CSE 321 - HomeWork 1

Abdurrahman BUYUT
19010422258

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 ; f(n) \in o(g(n)), & f(n) \text{ is slower} \\ c ; c>0, f(n) \in \Theta(g(n)), & \text{same} \\ \infty ; f(n) \in \omega(g(n)) & f(n) \text{ is faster} \end{cases}$$

* $f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$

---

①

- T_1 and T_2:

$$\lim_{n\to\infty} \frac{3\log_2 n+3}{4\log_2(\log_2 n)} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{\frac{3}{n}\cdot\frac{1}{\ln 2}}{\frac{4}{n\cdot\log_2 n}\cdot\frac{1}{\ln^2(2)}} = \lim_{n\to\infty} \frac{3\cdot\log_2 n\cdot\ln(2)}{4} = \infty, \text{ Thus } T_2 < T_1$$

- T_2 and T_3:

$$\lim_{n\to\infty} \frac{4\log_2(\log_2 n)}{n^5+8n^4} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{4\cdot\frac{1}{n\log_2 n}\cdot\frac{1}{\ln^2(2)}}{5n^4+32n^3} = \lim_{n\to\infty} \frac{\text{constant}}{\cdots\cdots} = 0, \text{ Thus } T_2 < T_3$$

- T_3 and T_4:

$$\lim_{n\to\infty} \frac{n^5+8n^4}{2000n+1} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{5n^4+32n^3}{2000} = \infty, \text{ Thus } T_4 < T_3$$

- T_4 and T_5:

$$\lim_{n\to\infty} \frac{2000n+1}{\left(\frac{n}{6}\right)^2} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{2000}{2\cdot\frac{n}{6}\cdot\frac{1}{6}} = 0, \text{ Thus } T_4 < T_5$$

- T_5 and T_6:

$$\lim_{n\to\infty} \frac{\left(\frac{n}{6}\right)^2}{3^n+n^2} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{n/18}{3^n\ln 3+2n} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{1/18}{3^n\cdot\ln(3)\cdot\ln(3)+\frac{2^x}{x}} = 0,$$

Thus $T_5 < T_6$

- T_6 and T_7:

$T_6(n) = 3^n + (n^2) \rightarrow$ negligible, prove: $\lim_{n\to\infty} \frac{3^n}{n^2} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{3^n\ln 3}{2n} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}}$

$$\lim_{n\to\infty} \frac{3^n\cdot\ln^2(3)}{2} = \infty, \; 3^n \text{ grows faster than } n^2$$

$T_7(n) = n^n + (1000n) \rightarrow$ negligible. prove: $\lim_{n\to\infty} \frac{n^n}{1000n} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{\frac{d}{dn}\left(e^{n\ln(n)}\right)}{1000}$

$$\lim \frac{x^x\cdot(\ln(x)+1)}{1000} = \infty, \; n^n \text{ grows faster than } 1000n$$

So,

$\lim_{n\to\infty} \frac{3^n+n^2}{n^n+100n}$ => simplify => $\lim_{n\to\infty} \frac{3^n}{n^n} = \lim_{n\to\infty} \left(\frac{3}{n}\right)^n$

$= \lim_{n\to\infty} \left(\left(\frac{3}{n}\right)^n\right)$ => $\frac{3}{n}$ goes to 0 (zero), Thus $\lim_{n\to\infty} \frac{T_6}{T_7} = 0$, $T_6 < T_7$

or we can use exponent rule and chain rule => $\lim_{n\to\infty} (e^{n\ln(\frac{3}{n})})$, $\lim_{n\to\infty} g(n) = b \Rightarrow \lim_{n\to\infty} n.\ln(\frac{3}{n}) = -\infty$, and $f(u) = e^u \Rightarrow \lim_{u\to\infty} e^u = 0$ //

T_7 and T_8

$T_7(n) = n^n + 100n$ → negligible. prove: Previous solution.

