CSE 321 - Homework 1

Due date: 31/10/2022, 16:45

1. **20pts.** Sort the following functions in increasing order of asymptotic growth. Prove that $T_i(n) \in O(T_j(n))$ for every $T_i < T_j$ where $1 \leq i, j \leq 8$ by using limit approach.

- $T_1(n) = 3logn + 3$
- $T_2(n) = 4log(logn)$
- $T_3(n) = n^5 + 8n^4$
- $T_4(n) = 2000n + 1$
- $T_5(n) = (\frac{n}{6})^2$
- $T_6(n) = 3^n + n^2$
- $T_7(n) = n^n + 1000n$
- $T_8(n) = 2^n + n^3$

Solution: $T_2 < T_1 < T_4 < T_5 < T_3 < T_8 < T_6 < T_7$

To prove it, we will use the limit method as seen below.

$f(n) \in O(g(n)) \longrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in \Omega(g(n)) \longrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$
$f(n) \in \Theta(g(n)) \longrightarrow 0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$

Since we are trying to prove $T_i(n) \in O(T_j(n))$, we should either show that $\lim_{n\to\infty} \frac{T_i(n)}{T_j(n)} = 0$ or $\lim_{n\to\infty} \frac{T_j(n)}{T_i(n)} = \infty$.

- *Proving* $T_2 < T_1$:

$\lim_{n\to\infty} \frac{T_2(n)}{T_1(n)} = \lim_{n\to\infty} \frac{4*log(logn))}{3*logn+3} = \frac{\infty}{\infty} \xrightarrow{L'hospital} \lim_{n\to\infty} \frac{4*\frac{1}{n*ln2}}{3*\frac{1}{n*ln2}} = \lim_{n\to\infty} \frac{4}{3*logn*ln2} = 0$

- *Proving* $T_1 < T_4$:

$\lim_{n\to\infty} \frac{T_1(n)}{T_4(n)} = \lim_{n\to\infty} \frac{3*logn+3}{2000*n+1} = \frac{\infty}{\infty} \xrightarrow{L'hospital} \lim_{n\to\infty} \frac{3*\frac{1}{n*ln2}}{2000} = 0$

- *Proving* $T_4 < T_5$:

$\lim_{n\to\infty} \frac{T_4(n)}{T_5(n)} = \lim_{n\to\infty} \frac{2000*n+1}{(\frac{n}{6})^2} = \frac{\infty}{\infty} \xrightarrow{L'hospital} \lim_{n\to\infty} \frac{2000}{2*n} = 0$

- *Proving* $T_5 < T_3$:

  $\lim_{n\to\infty} \frac{T_5(n)}{T_3(n)} = \lim_{n\to\infty} \frac{(\frac{n}{6})^2}{n^5+8*n^4} = \frac{\infty}{\infty} \quad \xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{\frac{n}{18}}{5*n^4+32*n^3} = \frac{1}{18*(5*n^3+32*n^2)} = 0$

- *Proving* $T_3 < T_8$:

  $\lim_{n\to\infty} \frac{T_3(n)}{T_8(n)} = \lim_{n\to\infty} \frac{n^5+8*n^4}{2^n+n^3} = \frac{\infty}{\infty} \quad \xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{5*n^4+32*n^3}{2^n*ln2+3*n^2} = \frac{\infty}{\infty}$

  We apply L'hospital again and again, at some point, the divider becomes a constant while the dividend still has 2^n (because the derivative of 2^n contains itself). Since dividend converge to the infinity under the limit and since $\frac{constant}{\infty} = 0$, $\lim_{n\to\infty} \frac{T_5(n)}{T_3(n)} = 0$

- *Proving* $T_8 < T_6$:

  $\lim_{n\to\infty} \frac{T_8(n)}{T_6(n)} = \lim_{n\to\infty} \frac{2^n+n^3}{3^n+n^2} = \frac{\infty}{\infty} \quad \xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{2^n*ln2+3*n^2}{3^n*ln3+2*n} = \frac{\infty}{\infty}$

  $\xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{2^n*ln^2 2+6*n}{3^n*ln^2 3+2} = \frac{\infty}{\infty} \quad \xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{2^n*ln^3 2+6}{3^n*ln^3 3} = \frac{\infty}{\infty}$

  $\xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{2^n*ln^4 2}{3^n*ln^4 3} = \frac{ln^4 2}{ln^4 3} * lim_{n\to\infty}(\frac{2}{3})^n = \frac{ln^4 2}{ln^4 3} * 0 = 0$

- *Proving* $T_6 < T_7$:

  $\lim_{n\to\infty} \frac{T_6(n)}{T_7(n)} = \lim_{n\to\infty} \frac{3^n+n^2}{n^n+1000n} = \frac{\infty}{\infty} \quad \xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{3^n*ln3+2*n}{n^n*(ln(n)+1)+1000} = \frac{\infty}{\infty}$

  $\xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{3^n*ln^2 3+2}{n^n*(ln(n)+1)^2+n^{n-1}} = \frac{\infty}{\infty}$

