Homework 2

Yuqiao Meng

1. Use the definition of θ and Omega to prove the sequence: $\Omega(g(n))$ = {f(n): there exsits a positive constants c, and n0 such that 0 <= f(n) <= cg(n) for all n >= n0}. So just find the three constants c, n0 to prove the sequence. And I think there are rules no need to prove: 1 < ln n < lg n< n < n^a(a>1) < b^n(b>1)
    a. 1
    b. n^(1/lgn) = 2 because 2^(lgn) = n. n^(1/lgn) = Θ(1)
    c. lg(lg*(n)): when n = 65536, this function equals to 2, so it grows faster than n^(1/lgn)
        i. n0 = 65536, lg(lg*(n0)) = 2, n0^(1/lgn0) = 1
        ii. c = 1, cg(n0) = 2 > 1
    d. lg*(lg(n)): let n = 2^m, lg*n = lg*2^m = lg*m + 1, lg*n = m, thus lg(lg*n) = lg(lg*m+1), lg*(lg(n)) = lgm, obviously the latter one grows exponentially faster.
        i. n0 = 65536, lg*(lgn) = 3, lg(lg*n) = lg3 = 2
        ii. c2 = 1, cg(n0) = 3 > 2
    e. lg*n: According to the definition, lg*(lg(n)) = lg*n - 1, so lg*n = Θ(lg*(lgn))
        i. n0 = 2, lg*n0 = 1, lg*lgn0 = 1
        ii. c1 = 1/2, c1g(n0) = 1/2 < 1
        iii. c2 = 3, c2g(n0) = 3 > 1
    f. 2^(lg*n): power grows faster than polynomial
        i. n0 = 4, 2^(lg*n) = 4, lg*n = 2
        ii. c = 1, cg(n0) = 4 > 2
    g. ln ln n: obviously lnlnn can't grow slower than lg*n for it's a recursive function
        i. n0=65536, lnlnn = 2.4060758017, 2^(lg*n) = 16.
        ii. c = 8, cg(n0) = 19.248 > 16
    h. lgn^(½): let n = 2^(m^2), lgn^(½) = m, ln ln n = 2*ln m = 2*(lgm/lge), thus the latter grows slower than former one
        i. n0 = 65536, ln ln n = 2.406, lgn^(1/2) = 4
        ii. c = 1, cg(n0) = 4 > 2.406
    i. ln n: lgn^(1/2) = (lnn/ln2)^(½) = 1.2*(ln n)^(½), thus ln n grows faster than lgn^(1/2)
        i. n0 = 8, ln n0 = 2.07, lgn^(1/2) = 2
        ii. c = 1, cg(n0) = 2.07 > 2
    j. lg^2(n): ln n = lgn/lge = 0.7lgn, thus lg^2(n) grows faster
        i. n0 = 4, lg^2(n0) = 4, ln n0 = 1.386
        ii. c = 1, cg(n0) = 4 > 1.386
    k. 2^((2lgn)^(1/2)): let n =lg((2^m/2)*2), thus 2^((2lgn)^(1/2)) = m, lg^2(n) = lg^2(lg((2^m/2)*2)), thus this equation grows faster.
        i. n0 = 2, 2^((2lgn)^(1/2)) = 4, lg^2(n) = 4
        ii. c = 2, cg(n0) = 8 > 4
    l. 2^(½*lgn): 1/2lgn grows faster than 2lgn^(1/2)
        i. n0 = 4, 2^(½*lgn) = 2, 2^((2lgn)^(1/2)) = 4
        ii. c = 4, cg(n0) = 8 > 4
    m. 2^(lgn): lgn grows faster than ½*lgn

