### Connor Jensen

### Written Assignment 01

### February 24, 2024

### P1. (10 pts) Proof by Induction

### Let F1 be the Fibonacci numbers as defined in Section 1.2

### Assume: F0=1, F1=1, and Fn = Fn-1 + Fn-2

### Prove:

### Base Case is True because both the left and right side are equal to 1 when N = 3

### Assumptions for Inductive Case (N = N + 1):

### N = (D + 2) => (N + 1) = D + 3

### Inductive Case is True because both sides are equal to when

* For N>= 1,

### Prove:

### Base Case is True because both the left and right side are equal to 1 when N = 1

### Assumptions for Inductive Case:

### Inductive Case is True because both sides are equal to (N + 1)2 when

P2. (5 pts) (5 pts) An algorithm takes 1 ms for input size 100. How long will it take for input size 800 if the running time is the following (assume low-order terms are negligible):

* + Linear

**A: 8 ms**

* + O (N log2 N)

**A: 24 ms**

* + Quadratic

**A: 64 ms**

* + Cubic

**A: 512 ms**

P3. (10 pts) For each of the following program fragments, give an analysis of the running time (using Big-Theta)

i)

def fun1(n):

i = 1 # C

sumA = 0 # C

while i < n \* n: # While loop = O(N^2)

sumA += 1

i += 3

A: Using Big-O analysis, the while loop uses (N2 / 3) processes to complete; so, the Big-Theta notation is equal to **O(N2)**

ii)

def fun2(n):

i = 1 # C

sumB = 0 # C

while i < n \* n: # While loop = O(log (N))

sumB += 1 # C

i \*= 3 # C

A: Using Big-O analysis, the while loop uses (N2 / i2) processes to complete; so, the Big-Theta notation is equal to **O(log N)**

iii)

def fun3(n):

sumC = 0 # C

for i in range(n): # N

for j in range(i\*2, n\*\*3): # For loop = O(n\*\*3/2i)

for k in range(j): # For loop = O(n\*\*3/2i)

sumC += 1 # C

A: Using Big-O analysis, the first for-loop uses N process to complete, while the subsequenct two for-loops each use (N3/2i) processes to complete; so, the Big-Theta notation is equal to **O(N7)**

iv)

def fun4(n):

sumD = 0 # C

for i in range(n): # N

for j in range(i\*2, n\*\*3): # N\*\*3/2i

if j < i: # Never happens: N

for k in range(j): # Never happens: N\*\*3

sumD += 1 # C

A: Using Big-O analysis, the first for loop uses (N) processes to complete, the second for loop uses (N\*\*3) processes, and the if statement never is true, so the nested loop inside will never be reached; so, the Big-Theta notation is equal to **O(N4)**

v)

def fun5():

k = 0 # C

n = 5 # C

if n > 10: # C

k = n

else: # C

for i in range(n):

for j in range(n):

k += 1

A: Using Big-O analysis since all variables in loops are stationary and thus cannot increase nor decrease, the amount of processes is constant, meaning the big-Theta notation is **O(1)**

P4. (10 pts) What is the asymptotic complexity of the following functions? Justify your answer.

1. (3 pts)

def fun1(n):

i = 0

if (n > 1):

fun1(n - 1)

for i in range(n):

print(" \* ",end="")

Recurrence Relation:

Complexity in Big-Oh: Show your process of finding the complexity from the recurrence relation.

**Using Master’s Theorem, 1 = 1, therefore, the complexity in Big-Oh is N\*log N**

1. (3 pts)

def fun(a, b):

if (b == 0):

return 1

if (b % 2 == 0):

return fun(a\*a, b//2)

return fun(a\*a, b//2)\*a

Recurrence Relation:

**T(N) = 2T(N/2) + O(N2) + C**

Complexity in Big-Oh: Show your process of finding the complexity from the recurrence relation.

**Using Masters Theorem, 2 < 22, therefore, the complexity in Big-Oh is N2**

### (4 pts)

def func(n,start,end,aux):

if(n==1):

print("Move disk 1 from",start,"to",end)

return

func(n-1,start,aux,end)

print("Move disk",n,"from",start,"to",end)

func(n-1,aux,end,start)

Recurrence Relation:

**T(N) = 2T(N – 1) + O(2n) + C**

Complexity in Big-Oh: Show your process of finding the complexity from the recurrence relation.

**2n is the biggest term in the Recurrence Relation, and thus will be the Big-Theta: 2n**