$T_8(n) = 2^n + n^3$ → negligible. prove: $\lim_{n\to\infty} \frac{2^n}{n^3} \doteq \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{2^n \ln(2)}{3n^2} = \frac{\infty}{\infty}$ → L'Hopital

$= \lim_{n\to\infty} \frac{2^n . \ln^2(2)}{6n} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{2^n . \ln^3(2)}{6n}$ → L'Hopital

$= \lim_{n\to\infty} \frac{2^n . \ln^4(2)}{6} = \infty$, Thus 2^n grows faster than n^3

simplify

$\lim_{n\to\infty} \frac{n^n}{2^n} = \lim_{n\to\infty} \left(\frac{n}{2}\right)^n$ => $\frac{n}{2}$ goes to ∞. $\lim_{n\to\infty} \left(\frac{n}{2}\right)^n = \infty$. Thus. $T_7 > T_8$

prove

Prove: $\lim_{n\to\infty} \left(\left(\frac{n}{2}\right)^n\right) = \lim_{n\to\infty} \left(e^{n\ln(\frac{n}{2})}\right)$ ($g(n) = n\ln(\frac{n}{2})$) = (chain rule) => $\lim_{n\to a} g(n) = b \Rightarrow \lim_{n\to\infty} n\ln(n) = \infty$

Set $g(n) = u$

$f(u) = e^u \Rightarrow \lim_{u\to\infty} e^{\infty} = \infty$

current result: $T_2 < T_1$

$T_2 < T_3$

$T_4 < T_3$

$T_4 < T_5 < T_6 < T_7$

$T_8 < T_7$

* T_1 and T_3:

$$\lim_{n\to\infty} \frac{3\log_2 n + 3}{n^5 + 8n^4} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{\frac{3}{n}\cdot\frac{1}{\ln 2}}{5n^4 + 32n^3} = 0, \text{Thus } T_1 < T_3$$

* T_1 and T_4:

$$\lim_{n\to\infty} \frac{3\log_2 n + 3}{2000n + 1} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{\frac{3}{n}\cdot\frac{1}{\ln 2}}{2000} = 0, \text{Thus } T_1 < T_4$$

current result: $T_2 < T_1 < T_4 < T_3$

$T_4 < T_5 < T_6 < T_7$

$T_8 < T_7$

* T_3 and T_5

$$\lim_{n\to\infty} \frac{n^5 + 8n^4}{(\frac{n}{6})^2} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{5n^4 + 32n^3}{n/18} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{20n^3 + 96n^2}{1/18} = \infty, \text{Thus, } T_3 > T_5$$

current result:

$T_2 < T_1 < T_4 < T_5 < T_3$

$T_6 < T_7$

$T_8 < T_7$

* T_6 and T_8

$$\lim_{n\to\infty} \frac{3^n + n^2 \to \text{negligible}}{2^n + n \to \text{negligible}} \Rightarrow \lim_{n\to\infty} \frac{3^n}{2^n} = \lim_{n\to\infty} \left(\frac{3}{2}\right)^n = \infty, \text{ Thus, } T_6 > T_8$$

* T_3 and T_8

$$\lim_{n\to\infty} \frac{n^5 + 8n^4 \to \text{negligible}}{2^n + n^3 \to \text{negligible}} \Rightarrow \lim_{n\to\infty} \frac{n^5}{2^n} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{5n^4}{2^n \ln(2)} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{20n^3}{2^n \cdot \ln^2(2)}$$

$= 4$ more L'Hopital $\Rightarrow \lim_{n\to\infty} \frac{120}{2^n \cdot \ln^4(2)} = 0$, Thus, $T_3 < T_8$

Final Result is: $T_2 < T_1 < T_4 < T_5 < T_3 < T_8 < T_6 < T_7$

Final Result is: $T_2 < T_1 < T_4 < T_5 < T_3 < T_8 < T_6 < T_7$

$f(n) = 99n$, $g(n) = n$

(2)

a) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{99n}{n} = \lim_{n\to\infty} 99 = 99$, constant and $c > 0$, Thus

