COMP3121 - Assignment 1

Mark Pollock - z3372890 - mpol525

Due: 26th March 2015

1 Question 1

1. Divide the coins into 3 piles of 4 coins. Let these be called $P1$, $P2$ and $P3$.
2. Weigh $P1$ vs $P2$ on the pan balance. This gives 2 outcomes:
   (a) **Outcome 1** If $P1$ matches $P2$, the counterfeit coin is in $P3$. Thus:
      i. Weigh $3a + 3b + 3c$ vs 3 coins from $P2$ (which we know are genuine).
         A. If this weigh is even then the counterfeit coin must be $3d$ and we simply weigh this against a coin from $P2$ to see whether it is lighter or heavier.
         B. **alternatively** : if this weigh shows that the 3 coins from $P3$ are different then we know that the counterfeit coin is $3a$, $3b$, or $3c$ and we know whether it is lighter or heavier. Weigh $3a$ vs $3b$: If they are different then the counterfeit is the coin which is either heavier or lighter than the other depending on whether the previous result showed that the counterfeit coin was heavier or lighter. If they are the same then the counterfeit coin is $3c$ and it is lighter or heavier depending on the previous result.
   (b) **Outcome 2** If $P2$ differs from $P1$ then the counterfeit is in $P1$ or $P2$. Note: Here we make the assumption that $P2$ is lighter than $P1$ - this can be made since we can rename the piles after this first weigh. Thus:
      i. Weigh $1a + 1b + 2a + 2b$ vs $2c$ and 3 coins from $P3$ (which we know are genuine).
         A. If this weigh is even then the counterfeit coin must be $1c$, $1d$ or $2d$. We then weigh $1c + 2d$ against 2 coins from $P3$. If this weigh is the same then $1d$ must be the counterfeit coin and it is heavier (see assumption above). If this weigh is heavier than $1c$ is the counterfeit and if this weigh is lighter than $2d$ is the counterfeit.
         B. **alternatively** If this weigh shows that $1a + 1b + 2a + 2b$ is heavier than $2c +$ 3 coins from $P3$ then the counterfeit coin must be $1a$, $1b$ or $2c$ and we weigh $1a + 2c$ against 2 genuine coins. If this weigh is the same then the counterfeit coin is $1b$ and it is heavier. If this weigh is heavier than the counterfeit coin is $1a$ and it is heavier. If this weigh is lighter than the counterfeit is $2c$ and it is lighter.
         C. **Finally,** if this weigh shows that $1a + 1b + 2a + 2b$ is lighter than $2c + 3$ coins from $P3$ then the counterfeit coin must be $2a$ or $2b$. We simply weigh them against each other and the lighter one is the counterfeit (and is clearly lighter).

Note: I verified this algorithm using c++ and this code is appended to my assignment.

2 Question 2

This question was ambiguous since it said there were n thieves yet only 2 remaining thieves. Hence I have solved the harder, more general case of n thieves.
Let there be n thieves and let us denote them by $T1$, $T2$, ..., Tn. The algorithm is then as such:

1. $T1$ makes a pile of what he believes is $\frac{1}{n}$ of the worth of the entire items.
2. $T2$ then has the option of accepting that this is a fair $\frac{1}{n}$ portion or declining.
   (a) In the case of declining, he believes that the pile made by the first thief is larger (has more worth) than $\frac{1}{n}$. Thus $T2$ adjusts the pile (by making it smaller) to what he thinks is a fair $\frac{1}{n}$ pile. **Note: now whoever gets this pile, $T1$ is happy since they are getting less than what he considered to be a fair $\frac{1}{n}$ fraction.**
   (b) In the case of accepting, the proposal is then put forward to the next thief. **Note: now $T2$ is happy whoever gets the pile since he believes it is a fair $\frac{1}{n}$ fraction**
3. $T3$ then has the option of accepting or declining.
4. Step 2 is repeated until all thieves have had an option to accept or decline the proposal. Then the last thief to decline the pile and adjust it to what they believe was a fair $\frac{1}{n}$ pile, keeps that pile. Now the problem has been reduced to $n-1$ thieves and we repeat the entire process on the $n-1$ case.

3 Question 3

Assumption: Everyone had met their partner (wife / husband) before the party.

