Juan Espinal
CS2810

Assignment #1

1) prove that $\frac{2}{n}$ is $O(n)$

we must find constant $c > 0$ and integer constant $n_0 \geq 1$ such that:

$$\frac{2}{n} \leq cn \quad \forall\ n \geq n_0$$

$\frac{2}{n} \leq cn$	$\forall\ n \geq n_0$	Multiply both sides by n
$2 \leq cn^2$	$\forall\ n \geq n_0$	
Choose $c=1$ $\quad 2 \leq n^2$	$\forall\ n \geq n_0$	
$n^2 \geq 2$	$\forall\ n \geq n_0$	
Choose $n_0 = 2$ $\quad n^2 \geq 2$	$\forall\ n \geq 2$	

$\therefore$ The inequality holds true for $c=1$ and $n_0 = 2$, thus proving $\frac{2}{n}$ is $O(n)$

3) Let $f(n)$ and $g(n)$ be positive functions such that $f(n)$ is $O(g(n))$

prove $f(n) \times g(n)$ is $O(g^2(n))$, where $g^2(n) = g(n) \times g(n)$

We must find constants $C > 0$ and integer constant $n_0 \geq 1$ such that:

$$f(n) \times g(n) \leq C\, g^2(n) \quad \forall\, n \geq n_0$$

Because $g^2(n) = g(n) \times g(n)$, we can write:

$$f(n) \times g(n) \leq C\, g(n) \times g(n) \quad \forall\, n \geq n_0$$

We can divide both sides of the inequality by $g(n)$ since $g(n)$ is a positive function.

$$f(n) \leq C\, g(n) \quad \forall n \geq n_0$$

We know $f(n)$ is $O(g(n))$. By definition, there exists constants $C > 0$ and integer constants $n_0 \geq 1$ such that the inequality is satisfied $\forall n \geq n_0$.

∴ Thus, because such constants exist,

$f(n) \times g(n)$ is $O(g^2(n))$

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

Proof by contradiction

Assume $n^3 + \frac{n^4}{4}$ is $O(n^3)$, then there exists constant $c > 0$ and integer constant $n_0 \geq 1$ such that:

$$n^3 + \frac{n^4}{4} \leq cn^3 \quad \forall\, n \geq n_0$$

Substract n^3 from both sides: $\frac{n^4}{4} \leq cn^3 - n^3 \quad \forall\, n \geq n_0$

Factor out n^3: $\frac{n^4}{4} \leq n^3[c-1] \quad \forall\, n \geq n_0$

Divide both sides by n^3: $\frac{n}{4} \leq [c-1] \quad \forall\, n \geq n_0$

Multiply by 4 both sides: $n \leq 4[c-1] \quad \forall\, n \geq n_0$

The inequality $n \leq 4[c-1]$ is valid only for values that are at most (but not more than) $4(c-1)$. Meaning, we can never obtain a value for n_0 such that $\forall\, n \geq n_0$, the inequality would hold true.

Specifically, if we choose n_0 to be $4[c-1]$, $4[c-1]+1$ makes the inequality false. Meaning that Not $\forall\, n > n_0$ would satisfy the inequality

Therefore, we have reached a contradiction as there are no constant values $c > 0$ and $n_0 \geq 1$ such that $n \leq 4[c-1]$ for all $n \geq n_0$. Consequently, $n^3 + \frac{n^4}{4}$ is not $O(n^3)$

4) The algorithm is correct:

Termination →

Case 1: x is in L

- i takes on values of 0, 1, 2, 3, ... until L[i] ≠ -1
- if L[i] = x, then the algorithm returns the value of i which is the position of x in L
- Otherwise, -1 is assigned to L[i] and the algorithm recursively calls itself again
- Because there is a return statement for search (L, n, x), the value of i is guaranteed to be returned once it finds x in L
- Therefore, search (L, n, x) does not recursively call itself infinitely

Case 2: x is not in L

- If x is not in L, all values in the array will be replaced by -1
- Because of this, the conditions (i < n) & (L[i] = -1) will always be true until i = n in some iteration of the while loop
- Once i reaches n, the while loop terminates and the condition i = n becomes true which then returns -1

- Because there is a return statement for search (L, n, x), -1 is guaranteed to be returned once i = n
- Thus, search (L, n, x) does not recursively call itself indefinitely

∴ In either case, the program always terminates

Correctness:

Case 1: x is in L

- The algorithm compares L[0], L[1], L[2]..... with -1
- if L[i] ≠ -1, the while loop terminates and it then checks if x is in L[i]
- if L[i] = x, then the algorithm returns the value of i which is the position of x in L
- if L[i] ≠ x, the value of L[i] is replaced with -1 and the algorithm calls itself again
- Because x is in L, the condition L[i] = x is guaranteed to be true in some recursive call of search (L, n, x). Thus, the algorithm correctly returns the position of x in L, if x is in L.

Case 2: x is not in L

- If x is not in L, all values in the array will be replaced by -1
- Because of this, the conditions (i < n) & (L[i] = -1) will always be true until i = n in some iteration of the while loop

- once i reaches n, the while loop terminates and the condition i=n becomes true which then returns -1
- thus, the algorithm correctly returns -1 if x is not in L

∴ In either case, the program returns the correct output

Therefore, Because the algorithm:

1) Terminates, and
2) produces the correct output

This proves that the algorithm is correct.

5)

			ret value = ← Top
c = 39	x = 0	target = 40	ret addr = A3
m = 40			ret value =
c = 30	x = 1	target = 39	ret addr = A2
m = 30			ret value =
c = 20	x = 1	target = 40	ret addr = A1
pos = 20	res =		ret addr = OS