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

$f(n) = 2n^4 + n^2$, $g(n) = (\log n)^6$

b) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2n^4+n^2}{(\log_2 n)^6} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{8n^3+2n}{\underbrace{6(\log_2(n))^5}_{\ln(2).n}} = \lim_{n\to\infty} \frac{(8n^4+2n^2).\ln(2)}{6(\log_2(n))^5} \Rightarrow$

$= \underbrace{\frac{2.\ln(2)}{6}}_{c} . \lim_{n\to\infty} \frac{(4n^4+n^2)}{(\log_2(n))^5} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} c.\lim_{n\to\infty} \frac{16n^3+2n}{\underbrace{5.(\log_2(n))^4}_{\ln(2).n}} = c.\lim_{n\to\infty} \frac{\overbrace{2.(16n^4+n^2).\ln(2)}^{\text{constant}}}{5.(\log_2(n))^4} \Rightarrow$

$= c.\lim_{n\to\infty} \frac{16n^4+n^2}{(\log_2(n))^4} \Rightarrow$ 3 L'Hopital repeatedly $\Rightarrow c_1.\lim_{n\to\infty} \frac{c_2.n^4+n^2}{\log_2(n)} = \frac{\infty}{\infty} == \xrightarrow{\text{L'Hopital}}$

$= c_1.\lim_{n\to\infty} \frac{4.c_2.n^3+2n}{\frac{1}{x.\ln(2)}} = c_1.\lim_{n\to\infty} (c_2.n^4+n^2).(\ln 2) = \infty$, Thus $f(n) \in \Omega(g(n))$ (Big Omega)

---

c) $f(n) = \sum_{x=1}^{n} x = 1+2+\dots+n = \frac{n.(n+1)}{2} = \frac{n^2+n}{2}$, $g(n) = 4n+\log n$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\frac{n.(n+1)}{2}}{4n+\log_2 n} = \frac{n^2+n}{8n+2\log_2 n} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{2n+1}{8+\frac{1}{n.\ln(2)}} = \lim_{n\to\infty} \frac{2n+1}{\underbrace{8n+\log_2 e}_{\uparrow}}$

$= \lim_{n\to\infty} \frac{2n^2+n}{8n+\log_2 e} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hopital}} \lim_{n\to\infty} \frac{4n+1}{8} = \infty$, Thus $f(n) \in \Omega(g(n))$ (Big Omega)

---

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

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3^n}{5^{\sqrt{n}}} = \lim_{n\to\infty} \frac{(3^{\sqrt{n}})^{\sqrt{n}}}{5^{\sqrt{n}}} = \lim_{n\to\infty} \left(\frac{3^{\sqrt{n}}}{5}\right)^{\sqrt{n}} = \lim_{n\to\infty} e^{\sqrt{n}.\ln\left(\frac{3^{\sqrt{n}}}{5}\right)} \Rightarrow$

$\sqrt{n} \to \infty$, $\ln\left(\frac{3^{\sqrt{n}}}{5}\right) \to \infty$

$\lim_{n\to\infty} e^{\infty.\infty} = \lim_{n\to\infty} e^{\infty} = \infty$, Thus $f(n) \in \Omega(g(n))$ (Big Omega)

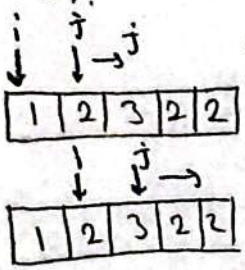
③
```
int myFunction(int nums[], int n)
{
  for (int i=0; i<n; i++){          → O(n)
    int count=1;                    → O(1)
    for(int j=i+1; j<n; j++)        → O(n)
      if(nums[j]==nums[i])          } O(1)
        count++;
    if (count > n/2)                } O(1)
      return nums[i];
  }
return -1;                          → O(1)
}
```

O(n²)

a) There is a function which takes an integer array and size of that array as input variables.