        i. $n0 = 2, 2^(lgn) = 2, 2*(½*lgn) = 1$
        ii. $c = 1, cg(n0) = 2 > 1$
    n. n: $2^lgn = n$, thus $2^lgn = Θ(n)$
    o. lg(n!): n = lg(2^n), 2^n grows slower than n! when n0 = 4, c = 1, thus lg(n!) grows faster.
        i. n0 = 4, lgn! = 4.58, n = 4
        ii. c = 1, cg(n) = 4.58 > 1
    p. nlg(n): lg(n!) = Θ(nlgn) according to the textbook
    q. n^2: lgn = O(n)
    r. 4^(lgn): 4^(lgn) = 2^(2lgn) = 2^(lg(n^2)) = n^2, 4^(lgn) = Θ(n^2)
    s. n^3
    t. (lgn)!: let m = lgn, thus n^3 = (2^m)^3 = 8^m, (lgn)! = m!, and m! grows faster than 8^m
    u. lgn^(lgn): n^n grows faster than n!
    v. n^(lglgn): lgn^(lgn) = n^lglgn because a^(log_b^c) = c^(logb_a)
    w. (3/2)^n: let m = lgn, lgn^lgn = m^m, (3/2)^n = (3/2)^(2^m), latter grows faster than former
    x. 2^n: 2 > 3/2
    y. n(2^n): n*2^n > 2^n
    z. e^n = 2^n(e/2)^n = Omega(n*2^n) for (e/2)^n = Omega(n)
    aa. n!: (n-1)! = 1*2*3*...(n-1), 2^n = 2*2*...2, for n > 3, 3*4*...(n-1) is larger than 2*2*...2, so n! grows faster than n(2^n)
        i. n0 = 6, n! = 720, n(2^n) = 384
        ii. c = 1, 720 > 384
    bb. (n+1)! = n! * (n+1)
    cc. 2^2^n: (n+1) = omega(2^n) =omega(2^2^n)
    dd. 2^2^(n+1) = (2^2^n)^2 = Omega(2^2^n)
2. Use master theorem to find upper bound and lower bound of each recurrence
    a. T(n) = 2T(n/2) + n^3:
        i. a = 2, b = 2, log_b^a = lg2 = 1, f(n) = n^3
        ii. let ε = 1, f(n) = n^3 = Ω(n^(1+ε)) = Ω(n^2)
        iii. let n0 = 2, c = 0.5, 2(n/2)^3 = ¼(n^3) <= ½(n^3)
        iv. T(n) = Θ(n^3)
    b. T(n) = T(9n/10) + n:
        i. a = 1, b = 10/9, log_b^a = lg_(10/9)^1 = 0, f(n) = n
        ii. let ε = 0.5, f(n) = n = Ω(n^(0+ε)) = Ω(n^0.5)
        iii. let n0 = 10/11, c = 0.5,(9n/10) <= 10n/11
        iv. T(n) = Θ(n)
    c. T(n) = 16T(n/4) + n^2:
        i. a = 16, b = 4, log_a^b = lg_4^16 = 2, f(n) = n^2
        ii. f(n) = n^2 = Θ(n^(lg_4^16)) = Θ(n^2)
        iii. T(n) = Θ((n^2)lgn)
    d. T(n) = 7T(n/3) + n^2:
        i. a = 7, b = 3, log_a^b = lg_3^7 < 2, f(n) = n^2
        ii. let ε = 1.9 - lg_3^7, f(n) = n^2 = Ω(n^(log_3^7+ε)) = Ω(n^1.9)
        iii. let n0 = 2, c = 8/9, 7(n/3)^2 = (7/9)(n^2) <= (8/9)(n^2)
        iv. T(n) = Θ(n^2)
    e. T(n) = 7T(n/2) + n^2:

        i. a = 7, b = 2, log_a^b = lg_2^7 > 2, f(n) = n^2
        ii. let ε = 2.1 + lg_2^7, f(n) = n^2 = O(n^(log_2^7-ε)) = O(n^2.1)
        iii. T(n) = (n^(log_2^7))
    f. T(n) = 2T(n/4) + n^(1/2):
        i. a = 2, b = 4, log_a^b = lg_4^2 = 1/2, f(n) = n^(½)
        ii. f(n) = Θ(n^(½))
        iii. T(n) = Θ(n^(½)lgn)
    g. T(n) = T(n-1) + n:
        i. use recurrence tree to solve this problem
        ii. T(n) = n + (n-1) + … + 1 = n(1+n)/2 = (½)n^2 +(½)n = Θ(n^2)
    h. T(n) = T(n^(1/2)) + 1:
        i. the power of 0 and 1 is not a increasing function, so assume a is the final termination constant = 2, thus n^(1/2)^a = 2, n = 2^2^a, a = lglgn
        ii. T(n) = Θ(lglgn)
3. Analyze: the first statement is the cost of the row, the second statement is the number of times it executes.
    a. for i = 1 to n: c1 (n+1)
       k[i] = 0 c2 (n+1)
       for i = 1 to n: c3 (n+1)
       for j = i to n: c4 sum_1^n(i) = (n+1)n/2
       k[i] = k[i] + j; c5 sum_1^n(i) = (n+1)n/2
       runtime: T(n) = (n+1)c1 + (n+1)c2 + (n+1)c3 + (n+1)n/2*c4 + (n+1)n/2*c5 = Θ(n^2)
    b. i = 1 c1 (1)
       while i < n c2 (lgn)
       i = 2*i c3 (lgn)
       runtime: T(n) = c1 + c2lgn + c3lgn = Θ(lgn)
4. Explanation:
    a. True: according to the definition of Θ, let c1 = 1, c2 = 1000000, for n0 > 100, 0 < c1g(n) < f(n) < c2g(n) is always true
    b. True: according to the definition of Ω, let c = 1, for n0 > 1, 0 < cg(n) < f(n)
    c. True: log(n^100) = 100logn = O(logn)
    d. True: 2^(n+1) = 2*2^n, according to the definition of Θ, let c1 = 1, c2 = 4, for n0 >= 1, 0 < c1g(n) < f(n) < c2g(n) is always true
    e. False: according to the definition of O, let c = 1, for n0 > 1, 0 < n^2 < n^3 is always true, so n^2 = o(n^3), n^3 doesn't equals to O(n^2)
5. Initial array: [1,3,9,2,8,0,1,5,7,6]
    a. j = 2, key = A[2] = 3, i = j - 1 = 1
        i. compare key(3) with A[i], A[i] < key, break: [1,3,9,2,8,0,1,5,7,6]
    b. j = 3, key = A[3] = 9, i = j - 1 = 2
        i. compare key(9) with A[i], A[i] < key, break: [1,3,9,2,8,0,1,5,7,6]
    c. j = 4, key = A[4] = 2, i = j - 1 = 3
        i. i = 3, compare key(3) with A[i], A[i] > key, A[i+1] = A[i] = 9, i = i - 1 = 2: [1,3,9,9,8,0,1,5,7,6]
        ii. i = 2, compare key(3) with A[i], A[i] > key, A[i+1] = A[i] = 3, i = i - 1 = 1: [1,3,3,9,8,0,1,5,7,6]
        iii. i = 1, compare key(3) with A[i], A[i] < key, break, A[i+1] = key = 2: [1,2,3,9,8,0,1,5,7,6]