1. Since Tom got 19 **different** answers and no one could shake more than 18 hands (since there was only 19 other people at the party and 1 of these people would be their partner - see assumption), we can conclude (via the pigeon - hole principle) that the answers he got were: $0, 1, 2, 3, ..., 16, 17, 18$.
2. Now the person who shook 18 hands must have shook everyone's hand except their partners. Therefore we can conclude that their partner must have been the one who answered 0 (since no one else could have shook 0 hands). Since Tom did not ask himself we can conclude that this couple cannot be Tom and his wife. Thus we can eliminate this couple from the problem and to do this we must take 1 handshake from everyone (since the person who shook everyone's hand is being eliminated from problem).
   Graphically we can depict this as:

   $$\cancel{0},\ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,\ \cancel{18}$$

   Then getting rid of the handshakes that are due to the eliminated couple result in a simplified problem with 17 different answers and these being:

   $$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16$$

3. We go on repeating this process continually eliminating couples and simplifying the problem further until there is only 1 answer left (of 0 in the simplified problem). This must be Tom's wife and thus in this simplified problem Tom has also shook 0 hands.

4. We then expand the problem adding in each couple 1 by 1 so that the number of hands Tom (and his wife) have shaken increase by 1 for each additional couple.

5. **Since there are 9 couples to be added in, Tom must have shook 9 hands.**

4 Question 4

Assumption: The outer highway is anti-clockwise. If this was to be clockwise the algorithm will work if every instance of *left* or *left-most* is replaced by *right* and *right-most* respectively.

1. Start at any point on the outer highway travelling in the direction of the highway.

2. Take the first possible left turn (i.e. the one-way street is facing the correct direction) and remember this intersection (vertex).

3. Travel along until the next intersection is reached.

   (a) If the left-most turn is possible take it and remember this intersection (vertex).

   (b) If the left-most turn is not possible forget all intersections and take the most left path that *is* possible.

4. Repeat step 3 until a remembered intersection is reached. Once this occurs, the block that lies to the left of the most recent path is able to be circumnavigated. In-fact the algorithm has just circumnavigated it.

5 Question 5

Note: Assuming the pirates know the positions of the other pirates in the line as well as their own.

Consider 2 pirates: The second pirate will not except anything the first pirate proposes and the first pirate will die.

Consider 3 pirates: The 2nd pirate will agree with any proposal put forward by the first pirate since otherwise it leads to 2 pirates and he will die. Therefore the 1st pirate would propose 100 bars to himself.

Consider 4 pirates: Since with 3 pirates, the last two pirates will not get any gold (explained above), the first pirate only needs to offer the last two pirates 1 bar each to have them except his proposal. Therefore the 1st pirate would propose 98 bars to himself and 1 each to the 3rd and 4th pirate.

Consider 5 pirates: Similarly, since with 4 pirates there is a pirate who will not get any gold and the 4th and 5th pirates will get 1 bar each. Therefore the 1st pirate only needs to offer 1 bar to the pirate who would get no gold in the 4 pirates case and 2 bars to either the 4th or 5th pirate. **i.e. He would propose 97 bars to himself, 1 bar to pirate #3 and 2 bars to pirate #4 (or #5).**

Explanation: Pirate #3 will accept since otherwise pirate #1 will be killed and we go to the 4 pirates case in which he will receive no gold. Pirate #4 (or #5) will accept since with 4 pirates he only gets 1 bar. Thus we have a majority vote.

6 Question 6

6.1 6a

The way in which I have completed this question is:
Pseudocode:

```
mergeSort(S)
for every element (e) in S
    declare y as int
    y = x - e
    if(binarySearch(S, y))
        print ("There does exist two elements in S whose sum is x. These are:" + e + "and" + y)
        end
print("There does not exist two elements in S whose sum is x.")
```

Note: The functions mergeSort(array a) and binarySearch(array a, int i) have been explained in question 10 (with pseudocode). These functions run in $\Theta(n\ log_2(n))$ and $\Theta(log_2 n(n)$ time respectively. As such, the above algorithm will run in $\Theta(n\ log_2(n))$ time as required.

6.2 6b

Pseudocode:

```
declare hm as an empty hashMap
for every element e in S
    declare y as an int
    y = x - e
    if (y is in hm)
        print("There does exist two elements in S whose sum is x. These are:" + e + "and" + y)
        end
    else
        add e to hm

print("There does not exist two elements in S whose sum is x.")
```

Explanation: This algorithm is of time complexity $O(n)$ since reading and writing to a hash map is of $O(1)$ and we simply iterate through the array element by element which is of $O(n)$.

7 Question 7

My solution to this question is:

- Convert each number to base n and store these in arrays of length $7^{\dagger}$. 1 array per number. - $O(n)$.
- Radix sort the arrays (each element in the array holds a digit of the number in base n). - $O(n)$.

