Mehmet Mete Şamlıoğlu
200104004093

1-)

a-) $f(n) = n^2+7n$ and $g(n) = n^3+7$

Take the limit of $f(n)/g(n)$

$$\lim_{N\to\infty} \frac{f(n)}{g(n)} = \lim_{N\to\infty} \frac{n^2+7n}{n^3+7} = \lim_{N\to\infty} \frac{\cancel{n^2}\left(1+\frac{7}{n}\right)}{\cancel{n^3}\left(1+\frac{7}{n^3}\right)} = \lim_{N\to\infty} \frac{\left(1+\cancel{\frac{7}{\infty}}^{\,0}\right)}{\infty\left(1+\cancel{\frac{7}{\infty^3}}^{\,0}\right)} = \lim_{N\to\infty} \frac{1}{\infty} = 0$$

Since $\lim_{N\to\infty} \frac{f(n)}{g(n)} = 0$, $g(n)$ is upper bound for $f(n)$.

$f(n) = O(g(n))$

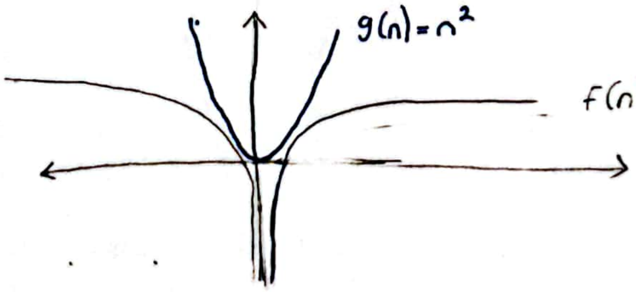
b-) $f(n) = 12n + \log_2 n^2$ and $g(n) = n^2+6n$

Take the limit of $f(n)/g(n)$

$$(*)\ \lim_{N\to\infty} \frac{f(n)}{g(n)} = \lim_{N\to\infty} \frac{12n+2\log_2 n}{n^2+6n} = \lim_{N\to\infty} \frac{\cancel{n^2}\left(\frac{12}{n}+\frac{2\log_2 n}{n^2}\right)}{\cancel{n^2}\left(1+\frac{6}{n}\right)} = \lim_{N\to\infty} \frac{\left(\cancel{\frac{12}{\infty}}^{\,0}+\frac{2\log_2 n}{n^2}\right)}{1+\cancel{\frac{6}{\infty}}^{\,0}} =$$

$$(*)\ \lim_{N\to\infty} \frac{0+\frac{\log_2 n^2}{n^2}}{1} = \lim_{N\to\infty} \frac{\log_2 n^2}{n^2}$$

Lets look at the graph



As you can see from the graph when n approaches to ∞ $n^2 (g(n))$ grows faster than $\log_2 n^2 (f(n))$. Therefore, the result of $\lim_{N\to\infty} \frac{\log_2 n^2}{n^2} = 0$. So that $f(n) = O(g(n))$

c-) $f(n) = n \cdot \log_2 3n$ and $g(n) = n + \log_2(8 \cdot n^3)$

Take the limit of $\frac{f(n)}{g(n)}$

$$\lim_{N\to\infty} \frac{n\log_2 3n}{n+\log_2(8n^3)} = \lim_{N\to\infty} \frac{\cancel{n^2}\left(\frac{\log_2 3n}{n}\right)}{\cancel{n^2}\left[\underbrace{\frac{1}{n}}_{0} + \underbrace{\frac{\log_2 8}{n^2}}_{0} + \frac{\log_2 n^3}{n^2}\right]} = \lim_{N\to\infty} \frac{\frac{\log_2 3n}{n}}{\left(\frac{\log_2 n^3}{n^2}\right)} =$$

$$\lim_{N\to\infty} \frac{\frac{\log_2 3n}{\cancel{n}}}{\frac{3\log_2 n}{\cancel{n^2}}} = \lim_{N\to\infty} \frac{n \cdot \log_2 3n}{3\log_2 n} = \infty.$$

We know that $n\log n$ grows faster than $\log n$ when n approaches to ∞. Since $f(n)$ grows faster than $g(n)$ the limit will be ∞. $\underline{\underline{f(n) = \Omega(g(n))}}$

d-) $f(n) = n^n + 5n$ and $g(n) = 3 \cdot 2^n$

Take the limit of $\frac{f(n)}{g(n)}$

$$\lim_{N\to\infty} \frac{n^n + 5n}{3 \cdot 2^n} = \lim_{N\to\infty} \frac{\cancel{n^2}\left[\frac{n^n}{n^2} + \underbrace{\frac{5}{\cancel{n}}}_{0}\right]}{\cancel{n^2}\left[\frac{3 \cdot 2^n}{n^2}\right]} = \lim_{N\to\infty} \frac{\frac{n^n}{\cancel{n^2}}}{\frac{3 \cdot 2^n}{\cancel{n^2}}} = \lim_{N\to\infty} \frac{n^n}{3 \cdot 2^n}$$

We know that n^n grows faster than 2^n. Therefore, $f(n)$ will grow faster than $g(n)$ so that the $\lim_{N\to\infty} \frac{f(n)}{g(n)}$ will be ∞

$f(n) = \Omega(g(n))$. $g(n)$ is a lower bound e-) $f(n) = \sqrt[3]{2n}$ and $g(n) = \sqrt{3n}$