The algorithm returns the element in an array that repeats more than half the number of elements of the array. If it cannot find such an element, it returns -1.

Example



→ j goes one by one on the array. There is no other 1 (one) in array. Then i and j are shifting to the right

→ it will find 2 elements 2. and the counting variable was 1 before. Then it makes 3. 3 is bigger than half of the length of the array. So, it returns 3.

b) Best case: Occurs if the first element repeats more than half the length of the list. The algorithm has to take at least one round over the array. $B(n) = n \in \Theta(n)$

worst case: Occurs when there are no more than half the number of repeating elements in the array. The inner loop has to work up to length.

$$W(n) = n^2 \in \Theta(n^2)$$

Time Complexity: Outer loop executes n+1 time, inner loop executes n-i times..

Complexity is $n \times (n-1) = n^2 \Rightarrow O(n^2)$

4)

```
int myFunction2(int nums[], int n)
{
  int i, *map, max=0;                          → O(1)

  for(i=0; i<n; i++)                           → O(n)
    if(nums[i] > max)                          → O(1)
      max = nums[i];                           → O(1)

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

  for(i=0; i<n; i++)                           → O(n)
    map[nums[i]]++;                            → O(1)

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

  return -1;                                   → O(1)
}
```

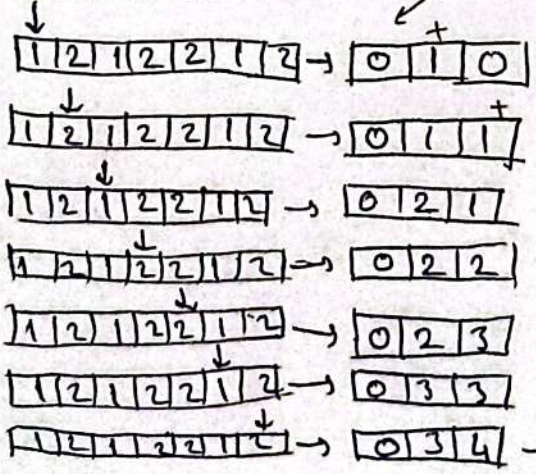
It does the same thing as before.

a) This function takes an array and the length of that array(n). Firstly, it finds the max element inside of the array. It creates another array similar to a table. It stores the number of each element in array. Each index of map array, keeps the number of repeated elements at other array

Example:

nums array: {1,2,1,2,2,1,2}

maps array: [0][0][0] initialize 0



Number of element 0 → 0
Number of element 1 → 3
Number of element 2 → 4
Length of array : 7
7/2 => 3
Since 4>3, it will return 2 as result.

(5)

Best case: It has to loop over the array at least one.

There are 3 loops in the algorithm. There is only one condition and it's on the third place. Even if the result is the first element, it has to do other loops.

So, $B(n) = n \in \Theta(n)$

worst case: whether or not the array has more than half the number of repeating elements, it can circulates on the array $\Theta(n)$ time. So,

$W(n) = n \in \Theta(n)$

Time complexity for this algorithm is linear Time

---

(5) Where as time complexity measures the time to run program, space complexity measures memory usage. Assigning variables, creating new data structures and Function calling and allocation are increase space complexity.

If we look at the time complexity, we can say that the second algorithm is better. Because, second algorithm has always $\Theta(n)$ complexity. But if we look at the first algorithm, we see that its only working $\Theta(n)$ in the best case scenario.

If we compare them by looking at the space complexity, Since second algorithm has more number of assignment, and has newly created data structure in it, first algorithm is better than second in terms of space complexity. We can say that, the second algorithm uses more space than first algorithm.