† - We can show that a number smaller than n^7 can have, at most, 7 digits:
Let D be the number of digits.

$$\begin{aligned} D &= \lfloor log_b(maxNum) \rfloor + 1 \\ &= \lfloor log_n(n^7 - 1) \rfloor + 1 \\ &= \lfloor log_n(n^7) - \varepsilon \rfloor + 1 \\ &= \lfloor 7log_n n - \varepsilon \rfloor + 1 \\ &= \lfloor 7 - \varepsilon \rfloor + 1 \\ &= 6 + 1 \\ &= 7 \end{aligned}$$

Pseudocode:

Note: Assume that the numbers are in-putted via an array A (of size n).

```
struct numBaseN{
    array[int] baseN
    int decimal
}

declare constant int n = A.size()
declare numBaseN B[n]

for i = 0 to n - 1
    declare num as int
    num = A[i]
    declare dig as int
    declare baseNNum as numBaseN
    baseNNum.decimal = num
    for j = 0 to 7
        dig = num mod n
        num = num / n
        baseNNum.baseN[j] = dig
    B[i] = baseNNum


declare digit as int
for digit = 0 to 6
// Perform counting sort on each digit place
    declare numBaseN tempArray[n]      declare int C[n]
    for i = 0 to n - 1
        C[i] = 0
    for j = 0 to n - 1
        C[B[j].baseN[digit]]++
    // C[i] now contains the number of elements equal to i
    for i = 1 to n - 1
        C[i] = C[i] + C[i - 1]
    // C[i] now contains the number of elements less than or equal to i
```

```
    for j = n to 1
        tempArray[C[B[j].baseN[digit]]] = B[j]
        C[B[j].baseN[digit]]–
    B = tempArray
//At this point, B holds the numBaseN's (which represent the decimal numbers) in correct order.

for i = 0 to n - 1
    A[i] = B[i].decimal
// At this point, A has been re-arranged to the correct order
```

8 Question 8

My solution to this problem is as such:

1. Bucket-sort the original array into another array with n buckets, each containing link lists (always insert into the start of the linked list). - $O(n)$
2. For each bucket in this new array, output the contents in the order they come. - $O(n)$ for each bucket in worst case.

Explanation: This will ensure that the required condition ($\Sigma_{i=2}^{n}|y_i - y_{i-1}| < 2$) is met due to the following logic:

The maximum difference between any 2 elements in the same bucket is the size of that bucket which is $\frac{1}{n}$. i.e. $Diff_{i,max} = \frac{1}{n}$ where $Diff_i$ represents the difference between two elements in bucket i.

Thus the maximum sum of differences that can be accrued by numbers in the one bucket (in any order) is: $\frac{1}{n} \times k < \frac{k}{n}$ where k represents the total number of numbers in the bucket.

The sum of these maximum sums is thus: $\Sigma\frac{k_i}{n} < \frac{\Sigma k_i}{n} = \frac{n}{n} = 1$

Then we must also add the maximum sum accrued from bucket to bucket throughout the entire array. Since the bins are in order, it is clear that this is equal to 1.

Thus, the total sum of difference between each number in order is: $\Sigma_{i=2}^{n}|y_i - y_{i-1}| < 1 + 1 = 2$ as required.

Pseudocode:

```
declare B as an array of linked list of size n
for i = 0 to n - 1
    B[i] = empty linked list
for each element e in input
    insert e into the head of list B[⌊n × e⌋]
concatenate the lists B[0], B[1], ..., B[n - 1] together in order.
```

9 Question 9

This question is solved by:

1. Every kid tries on shoe S_1. The first kid to fit these shoes keeps them and they are the root node N_0 of the binary tree. Every kid who found $S1$ too large, is put into $G1a$ which is the left child of N_0. Every kid who found that S_1 fit or was too small is put into $G1b$ which is the right child of N_0.
2. N_0 tries on the second pair of shoes S_2. If the shoes are too small for N_0, $G1a$ repeats step 1 with S_2. Else, if the shoes are right size or too large for N_0, $G1b$ repeats step 1 with S_2.
3. This repeats until all kids have a pair of shoes (and are nodes of the binary tree).

Expected Number:
The expected number of shoe trials is the required $O(nlogn)$ since building a binary tree in this way is $O(nlogn)$ and in-fact best case it will be $O(nlog_2n)$ since the binary tree will be perfectly balanced - i.e. the group is split into two equal parts in each iteration. However, in worst case (largest shoe picked first then second largest and so on) it will be $O(\Sigma_{i=1}^{n} n) = O(n^2)$. Thus, for this problem it is very important that the question uses the words "expected number".

