COMP3821 Assignment

Charlie Bradford z5114682

Question 1

> *Describe an O(n log n) algorithm (in the sense of the worst case performance) that, given an array S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.*

1. Presort the array using mergesort (nlogn)
2. Set i = S.length -1, and j = 0. (1)
3. Iterate through the list (n)
    - If S[i] + S[j] is greater than x, increase j
    - If the sum is less than x, decrease i
    - If it is equal, return True,
    - If i < j, return False.

```
function find_x (array S, num x)
    array T = sort(S)
    int i = T.length - 1
    int j = 0
    while ( i >= j )
        num sum = T[i] + T[j]
        if ( sum == x)
            return True
        elif ( sum > x )
```

```
            i = i - 1
        elif ( sum < x )
            j = j - 1
    return False
```

> *Describe an algorithm that accomplishes the same task, but runs in O(n) expected (average) time.*

1. Intialise a hash table H with all values in S mapped to False (n)
2. Iterate through n the the list (n)
    - If H[n] = False, set H[x-n] = True
    - If H[n] = True, return True
3. Return False if nothing is found (1)

```
function find_x_linear (array S, num x)
    hash H = {}
    for (n) in (S)
        H[n] = False
    for (n) in (S)
        if H[n] = True
            return True
        else
            H[n - x] = True
    return False
```

Question 2

> *You're given an array of n integers, and must answer a series of n queries, each of the form: "how many elements of the array have value between L and R?", where L and R are integers. Design an O(n log n) algorithm that answers all of these queries.*

1. Presort the array using mergesort (nlogn)
2. Create a hash H with each value in the array being a key for its index (n)
3. Return H[R] - H[L] - 1 for all queries (n)
    - The '- 1' is used to indicate a non-inclusive range
    - It can be changed to '+ 1' for an inclusive range

```
function elements_between_L_R (array S, array queries)
    array T = sort(S)
    hash H = {}
    for (int i = 0; i < S.length; i++)
        H[S[i]] = i

    array answers = []
    for (query) in (queries)
        num L = query[0]
        num R = query[1]
        answers.append(H[R] - H[L] - 1)

    return answers
```

Question 3

> *There are N teams in the local cricket competition and you happen to have N friends that keenly follow it. Each friend supports some subset (possibly all, or none) of the N teams. Not being the sporty type – but wanting to fit in nonetheless – you must decide for yourself some subset of teams (possibly all, or none) to support. You don’t want to be branded a copycat, so your subset must not be identical to anyone else’s. The trouble is, you don’t know which friends support which teams, so you can ask your friends some questions of the form “Does friend A support team B?” (you choose A and B before asking each question). Design an algorithm that determines a suitable subset of teams for you to support and asks as few questions as possible in doing so.*

1. Iterate simultaenously through both the list of teams and friends. (n)
2. Ask the Nth friend if they support the Nth team (1)
    - If they do, don’t support it
    - If they don’t, support the team

```
function find_team_subset (array friends, array teams)
    array supportedTeams = []
    for (friend, team) in (friends, teams)
        if NOT(friend supports team)
            supportedTeams.append(team)
    return supportedTeams
```

Question 4

> *Given n numbers x1, . . . , xn where each xi is a real number in the interval [0, 1], devise an algorithm that runs in linear time that outputs a permutation of the n numbers, say y1, . . . , yn, such that Pn i=2|yi − yi−1| < 2.*

1. Create n buckets (1)
2. Add each element s to the bucket indexed by floor(s * n) (n)
3. Print out each bucket in ascending order

```
function find_permutation (array S)
    listOfLists m
    int n = S.size
    for (s) (S)
        index = floor(s * n)
        m[index].append(s)
    for (i = 0; i < n; i++)
        print m[i].values()
```

Question 5

> *You are at a party attended by n people (not including yourself), and you suspect that there might be a celebrity present. A celebrity is someone known by everyone, but does not know anyone except herself/himself. (Of course everyone knows herself/himself). Your task*

> *is to work out if there is a celebrity present, and if so, which of the n people present is a celebrity. To do so, you can ask a person X if they know another person Y (where you choose X and Y when asking the question).*

> *Show that your task can always be accomplished by asking no more than 3n − 3 such questions, even in the worst case.*

1. Store all guests in a stack (No questions)
2. Pop two guests, A and B from the stack (n - 1 questions)
    1. Ask if A knows B
        - If A knows B, discard A and pop a new guest
        - If A doesn’t know B, discard B and pop a new guest
    2. Repeat until only 1 guest is left
    3. This guest is our potential celebrity (PC)
    - This is the discovery phase
3. Iterate through the list of all guests that aren’t PC (2n - 2 questions)
    - Ask the guest if they know the PC
        - If no, return False
    - Ask the PT if they know the guest
        - If yes, return False
    - This is the verification phase
4. The PC is now a verified celebrity, return them

```
function find_celebrity (array guests)
    stack S

    for (g) in (guests)
        S.push(g)

    guest A = S.pop()
    guest B = S.pop()
    while (S.size > 0)
        if (A knows B)
            A = S.pop()
        else
            B = S.pop()

    if (A knows B)
        guest potentialCelebrity = B
    else
        guest potentialCelebrity = A

    for (g) in (guests)
        if (g == potentialCelebrity)
            continue
        elif ((potentialCelebrity knows g) OR NOT(g knows pot
entialCelebrity))
            return False

    return potentialCelebrity
```

