CSE 321 - Homework 1

1. → Eliminate low order terms

→ 〃 constant factors

$T_1(n) = \cancel{3}\log n + \cancel{3} \in \Theta(\log n)$

$T_2(n) = \cancel{4}\log(\log n) \in \Theta(\log\log n)$

$T_3(n) = n^5 + \cancel{8n^4} \in \Theta(n^5)$

$T_4(n) = \cancel{2000}n + \cancel{1} \in \Theta(n)$

$T_5(n) = \left(\frac{n}{6}\right)^2 = \cancel{\frac{1}{6^2}}\,n^2 \in \Theta(n^2)$

$T_6(n) = 3^n + \cancel{n^2} \in \Theta(3^n)$

$T_7(n) = n^n + \cancel{1000n} \in \Theta(n^n)$

$T_8(n) = 2^n + \cancel{n^3} \in \Theta(2^n)$

Therefore; $T_2(n) < T_1(n) < T_4(n) < T_5(n) < T_3(n) < T_8(n) < T_6(n) < T_7(n)$ ①

$T_2 - T_1$

$$\lim_{n\to\infty} \frac{4\log(\log n)}{3\log n + 3} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{\frac{4}{n\log n}}{\frac{3}{n}} = \lim_{n\to\infty} \frac{4n}{3n\log n} = \lim_{n\to\infty} \frac{4}{3\log n + 3} = 0$$

$\therefore T_2(n) \in O(T_1(n))$

$T_1 - T_4$

$$\lim_{n\to\infty} \frac{3\log n + 3}{2000n + 1} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{3}{2000n} = 0 \quad \therefore T_1(n) \in O(T_4(n))$$

$T_4 - T_5$

$$\lim_{n\to\infty} \frac{2000n+1}{\left(\frac{n}{6}\right)^2} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{2000}{\frac{2}{36}n} = 0 \quad \therefore T_4(n) \in O(T_5(n))$$

$T_5 - T_3$

$$\lim_{n\to\infty} \frac{\left(\frac{n}{6}\right)^2}{n^5 + 8n^4} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{\frac{2n}{36}}{5n^4 + 32n^3} = \lim_{n\to\infty} \frac{2}{36(4n^3 + 96n^2)} = 0 \quad \therefore T_5(n) \in O(T_3(n))$$

$T_3 - T_8$

$$\lim_{n\to\infty} \frac{n^5 + 8n^4}{2^n + n^3} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{5n^4 + 32n^3}{2^n \ln 2 + 3n^2} = \lim_{n\to\infty} \frac{20n^3 + 96n^2}{2^n(\ln 2)^2 + 6n} = \lim_{n\to\infty} \frac{60n^2 + 192n}{2^n(\ln 2)^3 + 6}$$

$$= \lim_{n\to\infty} \frac{120n + 192}{2^n(\ln 2)^4} = \lim_{n\to\infty} \frac{120}{2^n(\ln 2)^5} = 0 \quad \therefore T_3(n) \in O(T_8(n))$$

$T_8 - T_6$

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

$$= \lim_{n\to\infty} \left(\frac{2}{3}\right)^n \left(\frac{\ln 2}{\ln 3}\right)^4 = 0 \quad \therefore T_8(n) = O(T_6(n))$$

$T_6 - T_7$

$$\lim_{n\to\infty} \frac{3^n + n^2}{n^n + 1000n} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{3^n(\ln 3) + 2n}{n^n(\ln n + 1) + 1000} = \lim_{n\to\infty} \frac{3^n(\ln 3)^2 + 2}{n^n(\ln n + 1)^2 + n^{n-1}}$$

$$= \lim_{n\to\infty} \left(\frac{3}{n}\right)^n = 0 \quad \therefore T_6(n) \in O(T_7(n))$$

2) a) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{99n}{n} = \underbrace{99}_{\text{constant}} \therefore f(n) \in \Theta(g(n))$

b) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2n^4+n^2}{(\log n)^6} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{8n^3+2n}{\frac{6(\log n)^5}{n}} = \lim_{n\to\infty} \frac{8n^4+2n^2}{6(\log n)^5}$

$= \lim_{n\to\infty} \frac{32n^4+4n^2}{30(\log n)^4} = \lim_{n\to\infty} \frac{128n^4+8n^2}{120(\log n)^3} = \lim_{n\to\infty} \frac{512n^4+16n^2}{360(\log n)^2} = \lim_{n\to\infty} \frac{2048n^4+32n^2}{720(\log n)}$

$= \lim_{n\to\infty} \frac{8192n^4+64n^2}{720} = \infty \therefore f(n) \in \Omega(g(n))$

c) $f(n) = \sum_{x=1}^{n} x = 1+2+\dots+n = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2+n}{8n+2\log n} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{2n^2+n}{8n+2} = \infty \therefore f(n) \in \Omega(g(n))$

d) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3^n}{5^{\sqrt{n}}} = \frac{\infty}{\infty} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{(3^n \ln 3)2\sqrt{n}}{5^{\sqrt{n}} \ln 5} = \lim_{n\to\infty} \frac{\frac{(2\ln^2 3\, n + \ln 3)3^n}{\sqrt{n}}}{\frac{(\ln 5)^2\, 5^{\sqrt{n}}}{2\sqrt{n}}}$

$= \lim_{n\to\infty} \frac{(4(\ln 3)^2 n + \ln 3)\, 3^n}{(\ln 5)^2\, 5^{\sqrt{n}}} = \lim_{n\to\infty} \frac{(\ln 3)^2(4\ln 3\, n + 5)\, 3^n\, 2\sqrt{n}}{(\ln 5)^3\, 5^{\sqrt{n}}} = \infty \therefore f(n) \in \Omega(g(n))$

3) a) Counts the number of occurences of each element in the array. If the number of occurences of an element in the array exceeds the half of the array length, it returns that element. I.e it finds the dominant element in the array.