Walk through example:

- Consider 10 kids ($k_1..k_{10}$) who have their corresponding shoes ($S_1..S_{10}$) where S_1 belongs to k_1 and so on. And for simplicity of this walk through lets assume that no-one has the same size and that the kids have been numbered such that $S_1 < S_2 < S_3 < S_4 < S_5 < S_6 < S_7 < S_8 < S_9 < S_{10}$.
- Say we pick S_6 as the first pair. Each kid tries on S_6 and we find that k_6 fits, $k_1..k_5$ find it too large and $k_7..k_{10}$ find it too small.
  - k_6 keeps the shoes that fit him (S_6) and he becomes the root node of the tree.
  - $k_1..k_5$ form the left hand child.
  - $k_7..k_{10}$ form the right hand child.
- We then pick our next pair of shoes - say S_4.
- We start at the root node (k_6) who tries on S_4 and reports they are **too small**.
- Thus we take the shoes to the left hand child and each kid in the left hand child tries on S_4.
- We find that k_4 fits, $k_1..k_3$ find it too large and k_5 finds it too small.
  - k_4 keeps the shoes that fit him (S_4) and he becomes the left hand child node of the root node - lets call this N_{1-left}.
  - $k_1..k_3$ form the left hand child of N_{1-left}.
  - k_5 forms the right hand child of N_{1-left}
- At this stage we have a binary tree that looks like Figure 9a.

- We go on repeating until all shoes have been taken by a child who fits them.

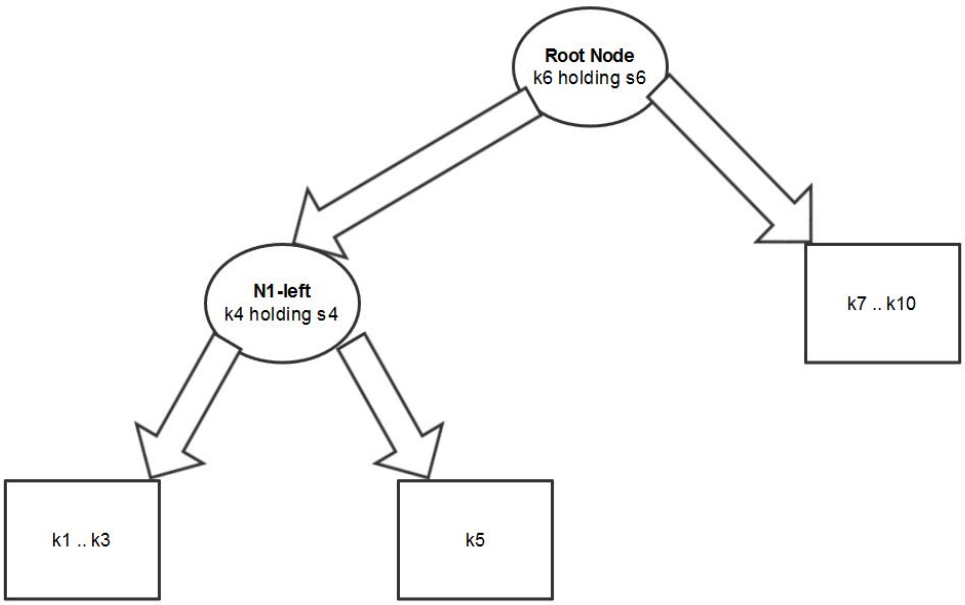


Figure 9a - Binary Tree showing the structure of the tree after the steps explained in the walk through above

Pseudocode:
Notes:

- Let's assume that shoes come in a queue S and the kids come in an array K.

- kid is a data structure that has a public function that takes in a shoe and returns whether the kid finds the shoe too small or too large or fits.

```
struct treeNode
    array[kid] group
    treeNode pointer left
    treeNode pointer right
    kid child
    shoe s
end struct


declare tree as a Tree
declare root as new treeNode
for each kid k in K
    add k to root.group
```

```
while(S is not empty)
    declare S_curr as a shoe
    S_curr = next from S
    remove next from S
    declare curr_Node as treeNode
    curr_Node = tree.root

    while (curr_Node.group is empty)
        declare left_or_right as int
        left_or_right = curr_Node.kid.tryShoe(S_curr)
        if(left_or_right = LEFT)
            curr_Node = curr_Node.left
        else
            curr_Node = curr_Node.right


    declare left as new treeNode
    declare right as new treeNode
    curr_Node.left = left
    curr_Node.right = right

    for each kid k in curr_Node.group
        remove k from curr_Node.group
        if(k.tryShoe(S_curr) == RIGHTSIZE AND curr_Node.child == NULL)
            curr_Node.child = k
            curr_Node.s = S_curr
        else if(k.tryShoe(S_curr) == TOOBIG)
            add k to left.group
        else
            add k to right.group
```

10 Question 10

The way in which I have completed this question is basically:

1. mergeSort array 1 ($\Theta(n\ log_2(n))$ time)
2. for each element in array 2, binary search array 1 for matching element ($\Theta(log_2(n))$ time for each element $\therefore \Theta(n\ log_2(n)$ for step 2.