d. j = 5, key = A[5] = 8, i = j - 1 = 4
   i. i = 4, compare key(8) with A[i], A[i] > key, A[i+1] = key = 9, i = i - 1 = 3: [1,2,3,9,9,0,1,5,7,6]
   ii. i = 3, compare key(8) with A[i], A[i] < key, break, A[i+1] = key = 8: [1,2,3,8,9,0,1,5,7,6]
e. j = 6, key = A[6] = 0, i = j - 1 = 5
   i. i = 5, compare key(8) with A[i], A[i] > key, A[i+1] = key = 9, i = i - 1 = 4: [1,2,3,8,9,9,1,5,7,6]
   ii. i = 4, compare key(8) with A[i], A[i] > key, A[i+1] = key = 8, i = i - 1 = 3: [1,2,3,8,8,9,1,5,7,6]
   iii. i = 3, compare key(8) with A[i], A[i] > key, A[i+1] = key = 3, i = i - 1 = 2: [1,2,3,3,8,9,1,5,7,6]
   iv. i = 2, compare key(8) with A[i], A[i] > key, A[i+1] = key = 2, i = i - 1 = 1: [1,2,2,3,8,9,1,5,7,6]
   v. i = 1, compare key(8) with A[i], A[i] > key, A[i+1] = key = 1, i = i - 1 = 0: [1,1,2,3,8,9,1,5,7,6]
   vi. i = 0, break, A[i+1] = key = 0: [0,1,2,3,8,9,1,5,7,6]
f. j = 7, key = A[2] = 1, i = j - 1 = 6
   i. i = 6, compare key(1) with A[i], A[i] > key, A[i+1] = key = 9, i = i - 1 = 5: [0,1,2,3,8,9,9,5,7,6]
   ii. i = 5, compare key(1) with A[i], A[i] > key, A[i+1] = key = 8, i = i - 1 = 4: [0,1,2,3,8,8,9,5,7,6]
   iii. i = 4, compare key(1) with A[i], A[i] > key, A[i+1] = key = 3, i = i - 1 = 3: [0,1,2,3,3,8,9,5,7,6]
   iv. i = 3, compare key(1) with A[i], A[i] > key, A[i+1] = key = 2, i = i - 1 = 2: [0,1,2,2,3,8,9,5,7,6]
   v. i = 2, compare key(1) with A[i], A[i] = key, break, A[i+1] = key = 1: [0,1,1,2,3,8,9,5,7,6]
g. j = 8, key = A[2] = 5, i = j - 1 = 7
   i. i = 7, compare key(8) with A[i], A[i] > key, A[i+1] = key = 9, i = i - 1 = 6: [0,1,1,2,3,8,9,9,7,6]
   ii. i = 6, compare key(8) with A[i], A[i] > key, A[i+1] = key = 8, i = i - 1 = 5: [0,1,1,2,3,8,8,9,7,6]
   iii. i = 5, compare key(8) with A[i], A[i] < key, break, A[i+1] = key = 5: [0,1,1,2,3,5,8,9,7,6]
h. j = 9, key = A[2] = 7, i = j - 1 = 8
   i. i = 8, compare key(8) with A[i], A[i] > key, A[i+1] = key = 9, i = i - 1 = 7: [0,1,1,2,3,5,8,9,9,6]
   ii. i = 7, compare key(8) with A[i], A[i] > key, A[i+1] = key = 8, i = i - 1 = 6: [0,1,1,2,3,5,8,8,9,6]
   iii. i = 6, compare key(8) with A[i], A[i] < key, break, A[i+1] = key = 7: [0,1,1,2,3,5,7,8,9,6]
i. j = 10, key = A[2] = 6, i = j - 1 = 9
   i. i = 9, compare key(6) with A[i], A[i] > key, A[i+1] = key = 9, i = i - 1 = 8: [0,1,1,2,3,5,7,8,9,9]
   ii. i = 8, compare key(6) with A[i], A[i] > key, A[i+1] = key = 8, i = i - 1 = 7: [0,1,1,2,3,5,7,8,8,9]

        iii. i = 7, compare key(6) with A[i], A[i] > key, A[i+1] = key = 7, i = i - 1 = 6: [0,1,1,2,3,5,7,7,8,9]
        iv. i = 6, compare key(6) with A[i], A[i] < key, break, A[i+1] = key = 6: [0,1,1,2,3,5,6,7,8,9]

6. Answer:
    a.
    ```
    for i = 1 to n:
            if A[i] equals v
                    return i
    return NIL
    ```
    b. Prove
        i. Initialization: Before the first loop iteration, i = 1, subarray A[1...i-1] contains no element, so if there is an element equals to v, it must be in the subarray A[i...n]
        ii. Maintenance: subarray A[1...i-1] contains those elements that have been checked not equaled to v. If A[i] doesn't equal to v, i = i+1, and the checked element will be moved into subarray A[1...i-1], and the subarray still contains no-v elements.
        iii. Termination: If A[i] equals to v, then return i.