Take the limit of $\frac{f(n)}{g(n)}$

$$\lim_{N\to\infty} \frac{f(n)}{g(n)} = \frac{\sqrt[3]{2n}}{\sqrt{3n}} = \frac{\sqrt[3]{2}\sqrt[3]{n}}{\sqrt{3}\sqrt{n}}$$, let's ignore the constant, the limit becomes

$$\lim_{N\to\infty} \frac{\sqrt[3]{n}}{\sqrt{n}} \quad \lim_{N\to\infty} \frac{n^{1/3}}{n^{1/2}} = \lim_{N\to\infty} \frac{1}{n^{1/6}} = \frac{1}{\infty} = 0$$, $\sqrt{n}$ grows faster than $\sqrt[3]{n}$ so that $\lim_{N\to\infty} \frac{\sqrt[3]{n}}{\sqrt{n}}$ is 0

$f(n) = O(g(n))$, $g(n)$ is upper bound for $f(n)$

2-)

a-) I assumed that the length of the string is n

```
static void methodA (string names[]) {
    for (int i=0 ; i< names.length; i++)  Θ(n)
        System.out.println (names[i]);  Θ(1)
}
```

$T(n) = O(n)$

b-)

```
static void methodB () {
String[] myArray = new String[]{"CSE222", "CSE505, "HW2"};   Θ(1)
for (int i=0; i< myArray.length; i++)  Θ(n)
    methodA (myArray);  Θ(n)

}
```

$T(n) = O(n^2)$

c-)

```
static void methodC int numbers[]) {
int i= 0;
while (i< numbers.length)  Θ(n)
    System.out.println(numbers[i]);  Θ(1)
}
```

(Since i is not increasing a run time error will occur but if i was increasing time complexity would be $O(n)$)

$T(n) = O(n)$

d-) static void methodD (int numbers[]){

```
    int i = 0
    while (numbers[i] < 4)   O(n)
        System.out.println(numbers[i++]);   O(1)
}
```

All the numbers in the int numbers array could be smaller than 4 Since we are trying to find worst case scenario. We will take into account that scenario.

$$T(n) = O(n)$$

3-) I assumed the length of myArray is n

```
static void withoutLoop(int[] myArray){
    int i = 0;
    System.out.println(myArray[i++]);
    System.out.println(myArray[i++]);
    System.out.println(myArray[i++]);
    ⋮
    System.out.println(myArray[i++]);
}
```

Steps/exec	freq	total
1	1	1
1	1	1
1	1	1
1	1	1
⋮	⋮	⋮
1	1	1
		$n+1$

n times

$T(n) = O(n)$

```
static void withLoop(int[] myArray){
    for (int i=0; i< myArray.length; i++)
        System.out.println(myArray[i]);
}
```

Steps/exec	freq	total
1	$n+1$	$n+1$
1	n	n
		$2n+1$

$T(n) = O(n)$

* Both statements have the same time complexity which is $O(n)$. Therefore there is no difference between them in terms of time complexity/efficiency. However, the withloop method's readability is better than withoutLoop's readability. Therefore, even though there is no difference between them in terms of time complexity, It is more advantageous to use withloop in terms of Clean code Principles.

4-)

No, it is not possible to detect whether array contains a specific integer in constant time if we don't have the information about array (sorted or not)

In this question, there is no information is given about array so that we have to calculate worst case scenario. In the worst case scenario we would check each index of array and detect whether array contains that specific integer or not. Check the following to see how to detect it in O(n) time.

```
Public boolean checkInteger(int[] arr, int target)
  for( int i = 0 ; i < arr.length ; i++){   O(n)
        if ( target == arr[i])   O(1)
              return true;   O(1)
  }
}
```

$T(n) = O(n)$

This is the way to check it in O(n) time, if we don't have any information about array. However, it is not possible to solve it in constant time O(1).

5-) $A = [a_0, a_1, \dots\dots, a_{n-1}]$
$B = [b_0, b_1, \dots\dots, b_{m-1}]$ $n, m \in \mathbb{Z}^+$

```
int findMinimumMultiplication( int[] A, int[] B) {

    int n = A.length;  // n is the length of array A        O(1)
    int m = B.length;  // m is the length of Array B        O(1)
    int minA = A[0];                  O(1)
    int minB = B[0];                  O(1)
    int maxPositiveA = A[0];          O(1)
    int maxPositiveB = B[0];          O(1)

    for(int i=0; i<n; i++){          O(n)
      if ( A[i] <= minA )         O(1)
          minA = A[i];            O(1)
      if (A[i] >= maxPositiveA)   O(1)
          maxPositiveA = A[i];    O(1)
    }

    for(int i=0; i<m; i++){          O(m)
      if( B[i] <= minB)               O(1)
          minB = B[i];                O(1)
      if (B[i] >= maxPositiveB)       O(1)
          maxPositiveB = B[i];        O(1)
    }

    int minPositive = minA * minB;
    int minNegative1 = minA * maxPositiveB;
    int minNegative2 = maxPositiveA * minB;
    if ( (minPositive <= minNegative1) && (minPositive <= minNegative2))
        return minPositive;
    else if( (minNegative1 <= minPositive) && ( minNegative1 <= minNegative2))
        return minNegative1;
    else if(( minNegative2 <= minPositive) && (minNegative2 <= minNegative1))
        return minNegative2;
    return 0;
}
```

$T(n) = O(m+n)$ Linear time