Thus, the entire algorithm is in $\Theta(n\ log_2(n) + n\ log_2(n)) = \Theta(n\ log_2(n))$ time.
Now, in more detail the algorithm is as follows:

Let the arrays be array1 and array2

Pseudocode:

```
mergesort (array1)
for i = 0 .. array2.size()
    if(binarySearch(array1, array2[i])
        print ("Arrays have an element in common - " + i)

print ("Arrays do not have an element in common")

function bool binarySearch(array a, int i)
declare index as int
declare maxIndex as a.size() - 1
declare minIndex as 0

index = a.size() / 2 rounded up to an integer
    while(i ≠ a[index] AND maxIndex ≠ minIndex)
        if(i > a[index])
            minIndex = index
            index = (index + maxIndex) / 2 rounded up to nearest integer
        else
            maxIndex = index
            index = (index + minIndex) / 2 rounded down to nearest integer

    if (i = a[index])
        return true
    else
        return false

end function

function mergesort (array a)
    if(a.size() == 1) return a
    declare l1 as array = a[0] ... a[n/2]
    declare l2 as array = a[n/2 + 1] ... a[n]

    l1 = mergesort (l1);
    l2 = mergesort (l2);

    return merge (l1, l2)
end function

function merge (array a, array b)
    declare c as array of size a.size() + b.size()

    while (a and b have elements)
        if (a[0] > b[0])
            add b[0] to the end of c
            remove b[0] from b
        else
            add a[0] to the end of c
```

```
            remove a[0] from a
    while (a has elements)
        add a[0] to the end of c
        remove a[0] from a
    while (b has elements)
        add b[0] to the end of c
        remove b[0] from b
    return c
```

end function

11 Question 11

My $\Theta(n)$ algorithm is as follows:

Pseudocode:

```
declare bool array B[100];
Set all elements of B to false;        // this is of Θ(n) time
for i = 0 to 99:                       // this for loop is also of Θ(n) time
    if(B[A[i]])
    return A[i];
else
    B[A[i]] = true;
```

At worst case the algorithm will have to go through every element in A (i.e. the duplicated value is the last value in A). Where as the average case will have the duplicated value in the centre of A.

Thus this algorithm clearly runs in $\Theta(n)$.

12 Question 12

Note: I have solved this problem via recursion.
Let the stations be numbered 1 to 11 and let F_i denote the distance the petrol available at station i (Si) will allow you to travel and D_i denote the distance until the next petrol station $(i + 1)$.

Consider the 2 petrol station case:
Since $F_1 + F_2 \geq D_1 + D_2$ (*"It is known that the total quantity of petrol on all stations is enough to go around the highway once..."*)
then, $F_1 \geq D_1$ **OR** $F_2 \geq D_2$.
Thus, there is a station that has enough fuel to reach the other station. At which point all available fuel is collected and hence it must be possible to complete the trip around the highway.

Consider the 3 petrol station case:
Since $F_1 + F_2 + F_3 \geq D_1 + D_2 + D_3$ then, $F_1 \geq D_1$ **OR** $F_2 \geq D_2$ **OR** $F_3 \geq D_3$.
Thus, there is a station that has enough fuel to reach the next station. For clarity lets name these

stations A and B. At this point we can eliminate B from the problem and add the fuel from B to the fuel already available at A.
Note: This does not change the problem since we know that if we are at station A we could reach B and fill up thus being able to get the same total distance from A as if all the fuel was at station A (and none at B).
Thus, the problem then becomes the 2 petrol station case, which is proved to be possible above.

Consider the n petrol station case:
Since $\sum F_i \geq \sum D_i$ then, $\exists i | F_i \geq D_i$. That is, there exists a petrol station (S_i) that has enough petrol to travel to the next station (S_{i+1}). **Note:** if $i+1$ is greater than n then it should be replaced with 1 (since the petrol stations are in a circle).

As such, we can eliminate petrol station $i+1$ from the problem and move its fuel to station i (see above case for explanation). Thus, we have simplified the problem to the $n-1$ case. We continue doing this until we reach the 2 station case which is proved above.

Hence, we have proved the statement for the general case and hence for the $n = 11$ case as required.

13 Question 13

13.1 13a

$$\begin{aligned} g(n) &= (n - 2log(n))(n + cos(n)) \\ &= n^2 + ncosn - 2nlogn - 2(cosn)(logn) \\ &= O(n^2) \end{aligned}$$

Thus for any large value of n, n^2 will dominate $g(n)$ and hence we can conclude that:

$$f(n) = \Theta(g(n))$$

13.2 13b

$$\begin{aligned} g(n) &= log(n^{logn}) + 2logn \\ &= (logn)(logn) + 2logn && (by\ log\ laws) \\ &= (logn)^2 + 2logn \\ &= O((logn)^2) \end{aligned}$$

Thus we can conclude that:

$$f(n) = \Theta(g(n))$$

13.3 13c

Consider the expression in the power of f(n). i.e. $\frac{1+sin(\frac{\pi n}{2})}{2}$. Now for discrete, integer values of n, $sin(\frac{\pi n}{2})$ progresses in the following pattern: 0, 1, 0, -1, 0, 1, 0, -1, 0, ...

