log2 n

T(n) = T(n-2) + log2 (n-1) + log2 n

T(n) = T(n -2) + log2 (n-1) + log2 n

T(n) = T(n-3) + log2 (n-2) + log2 (n-1) + log2 n

T(n) = T(n-3) + log2 (n-2) + log2 (n-1) + log2 n

For the kth iteration

T(n) = T(n-k) + log2 (n-k) + … + log2 (n-2) + log2 (n-1) + log2 n

If we kept on repeating this, then we will reach a point such that T(0) for some iteration.

Then, n – k = 0

Implies that, n = k

Therefore, we’ll get T(0) when n = k. On substituting the values,

T(n) = T(n-n) + log2 (1) + … + log2 (n-2) + log2 (n-1) + log2 n

T(n) = T(0) + log2 (1 \* 2 \* 3 \*…\* (n-2) \* (n-1) \* (n))

{as log a + log b = log a\*b}

T(n) = 1 + log2 n! {T(0) is constant, and so assuming T(0) = 1}

Therefore, **the bound for T(n) = T(n -1) + log2 n is (log2 (n!))**

**And so, the bound for f(n) = log2 n is (log2 (n!))**

1. if f(n) = n.

**Solution:**

We have, T(n) = T(n -1) + f(n)

T(n) = T(n -1) + n {as f(n) = n.}

Let’s now calculate T(n-1)

T(n-1) = T((n-1) -1) + (n-1)

T(n-1) = T(n-2) + (n-1)

Similarly,

T(n-2) = T(n-3) + (n-2)

On backward substituting value of T(n-1) and T(n-2) in T(n) we get:

T(n) = T(n-1) + n

T(n) = T(n-2) + (n-1) + n

T(n) = T(n -2) + (n-1) + n

T(n) = T(n-3) + (n-3) + (n-1) + n

For the kth iteration

T(n) = T(n-k) + (n-k) + …. + (n-3) + (n-1) + n

If we kept on repeating this, then we will reach a point such that T(0) for some iteration.

Then, n – k = 0

Implies that, n = k

Therefore, we’ll get T(0) when n = k. On substituting the values,

T(n) = T(n-n) + 0 + 1 + 2 …. + (n-1) + n

T(n) = T(0) + n (n + 1) / 2 {sum of the first n numbers is }

T(n) = 1 + (n2 + n )/2 {T(0) is constant, and so assuming T(0) = 1}

Therefore, **the bound for T(n) = T(n -1) + c is (n2)**

**And so, the bound for f(n) = n is (n2)**

1. if f(n) = na .

**Solution:**

We have, T(n) = T(n -1) + f(n)

T(n) = T(n -1) + na {as f(n) = na.}

Let’s now calculate T(n-1)

T(n-1) = T((n-1) -1) + (n-1)a

T(n-1) = T(n-2) + (n-1)a

Similarly,

T(n-2) = T(n-3) + (n-2)a

On backward substituting value of T(n-1) and T(n-2) in T(n) we get:

T(n) = T(n-1) + na

T(n) = T(n-2) + (n-1)a + na

T(n) = T(n-3) + (n-2)a + (n-1)a + na

For the kth iteration

T(n) = T(n-k) + (n-k)a + … + (n-2)a + (n-1)a + na

If we kept on repeating this, then we will reach a point such that T(0) for some iteration.

Then, n – k = 0

Implies that, n = k

Therefore, we’ll get T(0) when n = k. On substituting the values,

T(n) = T(n-n) + (0)a  + (1)a + … + (n-2)a + (n-1)a + na

T(n) = T(0) + (na+1 - 1) / (a+1)

{The sum of the series 0^a + 1^a + 2^a + ... + (n-2)^a + (n-1)^a + n^a is given by the following formula: (n^a+1 - 1) / (a+1) }

{Also, a hypothesis that sum of first n numbers raised to power one is n(n+1)/2 and so highest power is 2. For sum of first n numbers raised to power 2, the highest power is n cube, similarly sum of first n numbers raised to power of a, the highest power should be a+1}

T(n) = 1 + (na+1 - 1) / (a+1) {T(0) is constant, and so assuming T(0) = 1}

Therefore, **the bound for T(n) = T(n -1) + c is (na+1)**

**And so, the bound for f(n) = na is (na+1)**

1. if f(n) = an.

**Solution:**

We have, T(n) = T(n -1) + f(n)

T(n) = T(n -1) + an {as f(n) = an.}

Let’s now calculate T(n-1)

T(n-1) = T((n-1) -1) + an-1

T(n-1) = T(n-2) + an-1

Similarly,

T(n-2) = T(n-3) + an-2

On backward substituting value of T(n-1) and T(n-2) in T(n) we get:

T(n) = T(n-1) + an

T(n) = T(n-2) + an-1 + an

T(n) = T(n -2) + an-1 + an

T(n) = T(n-3) + an-2 + an-1 + an

For the kth iteration

T(n) = T(n-k) + an-k +…+ an-1 + an

If we kept on repeating this, then we will reach a point such that T(0) for some iteration.

Then, n – k = 0

Implies that, n = k

Therefore, we’ll get T(0) when n = k. On substituting the values,

T(n) = T(n-n) + a0 + a1 + a2 +…+ an-1 + an

T(n) = T(0) + (a^{n+1} - 1) / (a - 1)

{The sum of the series a0+a1+a3+...+an−2+an−1+an is given by the following formula: (a^{n+1} - 1) / (a - 1)}

T(n) = 1 + (a^{n+1} - 1) / (a - 1)

{T(0) is constant, and so assuming T(0) = 1}

Therefore, **the bound for T(n) = T(n -1) + c is (an+1)**

**And so, the bound for f(n) = an is (an+1)**

1. Solve T(n) = 2T(n-1) + c, where c is a constant

**Solution:**

Given the recurrence relation T(n) = 2T(n-1) + c

Let’s now calculate T(n-1)

T(n-1) = 2T((n-1)-1) + c

T(n-1) = 2T(n-2) + c

Similarly,

T(n-2) = 2T(n-3) + c

On backward substituting value of T(n-1) and T(n-2) in T(n) we get:

T(n) = 2T(n-1) + c

T(n) = 2[2T(n-2) + c] + c

T(n) = 22 T(n-2) +2c + c

T(n) = 22 [2T(n-3) + c] + 2c + c

T(n) = 23 T(n-3) + 22c + 2c + c

For the kth iteration

T(n) = 2k T(n-k) + 2k-1c + 2k-2c + …… 22c + 2c + c

If we kept on repeating this, then we will reach a point such that T(0) for some iteration.

Then, n – k = 0

Implies that, n = k

Therefore, we’ll get T(0) when n = k. On substituting the values,

T(n) = 2n T(n-n) + c + 21c + 22 c +…. + 2n-2c + 2n-1c

T(n) = 2n T(0) + c + 21c + 22 c +…. + 2n-2c + 2n-1c

T(n) = 2n + c + 21c + 22 c +…. + 2n-2c + 2n-1c

{T(0) is constant, and so assuming T(0) = 1}

Let’s assume c as 1 for simpler calculations.

T(n) = 2n + (1) + 21(1) + 22(1) +…. + 2n-2(1) + 2n-1(1)

We have a geometric series, and whose sum is given by 1 + 21+ 22+…. + 2n-2 + 2n-1 as (2n -1)

Therefore,

T(n) = 2n + 2n -1

T(n) = 2(2n) – 1

T(n) = 2n+1 -1

Therefore, **the bound for T(n) = 2T(n -1) + c is (2n+1)**

**And so, the bound for T(n) = 2T(n-1) + c is (2n+1)**