CS2210A – Assignment 1
27 September 2018
Syed Ahmed
250897473

Question 1

Using the definition of “big Oh”, we need to find constants c > 0 and n_0 >= 1 integer such that

$4/n$ *is* $<= c, \forall n >= n_0$ (1)

First, we simplify the inequality by multiplying both sides by n to get

$4 <= cn, \forall n >= n_0$

Since the right hand side of the inequality needs to be positive, we choose, for example, c = ¼:

$4 <= \frac{1}{4} n, \forall n >= n_0$

Note that n >= 4 for all values n >= 4, so we choose n_0 = 4.

As we have found constant values c = ¼ and n_0 = 4 that make inequality (1) true, then we have proven that $4/n$ *is* $O(1)$

Question 2

I am claiming that $2n$ *is* $O(n)$
Using the definition of “big Oh”, we need to find constants c > 0 and n_0 >= 1 integer such that

$2n <= cn, \forall n >= n_0$

Since $n >= n_0$ and n_0 >= 1, then n is positive. Hence we can divide both sides by n to get

$2 <= c, \forall n >= n_0$

Note that regardless of the value of c, n cannot be less than or equal to c for all $n >= n_0$ because n is a function that grows without bounds. Hence, for example if $n = max\{c,n_0\} +1$, this value is larger than or equal to n_0, but is also larger than c, hence the inequality is not true and we have derived a contradiction

Question 3
$f(n)$ *is* $O(g(n))$
Using the definition of “big Oh”, we need to find constants c > 0, n_0 >= 1 integer such that

$f(n) <= c(h(n)), \forall n >= n_0$

$g(n)$ *is* $O(h(n))$
Using the definition of “big Oh”. We need to find constants c > 0, n_0 >= 1 integer such that

$g(n) <= c(h(n))$

Using the max theorem,

f(n) + g(n) is O(max {f(n), g(n)}),

f(n) <= c(g(n)) <= c(h(n))

Therefore, *f(n) + g(n) is O(h(n)*

Question 4

Pseudocode for algorithm as described:

set integer **size** equal to size of array
set boolean **repeat** equal to false
set integer **currentPosition** equal to zero
create empty array **check**
while **currentPosition** is less than or equal to **size** and **repeat** is equal to false
if **currentPosition** is in **check** then set **repeat** equal to true
else then add **currentPosition** to **check**
return **repeat**

This algorithm terminates after a finite time as the array has a finite number of integers in it, so when the while loop is entered, it will exit as soon as one repeat is found, or in the worst case scenario, it will exit when it has iterated through every single integer in the array. The currentPosition variable is initialized at zero and bounded by the size of the array, if it was not upper bounded then there would be infinite number of iterations, but this is not the case.

This algorithm is correct as it outputs the correct answer by using a boolean check. The key component to evaluate to check correctness is the return statement which has one of two options – true or false. As soon as one repetition is found, the repeat boolean is turned to true, and the while loop is exited and the value of true is returned. If the entire while loop is iterated through and no repeats are found, then the repeat boolean will return false, the value it was initialized with.

The time complexity of the algorithm in the worst case is *O*(n) as it will be iterated through the array n number of times, which will be the size of the entire array. This is the only time-dependent factor in the algorithm so it is only operation needed to take into consideration for worst case.

Question 5

n value	Factorial Search	Quadratic Search	Linear Search
5	226991 ns	1415 ns	981 ns
8	32834425 ns	-	-
9	123273862 ns	-	-
10	783183278 ns	872 ns	415 ns
11	26570461614 ns	-	-
12	236361190303 ns	-	-
100	-	23306 ns	1180 ns
1000	-	303577 ns	9412 ns
2000	-	2226910 ns	27739 ns
10000	-	-	26826 ns