THE UNIVERSITY OF MELBOURNE

# COMP90038 Algorithms and Complexity SM2, 2018 Assignment 2

Peiyong Wang 955986

October 7, 2018

## 1 PROBLEM ONE

See Figure 1.1.

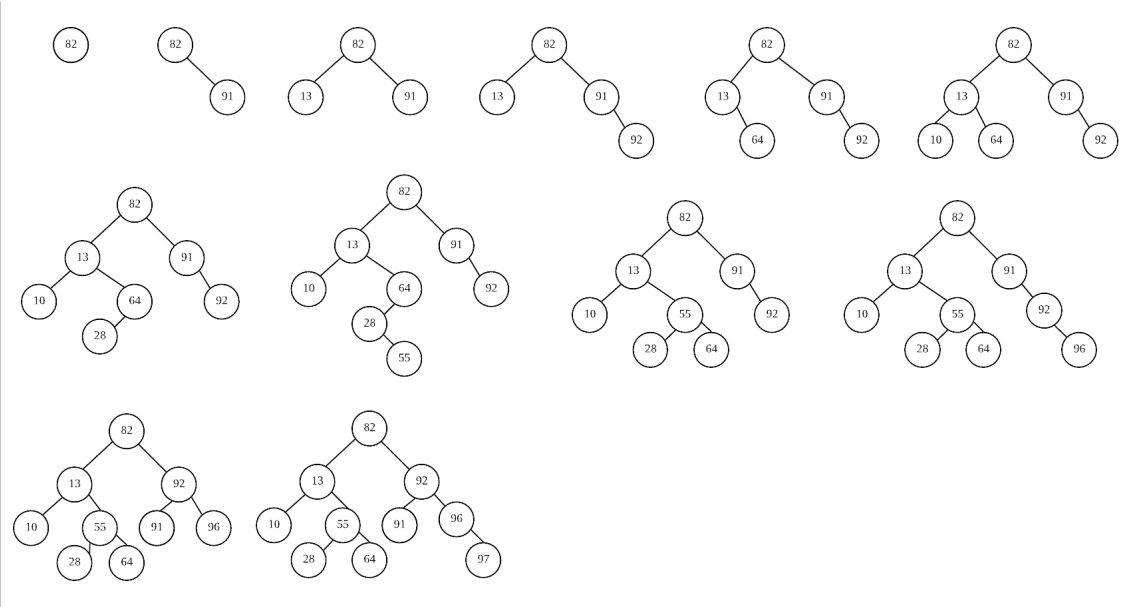


Figure 1.1: Answer for Problem 1

## 2 PROBLEM TWO

A See Algorithm 1.
The time complexity of Algorithm 1 should be O($min(mlog_2m+nlog_2m, mlog_2n+nlog_2n)$), where m and n are the size of X and Y, respectively.

B See Algorithm 2.
The time complexity of Algorithm 2 should be $O(m+n)$, where m and n are the size of X and Y, respectively.

**Algorithm 1** Find intersections of two sets with sorting

**Require:** Two sets of integers X and Y

```
1:  function FINDSETINTERSECTION(X, Y)
2:      I = []                                   ▷ initialize the intersection as empty
3:      if X.size ≥ Y.size then
4:          smallArray = MERGESORT(Y)
5:          largeArray = X
6:      else
7:          smallArray = MERGESORT(X)
8:          largeArray = Y
9:      end if
10:     for every element c in largeArray do
11:         searchResult = BINSEARCH(smallArray, smallArray.size,c)
12:                                  ▷ Call the BinSearch algorithm from lecture 10.
13:         if searchResult is not -1 then       ▷ If c exists in smallArray
14:             I.append(c)                      ▷ copy c to I
15:         end if
16:     end for
17:     return I
18: end function
```

**Algorithm 2** Find intersections of two sets with hashing

**Require:** Two sets of integers X and Y

```
1:  function FINDSETINTERSECTION(X, Y)
2:      I = []                                   ▷ initialize the intersection as empty
3:      S = HASHSET()                            ▷ initialize an empty hash set
4:      for every x in X do
5:          S.add(x)
6:      end for
7:      for every y in Y do
8:          if S.contains(y) then
9:              I.append(y)
10:         end if
11:     end for
12:     return I
13: end function
```

# 3 PROBLEM THREE

See Algorithm 3.
For a node i with children [1,2,3,...,n], of which childern 1 needs the most time to pass the message to all its childern and childern n needs the least time to pass the message to all its childern. So we can get through mathematical induction that the time that node i needs to pass the message to all its childern T(i) = n + max(T(n), T(n-1)-1, T(n-2)-2,...,T(1)-(n-1))
Time complexity is O($n^2 logn$).

**Algorithm 3** compute the minimumnumber of days required for the decision to be disclosed to all employee

```
Require: Employee tree C[][]
function MinDays(C[][])
    return MinDaysForEmployeeI(C[][],0)
end function
function MinDaysForEmployeeI(C[][], i)
    if C[i][0] is 0 then
        return 0
    end if
    childNodes = []                          ▷ initialize an empty to store child nodes of i
    if C[i][0] is not 0 then
        m = 1
        while C[i][m] is not -1 do
            childNodes.append(C[i][m])
            m = m+1
        end while
    end if
    days = [MinDaysForEmployeeI(C[][], x) for x in childNodes]
    days = QuickSort(days)
    for j from 0 to days.length - 1 do
        days[j] = days[j] - j
    end for
    dayI = C[i][0] + Max(days)               ▷ function Max() has time complexity O(3n/2)
    return dayI
end function
```

# 4 PROBLEM FOUR

A See Algorithm 4. The time complexity should be O(n), where n is the length of input array.

B See Algorithm 5. The time complexity should be O($2nlog_2 n + n$) = O($nlogn$)

**Algorithm 4** Find the nearest larger element

**Require:** An array of integers A[]

```
1: function NEXTLARGER(A[])
2:     res = list(n, -1)                    ▷ initialize a list of length n and all of its elements are -1
3:     monoStack = stack()                                                  ▷ initialize an empty stack
4:     for i from to A.size - 1 do
5:         while monoStack is not empty AND A[monoStack.top]<A[i] do
6:             res[monoStack.top] = i
7:             monoStack.pop
8:         end while
9:         monoStack.push(i)
10:    end for
11:    return res
12: end function
```

**Algorithm 5** For each of the elementA[i], find the minimumA[j] so thatA[j]> A[i] andj > i

**Require:** An array of integers A[]

```
1: function NEXTSMALLESTLARGER(A[])
2:     res = list(n, -1)                    ▷ initialize a list of length n and all of its elements are -1
3:     B = [][]                                                          ▷ initialize B as a 2-D list
4:     B = [[A[i],i] for i from 0 to A.length -1]          ▷ store A[i] and i, B = [[A[0],0],...,[A[n-1], n-1]]
5:     MERGESORT(B[·][0])                                              ▷ sort B according to values from A
6:     MINHEAPARRAY(B[·][0])   ▷ Turn B[·][0] into a min heap stored in an array, original index sift with
   values from A
7:     listOfJ = B[·][1]                          ▷ original index after sorted with elements from A
8:     monoStack = stack()                                                  ▷ initialize an empty stack
9:     for j from 0 to n-1 do
10:        while monoStack is not empty AND B[monoStack.top][1] < B[j][1] AND B[monoStack.top][0] <
   B[j][0] do
11:            res[B[monoStack.top][1]] = listOfJ[j]
12:            monoStack.pop
13:        end while
14:        monoStack.push(j)
15:    end for
16:    return res
17: end function
```