Inputs: Array of numbers, length of the array

Output: The dominant element in the array if it exists, otherwise -1.

b) Worst case = There is no such element in the array satisfies the condition count $> n/2$. It traverses all the elements and counts them.

$T_w = (n-1) + (n-2) + \dots + 1 = \frac{(n-1)n}{2} = \frac{n^2-n}{2} \in \Theta(n^2)$

Best case = The first element in the array satisfies the condition count $> n/2$.

$T_b = (n-1) \in \Theta(n)$

4) a) First, it finds the highest number in the array. Then allocates an integer array of max+1 size. Counts the number of occurences of each element in the array and saves the results to allocated array. Then, it checks if any of the occurences exceeds the half of the array size. If any of them satisfies the condition, it returns that element, otherwise returns -1.

Inputs: Array of numbers, length of the array.

Output: The dominant element in the array if it exists, otherwise -1.

b) Worst case = Best case; since it will traverse the array to find the max element, and to find the occurences of each element in all cases. The last loop can depend on either the dominant element is the first element in the map array or it does not exists. But either way the last loop will not change the complexity of the algorithm.

$T_w = n+n+n = 3n \in \Theta(n)$

$T_b = n+n+1 = 2n+1 \in \Theta(n)$

5) Time complexity: They both have the same time complexity at best case, but in worst case, the first algorithm is worse than the second one $[O(n) \subset O(n^2)]$

$$T_{w_1} \in \theta(n^2), \quad T_{w_2} \in \theta(n) \quad > \quad T_{w_1} \in \omega(T_{w_2})$$

Space used:

Algoritm 1: nums[ ] => n space

n, i, count, j => 4 space

Total = n + 4 space $\in \theta(n)$

Algorithm 2: nums[ ] => n space

n, i, map, max = 4 space

map[ ] => max + 1 space

Total = max + n + 5 space $\in \theta(max + n)$

In terms of time complexity, the second algorithm is better for worst case scenario. If we compare the space usage, it depends on the max element in the array. If it is larger than array length, the first algorithm is better, otherwise the second algorithm is better for space usage.

6) a)

```
int findMax(int arrA[], int n, int arrB[], int m) {
    int maxA = 0, maxB = 0;
    for (int i = 0; i < n; ++i) {   // Find maxA      } n iteration
        if (arrA[i] > maxA)
            maxA = arr[i];
    }
    for (int i = 0; i < m; ++i) {   // Find maxB      } m iteration
        if (arrB[i] > maxB)
            maxB = arrB[i];
    }
    return maxA * maxB;
}
```

Worst case = best case since it will traverse both arrays no matter what.

Tw = Tb = n + m $\in \theta(n+m)$

b)

```
int* sortAndMerge(int arrA[], int n, int arrB[], int m) {
    // Bubble sort array A
    for (int i = 0; i < indexOfLastUnsortedElement - 1 && swapped; ++i) {
        swapped = false
        if (arrA[i] < arrA[i+1]) {
            swap(arr[i], arr[i+1]);
            swapped = true;
        }
    }
    // Bubble sort array B
    for (int i = 0; i < indexOfLastUnsortedElement - 1 && swapped; ++i) {
        swapped = false;
        if (arrB[i] < arrB[i+1]) {
            swap(arrB[i], arrB[i+1]);
            swapped = true;
        }
    }
```

```
// Merge both sorted arrays into one array
int i=0, j=0, k=0;
int arrC[m+n];
while ( k < m+n && i<n && j<m) {
    if( arrA[i] > arrB[j] ) {
        arrC[k] = arrA[i]
        i++;
    }
    else {
        arrC[k] = arrB[j];
        j++;
    }
}
if ( i == n) {
    while ( j < m)
        arrC[k] = arrB[j];
}
else if (j == m) {
    while ( i < n)
        arrC[k] = arrA[i];
}
return arrC;
```

m+n iterations

Worst case:

The input arrays are sorted in increasing order.

$Tw = \frac{n(n-1)}{2} + \frac{m(m-1)}{2} + m + n \in \Theta(n^2 + m^2)$

Best case:

The input arrays are already sorted in decreasing order.

$Tb = n - 1 + m - 1 + m + n = 2m + 2n - 2 \in \Theta(m+n)$

c)
```
int* addElement ( int arr[], int n, int index, int elem ) {
    if ( index < 0 || index > n)
        throw
    int* newArr = (int *) calloc (n+1, sizeof(int));
    int i = 0
    while ( i < index) {
        newArr[i] = arr[i];
        i++;
    }
    newArr[i] = elem;
    while ( i < n) {
        newArr[i + 1] = arr[i];
        i++;
    }
    return newArr;
}
```

index iterations

(n - index) iterations

Worst case = Best case = $n - \cancel{index} + \cancel{index} = n \in \Theta(n)$

d)
```
int* deleteElement( int arr[ ], int n, int index) {
    int* newArray = (int *) calloc (n - 1 , sizeof (int));
    int i = 0;
    while ( i < index) {
        newArr [i] = arr [i];          index iterations
        i++;
    }
    while ( i < n - 1 ) {
        newArr [i] = arr [i + 1];      (n - 1 - index) iterations
        i++;
    }
    return newArr;
}
```

Worst case = Best case = $\cancel{index} + n - 1 - \cancel{index} = n - 1 \in \Theta(n)$