Thus the above expression progresses in the following pattern:

$$\frac{1}{2}, 1, \frac{1}{2}, 0, \frac{1}{2}, 1, \frac{1}{2}, 0, \frac{1}{2}, ...$$

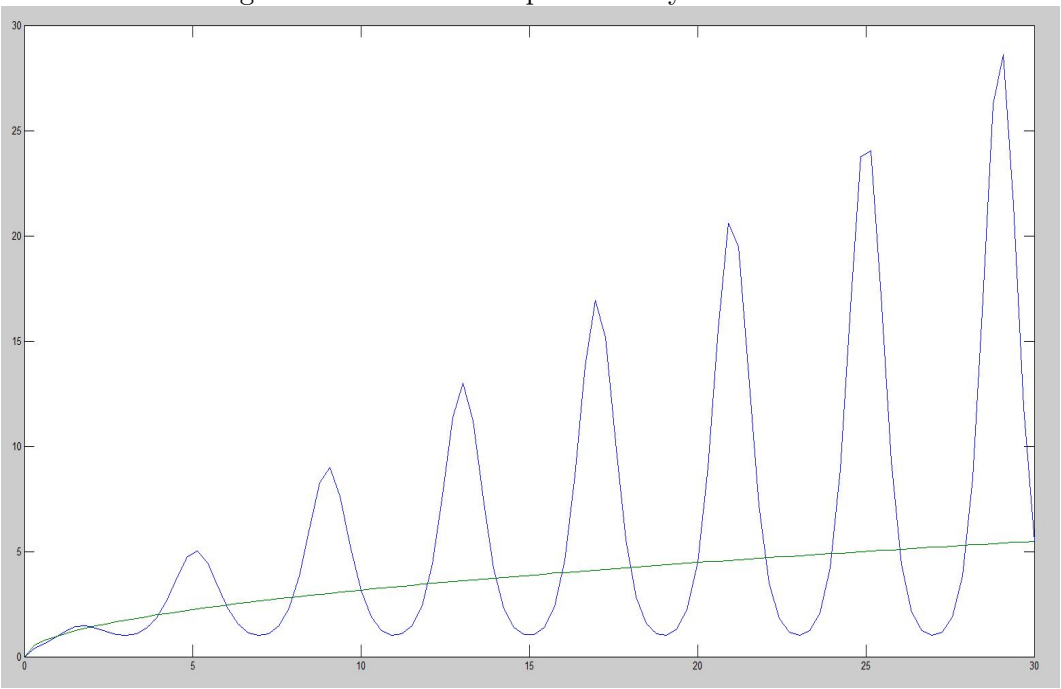
And thus $f(n)$ progresses in the following pattern:

$$n^{\frac{1}{2}},\ n^1,\ n^{\frac{1}{2}},\ n^0,\ n^{\frac{1}{2}},\ n^1,\ n^{\frac{1}{2}},\ ... = \sqrt{n},\ n,\ \sqrt{n},\ 1,\ \sqrt{n},\ n,\ \sqrt{n},\ 1,\ \sqrt{n},\ ...$$

By inspecting this it can be seen that we have 3 cases:

- **Case 1:** $f(n) = \sqrt{n}$
  In this case it is clear that $f(n) = g(n)$ and hence:
  $$f(n) = \Theta(g(n))$$
- **Case 2:** $f(n) = n$
  In this case it is clear that $f(n)$ grows much more quickly then $g(n)$ and hence:
  $$f(n) = \Omega(g(n))$$
- **Case 3:** $f(n) = 1$
  It is clear that $g(n)$ grows much more quickly than $f(n)$ and hence:
  $$f(n) = O(g(n))$$

From the above, it is shown that $f(n)$ can not be related to $g(n)$ via any of $\Omega,\ \Theta,\ O$ notations. This is shown in the figure below which was produced by matlab.



Further explanation:
Logically, the reason that f(n) cannot be expressed as growing more quickly, less quickly or at the same rate then g(n) is that:

- Consider $f(n) = \Omega(g(n))$.
  So this means that $f(n) > k \times g(n)$ for some k.
  However we cannot choose this k, since even if we choose the smallest number we can think of, there will be an n large enough that $g(n) > 1$ and such $f(n) < g(n)$.