  $\xrightarrow{L'hospital} \quad \lim_{n\to\infty} \frac{3^n*ln^3 3}{n^n*(\ln(n)+1)^3+3*n^{n-1}*ln(n)+3*n^{n-1}-n^{n-2}}$

  $= ln^3 3 * \lim_{n\to\infty} \frac{3^n*ln^3 3}{n^n*((\ln(n)+1)^3+3*n^{-1}*ln(n)+3*n^{-1}-n^{-2})}$

  $= ln^3 3 * \lim_{n\to\infty}(\frac{3}{n})^n * \lim_{n\to\infty} \frac{1}{(ln(n)+1)^3+3*n^{-1}*ln(n)+3*n^{-1}-n^{-2}} = ln^3 3 * 0 * 0 = 0$

2. **20pts.** For the following functions prove whether $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$ or $f(n) \in \theta(g(n))$ by using limit approach.

   (a) $f(n) = 99n$ and $g(n) = n$

   (b) $f(n) = 2n^4 + n^2$ and $g(n) = (logn)^6$

   (c) $f(n) = \sum_{x=1}^{n} x$ and $g(n) = 4n + logn$

   (d) $f(n) = 3^n$ and $g(n) = 5^{\sqrt{n}}$

Solution: We are using the same method as in Question 1.

$f(n) \in O(g(n)) \longrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in \Omega(g(n)) \longrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$
$f(n) \in \Theta(g(n)) \longrightarrow 0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$

(a) $f(n) = 99n$ and $g(n) = n$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{99n}{n} = \lim_{n\to\infty} 99 = 99$

$0 < c = 99 < \infty$

Therefore $f(n) \in \Theta(g(n))$.

(b) $f(n) = 2n^4 + n^2$ and $g(n) = (logn)^6$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2n^4+n^2}{(logn)^6} = \frac{\infty}{\infty} \xrightarrow{L'hospital} \lim_{n\to\infty} \frac{8*n^3+2*n}{\frac{6*ln^5(n)}{n}} = \lim_{n\to\infty} \frac{8*n^4+4*n^2}{6*ln^5(n)} = \frac{\infty}{\infty}$

We apply L'hospital rule again and again, at some point the dividend becomes a constant while the divisor is a polynomial expression, which ends up as ∞. Therefore $f(n) \in \Omega(g(n))$.

(c) $f(n) = \sum_{x=1}^{n} x$ and $g(n) = 4n + logn$

$\rightarrow \quad f(n) = \sum_{x=1}^{n} = \frac{n*(n+1)}{2}$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n*(n+1)}{2}}{4*n+logn} = \frac{n*(n+1)}{8*n+2logn} = \frac{\infty}{\infty}$

$\xrightarrow{L'hospital} \lim_{n\to\infty} \frac{2*n+1}{8+\frac{2}{n*ln2}} = \frac{(2*n+1)*(n*ln2)}{8*n*ln2+2} = \frac{2*n^2*ln2+n*ln2}{8*n*ln2+2} = \frac{\infty}{\infty}$

$\xrightarrow{L'hospital} \lim_{n\to\infty} \frac{4*n+ln2}{8*ln2} = \infty$

Therefore $f(n) \in \Omega(g(n))$.

(d) $f(n) = 3^n$ and $g(n) = 5^{\sqrt{n}}$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3^n}{5^{\sqrt{n}}} = \lim_{n\to\infty} (\frac{3^{\sqrt{n}}}{5})^{\sqrt{n}} = \infty$

Therefore $f(n) \in \Omega(g(n))$.

3. **20 pts** Examine the following algorithm and answer the questions.

```
int myFunction (int nums[], int n)
{
    for (int i = 0; i < n; i++){
      int count = 1;
      for (int j = i + 1; j < n; j++)
        if (nums[j] == nums[i])
          count++;
      if (count > n / 2)
        return nums[i];
    }
return -1;
}
```

(a) What does the algorithm do? Explain input/output variables.

(b) What is the time complexity of the algorithm? Analyze the worst and best cases.

Solution:

(a) The algorithm takes an array of integers and the size of the array as input. Then it computes the frequency of each unique element in the array. If an element's frequence is larger than half of the size of the array, the algorithm returns that element. If no element holds this condition, the algorithm returns -1.

(b) The best case scenario: The array's first element's frequence is larger than half of the size of the array. Since we are analyzing the best case, assume all of these elements are consecutive in the array (i.e. the first $n/2 + 1$ elements are the same). In this case, checking the first $n/2 + 1$ items would be enough. We ignore constant values while computing complexity. So the best case is in $\Omega(n)$.
On the other hand, the worst case occurs when there is no such element with a frequency larger than $n/2 + 1$. To confirm this, the algorithm has to complete the nested loops. Since both loops have n iterations, the worst time complexity is in $O(n^2)$.

4. **20 pts** Examine the following algorithm and answer the questions.

```
int myFunction2 (int nums[], int n)
{
    int i, *map, max = 0;

    for (i = 0; i < n; i++)
        if (nums[i] > max)
            max = nums[i];

map = (int *) calloc (max + 1, sizeof (int));

    for (i = 0; i < n; i++)
        map[nums[i]]++;

    for (i = 0; i < n; i++)
        if (map[nums[i]] > n / 2)
            return nums[i];
return -1;
}
```

(a) What does the algorithm do? Explain input/output variables.

