CS 2210 : Data Structures & Algorithms
Assignment - 1

Q1. $\frac{2}{n}$ is $O(n)$: To prove

We must find a constant $c > 0$ and an integer $n_0 \geq 1$ such that $\frac{2}{n} \leq cn$ for all $n \geq n_0$

Simplifying the inequality :
move $\frac{1}{n}$ to the right side of inequality,

$$2 \leq cn^2 \text{ for all } n \geq n_0$$

let $c = 1$

$$n^2 \geq 2$$

$$n \cdot n \geq 2$$

(inequality is valid for all values from 2 onwards)
Therefore, $n_0 = 2$

$\Rightarrow$ constants are : $c = 1$, $n_0 = 2$
which satisfies the inequality and so
$\frac{2}{n}$ is $O(n)$ is proven.

(Note, these are not the only values that we could ~~select~~ select to prove that $\frac{2}{n}$ is $O(n)$)

Q2 $f(n) > 0$ and $g(n) > 0 \longrightarrow f(n)$ is $O(g(n))$

To prove $f(n) \times g(n)$ is $O(g^2(n))$, $\{g^2(n) = g(n) \times g(n)\}$

we must find constants $c > 0$ and integer $n_0 \geq 1$
such that $f(n) \times g(n) \leq c\,(g(n) \times g(n))$ for all $n \geq n_0$

~~Dividing both sides of inequality by $g(n)$:~~
~~$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$~~
~~(since $g(n) > 0$, it is allowed)~~

~~Again dividing~~
given that $f(n)$ is $O(g(n))$
$\Rightarrow$ there exists a positive constant c and
an integer $n_0 \geq 1$
such that $f(n) \leq c(g(n))$ for all $n \geq n_0$ —(1)

multiplying $g(n)$ on both sides since $g(n) > 0$
$f(n) \cdot g(n) \leq c\,(g(n) \cdot g(n))$ {required}
for all $n \geq n_0$

Q3 To prove : $n^3 + \frac{n^4}{4}$ is not $O(n^3)$

We will use a proof by contradiction : Assuming that the claim is false i.e., $n^3 + \frac{n^4}{4}$ is $O(n^3)$ and derive a contradiction from this assumption.

If $n^3 + \frac{n^4}{4}$ is $O(n^3)$ then by the definition of big Oh, there exists constants $c > 0$ & $n_0 \geq 1$ such that $n^3 + \frac{n^4}{4} \leq cn^3$ for all $n \geq n_0$

Multiply both sides of the inequality by 4 :

$$4n^3 + n^4 \leq 4cn^3 \quad \text{for all } n \geq n_0$$

Divide both sides of the inequality by n^3 :

$$4 + n \leq 4c \quad \text{for all } n \geq n_0$$

Subbracting 4 on both sides of the inequality :

$$n \leq 4(c-1) \quad \text{for all } n \geq n_0$$

let $c = 1$, $n \leq 0$

The inequality states that $n \leq 0$ which isn't possible since $n_0 \geq 1$ and $n \geq n_0$.

Therefore, we have reached a contradiction as there aren't constant values $c > 0$ and $n_0 \geq 1$ such that $n^3 + \frac{n^4}{4} \leq cn^3$ for all $n \geq n_0$

Consequently, $n^3 + \frac{n^4}{4}$ is not $O(n^3)$.

Q4 The given algorithm is correct as it produces the correct output for any array L storing n integers values and any positive integer value x. It will always terminate and give the position of x in L or give -1 if x is not in L.
The algorithm works in the following way: a function search receives an array L, with n as the size and x as a positive integer to be found in the array. The array L has greater than or equal to 1 number of values. i is initialized to 0 and then the while loop checks if i is less than n and also simultaneously checks if L[i] is equal to -1 (meaning that if any element in the array is a negative number then it cannot be x which is a positive number). If any of these 2 check conditions is true, it increaments the value of i by 1 and goes to the if condition. If any check condition is false, it skips the loop and i increament and goes to the if condition which checks that if i is equal to n ~~(meaning at the last position plus 1)~~ then ~~we have~~ searched the array ~~containing only~~ contains no value since i was 0 so n would also be 0. In this value isn't found. The ~~other~~ scenario maybe if after the while loop i has some positive value which

matches x and so returns -1 and terminates program since the array has only negative value. In case the first if is false, the second else if check if the corresponding element of the array matches x. If its true, it returns the position of x or else goes to the last else condition. The last else gives a recursive return where ~~if~~ it calls search (L, n, x) after resetting the ~~sep~~ specific element to -1 which wasn't equal to x so that when recursion takes place, the value of i is increased by 1 in the while loop.

Example, $L=[1,2,3,4]$, $n=4$, $x=4$.

The algorithm goes like this:

The while isn't entered as $L[i] \neq -1$ since its 1 $(L[0]=1)$. The if and else if are not false, so it reaches else and initilizes $L[0]=-1$ and calls the function again. Now $L=[-1,2,3,4]$, $n=4$, $x=4$. The while now executes and i becomes 1, so $L[1]=2$. The function runs the same way till $i=3$ and at $L[3]=4$, the else if condition becomes true as $L[3]=x$ is true. It return $i=3$ and the program terminates.

Hence, the algorithm executes correctly.

Q5 The Activation Records :

c = 39 x = 0	target = 40	ret addr = A3	⟵ top
c = 30 x = 1	target = 39 m = 40	ret addr = A2	
c = 20 x = 1	target = 40 m = 30	ret addr = A1	
pos = 20	res =	ret addr = OS	⟸ ~~top~~

Execution Stack

The execution stack looks like the above stack before return x (***) is executed

Q6

n	Linear Search
5	211 ns
10	401 ns
100	1163 ns
1000	4693 ns
10000	9442 ns
100000	28802 ns

n	Quadratic Search
5	510 ns
10	1175 ns
100	9774 ns
1000	201348 ns
10000	9344463 ns

n	Factorial Search
7	1767090 ns
8	11557417 ns
9	53603675 ns
10	549293458 ns
11	6092699041 ns
12	74644015083 ns

Note : ns represents nano - seconds