  Thus, $f(n) = \Omega(g(n))$ must be false.

- Consider $f(n) = O(g(n))$.
  This means that $f(n) < k \times g(n)$ for some k.
  However we cannot choose a k that satisfies this. Even if we choose the largest constant number we can think of, there will be an n large enough that $f(n) > k \times g(n)$ (this will occur in the $f(n) = n$ case).

  Thus, $f(n) = O(g(n))$ must be false.

- Consider $f(n) = \Theta(g(n))$.
  So this means that $f(n) > k_1 \times g(n)$ AND $f(n) < k_2 \times g(n)$ for some k_1 and k_2.
  But using the same logic used in the Ω and O cases, we can show that both of these equations will be false for some cases of n.

  Thus, $f(n) = \Theta(g(n))$ must be false.

14 Appendix

14.1 c++ Code For Question 1

14.1.1 weighPan.ccp

```
//============================================================
// Name        : weighPan.cpp
// Author      : mark pollock
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================

#include <iostream>
#include <list>

#include "weigh.h"
#include "find_solution.h"

using namespace std;


\#define LIGHTER 1
\#define HEAVIER 2
```

```
int main() {
        // second integer is the coin number and the
        first integer says whether it is heavier or lighter
        for(int i = 1; i < 13; ++i){
                pair<int, int> p = make_pair<int, int>(LIGHTER, i);
                Weigh w(p);
                Find_solution sol = Find_solution(w);
                std::pair<int, int> s = sol.solve_problem();
        }
        for(int i = 1; i < 13; ++i){
                pair<int, int> p = make_pair<int, int>(HEAVIER, i);
                Weigh w(p);
                Find_solution sol = Find_solution(w);
                std::pair<int, int> s = sol.solve_problem();
        }

        return 0;
}
```

14.1.2 weigh.h

```
#ifndef __WEIGH_H__
#define __WEIGH_H__
#include <vector>

class Weigh {

        public:
                Weigh() {};
                Weigh(std::pair<int, int>);
                int weigh_action(std::vector<int> l, std::vector<int> r);


        private:
                std::pair<int, int> p;

};

#endif
```

14.1.3 weigh.ccp

```
#include <iostream>
#include <vector>
#include <algorithm>

#include "weigh.h"


using namespace std;

#define LEFT 1
#define RIGHT 2
#define SAME 3
#define LIGHTER 1
#define HEAVIER 2



Weigh::Weigh(std::pair<int, int> p2){
        p = p2;
}


// returns which is heavier
int Weigh::weigh_action(vector<int> l, vector<int> r){
        int coin = p.second;

        if (find(l.begin(), l.end(), coin) != l.end()){
                // coin is in left
                if(p.first == LIGHTER)
                        return RIGHT;
                else
                        return LEFT;
        }
        else if(find(r.begin(), r.end(), coin) != r.end()){
```

```
                //coin is in right
                if(p.first == LIGHTER)
                        return LEFT;
                else
                        return RIGHT;
        }else
                return SAME;
}
```

14.1.4 find_solution.h

```
#ifndef __FIND_SOLUTION_H__
#define __FIND_SOLUTION_H__

#include "weigh.h"

class Find_solution {

        public:
                Find_solution(Weigh);


                std::pair<int, int> solve_problem();

        private:
                Weigh w;
};


#endif
```

14.1.5 find_solution.ccp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "find_solution.h"
#include "weigh.h"

#define LEFT 1
#define RIGHT 2
#define SAME 3
#define LIGHTER 1
#define HEAVIER 2

 Find_solution::Find_solution(Weigh w){
  this->w = w;
 }

 std::pair<int, int> Find_solution::solve_problem(){
  int coin;
  int diff;

  std::vector<int> l;
  std::vector<int> r;

  for(int i = 0; i < 5; ++i)
   l.push_back(i);
```

```
for(int i = 5; i < 9; ++i)
 r.push_back(i);

//outcome 1 - P1 matches P2
int result0 = w.weigh_action(l, r);

if(result0 == SAME){
 l.clear();
 r.clear();
 l.push_back(9);
 l.push_back(10);
 l.push_back(11);
 r.push_back(1);
 r.push_back(2);
 r.push_back(3);