(b) What is the time complexity of the algorithm? Analyze the worst and best cases.

Solution:

(a) The algorithm takes an array of integers and the size of the array as input. Then it uses a map to keep the frequency of each element in the input array. The algorithm uses the index of the map to keep the frequencies and this operation is being done in a single loop. Then it analyzes the map, if there is a value (frequency) larger than $n/2 + 1$, it returns the index of that element (which is the element we are looking for). If there is none, it returns -1.

(b) Regardless of the best or worst scenario, the algorithm computes the frequency of each element by using a single loop with n iterations. The difference occurs in the last loop. If the first element of the input array has a frequency larger than $n/1 + 1$ this loop would return that element at the first iteration. But since we spent $O(n)$ time just to compute the frequencies, the best time complexity becomes $\Omega(n)$ (The first loop doesn't affect the complexity since it has n iterations as well, not more).

In the worst case, there would not be such an element in the array that holds the frequency condition. So, we complete all loops and spend n time slices for each of them. But since there is no nested loop, we should add the number of iterations (instead of multiplying as in Question 3). Since constant values are ignored, 3n time slices project to $O(n)$).

PS: This solution ignores the complexity of the calloc function since the size of the map is unknown. But if we take it into consideration and if the map is too large, then the time complexity could be larger.

5. **10pts.** Compare the algorithms in Question 3 and 4 in terms of time complexity and space used. Explain what makes them better.

Solution: Both algorithms return the same output for the same inputs, i.e. they do the same thing by using different methods.

Time analysis: The algorithm in Question 3, go through the whole array for each element. The algorithm in Question 4, saves the frequencies to not compute them again and again. In the best case, both have the same complexity. But in the worst case, the algorithm in Question 4 is significantly faster.

Space analysis: The algorithm in Question 3 uses only an integer to keep the frequencies of the elements. But the algorithm in Question 4, saves a map to keep all of these values. In this algorithm, the length of this map can be as large as the maximum element of the input array. By smart coding, it could be much less, but not less than an array with n elements. As a result, the algorithm in Question 3, is much better in terms of space used.

6. **10pts.** Consider you are given 2 arrays as follows: $A = [a_1, a_2, ..., a_n]$ and $B = [b_1, b_2, ..., b_m]$ Describe an algorithm for each of the following problems. Write the pseudo-code of the algorithm and analyze the time complexity of the worst and best cases.

   (a) Finding $max\{a_i * b_j\}$ where $1 \leq i \leq n$ and $1 \leq j \leq m$.

   (b) Sorting all elements of A and B in descending order as a single array, e.g. $\{a_3, b_1, b_7, ...\}$

   (c) Adding an element to one of the arrays.

   (d) Deleting an element from one of the arrays.

Solution:

(a) The following algorithm spends n time slices for any case because it has to find the maximum element of each array. To do this, it has to check each element in that array. Therefore the best and the worst cases are the same: $O(k)$ where $k = max(n, m)$.

```
max_multiplication (A, B)
max_a ← 0
max_b ← 0
for each element in A
        if element > max_a then
                max_a ← element
        endif
end for
for each element in B
        if element > max_b then
                max_b ← element
        endif
end for
return max_a*max_b
```

(b) The following algorithm concatenates two arrays and then sorts the new array by using the insertion sort algorithm. Concatenating two arrays will take $O(k)$ time where $k = max(n, m)$ and sorting will take $O((n+m)^2)$ and these values are independent of the inputs, which means the best case and the worst case time complexities are the same: $O((n+m)^2)$.

```
sorting (A, B)
new_array ← []
for each element in A
        append element to new_array
end for
for each element in B
        append element to new_array
end for
for i= 0 to len(new_array) - 1
        current ← new_array[i]
        for j = i+1 to len(new_array)
                if new_array[j]< new_array[i] then
                        swap new_array[i] and new_array[j]
                end if
        end for
end for
return new_array
```

(c) The following algorithm adds a new element to an array. Since we don't know where the new element will be added, we should use an algorithm to cover all possible positions. Copying the input array and inserting the new element could do this with $O(n)$ time complexity in any case (assuming the input array is A).

```
append (A, new_element, position)
new_array ← []
while(i<len(A))
        if i==position then
                append new_element to new_array
        else
                new_array [i] ← new_array[i]
        i ← i +1
        endif
end while
return new_array
```

(d) As in the previous algorithm, we should construct an algorithm to cover all possible inputs. Therefore, copying the array could work here too (by ignoring the specified element). Similarly, the worst and the best case time complexities are $O(n)$ (assuming the input array is A).

```
delete (A, element_to_delete)
new_array ← []
while(i<len(A))
        if A[i] != element_to_delete then
                append A[i] to new_array
                i ← i +1
        endif
end while
return new_array
```

PS: Some problems may have more than a single correct answer, i.e. there are many different ways to construct an algorithm for a given problem as in the last question. Your answer doesn't have to be the same as in this key sheet, it is accepted if correct.