**Firas Aboushamalah**

**250-920-750**

**CS2210A**

**Assignment 1**

Part 1:

1. Prove that 4/n is O(1)

First we need to find constants c > 0 and n0 > 1such that

f(n) ≤ cg(n), ∀n ≥ n0

Then f(n) = 4/n is O(1) when:

4/n ≤ c, ∀n ≥ n0

Now we choose c = 1 to further the equation:

4/n ≤ 1, ∀n ≥ n0

Simplify the equation by moving n to the other side

4 ≤ n, ∀n ≥ n0

Now we may choose n0 = 4 which, with constant c = 1 makes the inequality true. As a result we have proven that f(n) = 4/n is O(1).

1. Prove that 2n is **NOT** O(1/n) (**by contradiction**)

First we must falsely assume that 2n is O(1/n) such that there exists constants c > 0 and n0 ≥ 1 ∀n ≥ n0:

f(n) ≤ cg(n), ∀n ≥ n0

Then we simplify the equation by plugging in the correct values

2n ≤ c/n, ∀n ≥ n0

Now we multiply both sides by n to get

2n2 ≤ c, ∀n ≥ n0

However, this cannot be true because 2n2 will constantly grow without limit regardless of c and n0**,** meaning it is not O(1/n).

1. Prove that f(n) + g(n) is O(h(n)).

We know that if f(n) is O(h(n)) then there exist constants for f(n) such that C1 > 0 and n10 ≥ 1 such that

f(n) ≤ c1h(n), ∀n ≥ n10

This is also true that if g(n) is O(h(n)), then there exist constants

c2 > 0 and n20 ≥ 1 such that

g(n) ≤ c2h(n), ∀n ≥ n20

This, we may add these two functions together and combine their constants such that

f(n) + g(n) ≤ c1h(n) + c2h(n), ∀n ≥ max(n10, n20)

However, we can simplify this in order to extrapolate the constants:

f(n) + g(n) ≤ c1 + c2(h(n)) ∀n ≥ max(n10, n20)

Thus, we can determine c1 + c2 and max(n10, n20) as our two constants.

**Algorithm:** ArrayFind (*A, k, n*)

**Input**: Array *A* of size *k* and input value *n*.

**Output**: Return false if there is a duplicate in Array *A* or true if every value in *A* is different.

Pseudocode version

{

**for** *i* ← 0 **to** *k* ***do*** *{*

**for** *j* ← *i* + 1 **to** *k* **do** {

**if** *j*!=*i* **and** *A*[*j*] == *A*[*i*] **then** **return** false

}

**return** true

}

Code version

{

for (i=0;i<k;i++) {

for (j=i+1;j<k;j++) {

if (j!=i && A[j] == A[i])

return false

}

return true

}

**4a**. To show that an algorithm is correct we must show that it terminates and produces the proper output. In this case, the algorithm begins with i and j performing iterations incrementing by 1 and ending at *k* iterations because of (i=0;i<k;i++) and the second loop (j=i+1;j<k;j++). Therefore, both counters end once they have reached k to determine whether the array is true or false.

**4b.** The algorithm compares the variable *j* position in the array with the position *i as j=i*+1. Therefore, when *i*=0, j = 1 and compares values at A[1], A[2], A[3], A[4]…A[k] with the value of A[i] which is in place 0. If there are no matches, then the *i* counter is increased by 1 and the j traverses through the loop again by comparing all the values with A[1]. Hence, if *i*==*j* and also not the same in the same location as j (j!=i) then a match is made and **return** false, thereby exiting the loop completely. If there are no duplicates then the loop continues until k iterations are reached and the program **returns** true.

**The worst case** occurs when A[j] is not equal (!=) to A[i] and the maximum number of iterations are reached (k iterations). Therefore, the most amount of time is exhausted because the if statement constantly does not return false and is bypassed, until eventually the program returns true meaning it traversed over the entire array.

**Time complexity of ArrayFind in Worst Case**

In the above algorithm, there are two nested for loops. In the outer loop, there are **3** primitive operations (=, <, ++) multiplied by **n** iterations. In the inner loop, there are **7** primitive operations (=, +, <, ++, !=, &&, ==) followed by a return statement. All of this is multiplied by **n** iterations. Outside of the nested loops, there is one return statement. This gives us **24n2****+ 2**, meaning that the algorithm is in fact, **O(n2).**

1. Question 5

|  |  |  |
| --- | --- | --- |
| Factorial Search | **Input Size n = 5** | **Output:** 166120 ns |
| **Input Size n = 8** | **Output:** 14213774 ns |
| **Input Size n = 9** | **Output:** 67354678 ns |
| **Input Size n = 10** | **Output:** 644636257 ns |
| **Input Size n = 11** | **Output:** 7385665747 ns |
| **Input Size n = 12** | **Output:** 90078588553 ns |

|  |  |  |
| --- | --- | --- |
| Quadratic Search | **Input Size n = 5** | **Output:** 360 ns |
| **Input Size n = 10** | **Output:** 714 ns |
| **Input Size n = 100** | **Output:** 6883 ns |
| **Input Size n = 1000** | **Output:** 127959 ns |
| **Input Size n = 2000** | **Output:** 465464 ns |
|

|  |  |  |
| --- | --- | --- |
| Linear Search | **Input Size n = 5** | **Output:** 179 ns |
| **Input Size n = 10** | **Output:** 375 ns |
| **Input Size n = 100** | **Output:** 983 ns |
| **Input Size n = 1000** | **Output:** 5157 ns |
| **Input Size n = 2000** | **Output:** 8064 ns |
| **Input Size n = 10000** | **Output:** 11760 ns |