If we have enough memory space and need something that needs to be done quickly, it might make sense to use the second algorithm. The second algorithm runs faster due to low time complexity.

If we dont have enough memory space or if we have a limited amount memory space or speed is not a priority for us, we can use first algorithm.

⑥ $A = [a_1, a_2, \dots a_n]$ pseudo-code or (python)

$B = [b_1, b_2, \dots b_n]$ [+] will write python code.

a) find $\max[a_i * b_j]$

```
def find_max(array1, array2):
    if len(array1) and len(array2):        } O(n)
        return -1

    max = array1[0] * array2[0]            → O(1)
    for i in range(len(array1)):           → n   } n×m = O(n²)
        for j in range(len(array2)):       → m   }
            if max < array1[i] * array2[j]:     } O(1)
                max = array[i] * array2[j]
    return max                             → O(1)
```

=> It has to circulate in both two loops. No matter what they have, all elements in any array should multiply with corresponding elements in other array. So,

$B(n) = n^2 \in \Theta(n^2)$

$W(n) = n^2 \in \Theta(n^2)$

*Not. Len() func works constant time.

inputs: Two array

outputs: maximum multiplication /variable

---

b) ~~def~~ Sorting

```
def my_sort(arr1, arr2):
    result = []
    for i in range(len(arr1)):             → O(n)
        result.append(arr1[i])             → O(1)
    for i in range(len(arr2)):             → O(n)
        result.append(arr2[i])             → O(1)

    for i in range(len(result)):           → n     } n×(n-i)
        for j in range(i, len(result)):    → n-i   } = O(n²)
            if result[i] < result[j]:      → O(1)
                temp = result[i]           }
                result[i] = result[j]      } O(1)
                result[j] = temp
    return result                          → O(1)
```

=> Best case occurs when both two arrays are already sorted. The complexity of combining two lists is $\Theta(n)$. If two lists are sorted both within and among themselves,

$B(n) = n^2 = \Theta(n^2)$

Worst case occurs if the inputs are already sorted in reverse order.

$W(n) = \sum_{i=0}^{n} (i-1) = \frac{n.(n-1)}{2} \in \Theta(n^2)$

inputs: Two arrays

output: a combined array

(c) Adding

```
def my_add(arr, index, element=0):
    if index<0 or index>len(arr)-1:        } O(1)
        return -1
    result = [None]*(len(arr)+1)  → O(1)
    flag = 0                      → O(1)
    for i in range(len(arr)):     → O(n)
        if i==index:
            result[i+1]=arr[i]
            result[i] = element
            flag = 1                       } O(1)
            continue
        result[i+flag] = a[i]
    return result                 → O(1)
```

=> This algorithm adds an element to an array by looking the index. As default, it adds to first index. There is a newly created list which is length of one more as input array. flag variable is to make shifting. it will shift to right when it reaches to the index. It will circulates in input array.

So,

$B(n) = n \in O(n)$

$W(n) = n \in O(n)$

Inputs: array, index for elements, adding element

Output: an array after adding operation

(d) Deleting

```
def my_delete(arr, element):
    result = [None]*(len(arr)-1) → O(1)
    flag = 0
    for i in range(len(arr)):  → O(n)
        if arr[i]==element:
            flag = 1                       } O(1)
            continue
        arr[i-flag] = arr[i]
    if flag==1:
                                  → O(1)
        arr[len(arr)-1] = None
        for i in range(len(result)): → O(n)
            result[i] = arr[i]         } O(1)
        return result
    return arr                    → O(1)
```

=> This algorithm takes an array and an element. It will find the element in the array if the element exists. Firstly, it delete that element and shift to left. After that, If that element is found, All other elements are assigned to newly created array. It has to move on array for shifting. So,

$B(n) = n \in O(n)$

$W(n) = n \in O(n)$

Inputs: an array
an element to delete

Outputs: If the element exists, an array which comes from input.

if the element does not exist, a newly created array in deleted form.