> *Show that your task can always be accomplished by asking no more than 3n − log2 n − 2 such questions, even in the worst case.*

1. Arrange the guests in a tournament tree for the discovery phase to find the potential celebrity (n - 1 questions)
    - Record answers
    - The celebrity is asked or refered to in at least floor(logn) questions
    - Use a queue in implementation
2. During the verification phase use recorded answers to minimise questions asked (2n - 1 - floor(logn) questions)

```
function find_celebrity (array guests)
    queue Q
    for (g) in (guests)
        Q.append(g)

    n = guests.length
    matrix[n][n] M = map(0) // n*n matrix of all values 0

    while ( Q.size > 1 )
        A = Q.pop()
        B = Q.pop()
        if ( A knows B )
            M[A][B] = True
            Q.append(B)
        else
            M[A][B] = False
            Q.append(A)

    guest PC = Q.pop()


    for (g) in (guests)
        if ( g == PC )
            continue

        boolean gKnowsPC
        boolean PCKnowsG

        if ( M[g][PC] == 0 )
            gKnowsPC = (g knows PC) // Question asked
        else
            gKnowsPC = M[g][PC] // Question not asked

        if ( M[PC][g] == 0 )
            PCKnowsG = (PC knows g)
        else
            PCKnowsG = M[PC][g]

        if ((PCKnowsG) OR NOT(gKnowsPC))
            return False

    return potentialCelebrity
```

Question 6

> *You are conducting an election among a class of n students. Each*

> *student casts precisely one vote by writing their name, and that of their chosen classmate on a single piece of paper. However, the students have forgotten to specify the order of names on each piece of paper – for instance, "Alice Bob" could mean Alice voted for Bob, or Bob voted for Alice!*

> *Show how you can still uniquely determine how many votes each student received.*

Each student votes once, so their own name appears at least once. However, if their name appears more than once, then the other names must be votes for the student. So if a students name appears x times in the ballots, that student recieved x - 1 votes.

> *Hence, explain how you can determine which students did not receive any votes. Can you determine who these students voted for?*

If a student recieved no votes, then the only time their name appears in the ballots is when they voted. Therefore any student whose name only appears once in the ballots did not recieve any votes.

> *Suppose every student received at least one vote. What is the maximum possible number of votes received by any student? Justify your answer.*

One. A group of n students can cast n votes. If all n students get one vote then there are no votes left so no student can get more than one vote.

> *Using parts (b) and (c), or otherwise, design an algorithm that constructs a list of votes of the form "X voted for Y " consistent with the pieces of paper. Specifically, each piece of paper should match up with precisely one of these votes. If multiple such lists exist, produce any. An O(n^2) algorithm earns 7 marks, and an O(n) algorithm earns an additional 3 marks.*

1. Create a hash H with all student names included, with each value intialised to 0 (n)
2. Count the number of times each name appears in the ballot, add these to the hash (n)
3. Check if everyone got votes (n)
    - If everyone got votes iterate through the list of ballots and print out the names of the students separated by ' voted for '
4. Otherwise, initialise a queue with all the ballots (n)
5. For every ballot nameA nameB: (n^2)
    - If H[nameA] == 1

      Print 'nameA voted for nameB'

      Subtract 1 from H[nameB] and H[nameA]
    - If H[nameB] == 1

      Print 'nameB voted for nameA'

      Subtract 1 from H[nameA] and H[nameB]
    - If nameA == nameB

      Print 'nameA voted for nameA'

      Subtact 1 from H[nameA]
    - Else append the ballot to the end of the queue

    - The hash keeps track of votes that haven’t been accounted for

```
function attribute_votes (array ballots)
    set students = <>
    for (ballot) in (ballots)
        string studentA, studentB = ballot.split(' ')
        students.append(studentA)
        students.append(studentB)

    hash H = {}
    for (student) in (students)
        H[student] = 0

    queue Q
    for (ballot) in (ballots)
        Q.append(ballot)
        string studentA, studentB = ballot.split(' ')
        H[studentA]++
        H[studentB]++

    boolean everyoneReceivedVotes = True
    for (student) in (students)
        if H[student] == 1
            everyoneReceivedVotes = False

    if (everyoneRecievedVotes)
        for (ballot) in (ballots)
```

```
            print ballot.replace(' ', ' voted for ')
    else
        while ( Q.size > 0 )
            string ballot = Q.pop()
            string studentA, studentB = ballot.split(' ')
            if ( studentA == studentB )
                print (studentA + ' voted for ' + studentA)
                H[studentA]--
            elif ( H[studentA] == 1 )
                print ( studentA + ' voted for ' + studentB )
                H[studentB]--
            elif ( H[studentB] == 1 )
                print ( studentB + ' voted for ' + studentA )
            else
                Q.append(ballot)

    return NULL
```