 int result = w.weigh_action(l, r);
 if(result == SAME){
  //counterfeit is coin 12
  coin = 12;
  l.clear();
  r.clear();
  l.push_back(1);
  r.push_back(12);
  if(w.weigh_action(l, r) == LEFT){
   std::cout << "coin 12 is lighter" << std::endl;
   diff = LIGHTER;
  }else{
   std::cout << "coin 12 is heavier" << std::endl;
   diff = HEAVIER;
  }
 }else{
  l.clear();
  r.clear();
  l.push_back(9);
  r.push_back(10);
  int result2 = w.weigh_action(l, r);
  if(result == LEFT && result2 == LEFT){
   std::cout << "coin 9 is heavier" << std::endl;
   coin = 9;
   diff = HEAVIER;
  }else if(result == LEFT && result2 == RIGHT){
   std::cout << "coin 10 is heavier" << std::endl;
   coin = 10;
   diff = HEAVIER;
  }else if(result == RIGHT && result2 == LEFT){
   std::cout << "coin 10 is lighter" << std::endl;
   coin = 10;
   diff = LIGHTER;
  }else if(result == RIGHT && result2 == RIGHT){
   std::cout << "coin 9 is lighter" << std::endl;
   coin = 9;
   diff = LIGHTER;
  }else if(result == LEFT && result2 == SAME){
   std::cout << "coin 11 is heavier" << std::endl;
   coin = 11;
   diff = HEAVIER;
  }else if(result == RIGHT && result2 == SAME){
   std::cout << "coin 11 is lighter" << std::endl;
```

```
    coin = 11;
    diff = LIGHTER;
   }
  }


 }else{
  l.clear();
  r.clear();
  l.push_back(1);
  l.push_back(2);
  l.push_back(5);
  l.push_back(6);

  r.push_back(7);
  r.push_back(9);
  r.push_back(10);
  r.push_back(11);

  int result2 = w.weigh_action(l, r);

  if(result2 == SAME){
   // counterfeit coin is either 1c, 1d or 2d ie 3, 4, 8
   l.clear();
   r.clear();
   l.push_back(3);
   l.push_back(8);
   r.push_back(9);
   r.push_back(10);
   int result3 = w.weigh_action(l, r);
   if(result3 == SAME){
    // coin 4 is counterfeit
    if(result0 == LEFT){
     std::cout << "coin 4 is heavier" << std::endl;
     coin = 4;
     diff = HEAVIER;
    }else if(result0 == RIGHT){
     std::cout << "coin 4 is lighter" << std::endl;
     coin = 4;
     diff = LIGHTER;
    }
   }else if(result3 == LEFT){
    if(result0 == LEFT){
     std::cout << "coin 3 is heavier" << std::endl;
     coin = 3;
     diff = HEAVIER;
    }else if(result0 == RIGHT){
     std::cout << "coin 8 is heavier" << std::endl;
     coin = 8;
     diff = HEAVIER;
    }
   }else if(result3 == RIGHT){
    if(result0 == LEFT){
     std::cout << "coin 8 is lighter" << std::endl;
     coin = 8;
     diff = LIGHTER;
    }
    else if(result0 == RIGHT){
     std::cout << "coin 3 is lighter" << std::endl;
```

```
      coin = 3;
      diff = LIGHTER;
     }

    }
   }else if(result2 == LEFT){
    // 1a + 1b + 2a + 2b is heavier than 2c + 3 genuine.
    if(result0 == LEFT){
     l.clear();
     r.clear();
     l.push_back(1);
     l.push_back(7);
     r.push_back(9);
     r.push_back(10);
     int result7 = w.weigh_action(l, r);
     if(result7 == SAME){
      std::cout << "coin 2 is heavier" << std::endl;
      coin = 2;
      diff = HEAVIER;
     }else if(result7 == LEFT){
      std::cout << "coin 1 is heavier" << std::endl;
      coin = 1;
      diff = HEAVIER;
     }else if(result7 == RIGHT){
      std::cout << "coin 7 is lighter" << std::endl;
      coin = 7;
      diff = LIGHTER;
     }
    }else if(result0 == RIGHT){
     l.clear();
     r.clear();
     l.push_back(5);
     r.push_back(9);
     int result8 = w.weigh_action(l, r);
     if(result8 == LEFT){
      std::cout << "coin 5 is heavier" << std::endl;
      coin = 5;
      diff = HEAVIER;
     }else{
      std::cout << "coin 6 is heavier" << std::endl;
      coin = 6;
      diff = HEAVIER;
     }
    }

   }else if(result2 == RIGHT){
    // 1a + 1b + 2a + 2b is lighter than 2c + 3 genuine
    // therefore 1a, 1b is lighter or 2c is heavier or 2a, 2b is lighter.
    if(result0 == LEFT){
     // counterfeit coin is 2a, 2b
     l.clear();
     r.clear();
     l.push_back(5);
     r.push_back(6);
     int result5 = w.weigh_action(l, r);
     if(result5 == LEFT){
      std::cout << "coin 6 is lighter" << std::endl;
      coin = 6;
      diff = LIGHTER;
```

```
    }else if(result5 == RIGHT){
     std::cout << "coin 5 is lighter" << std::endl;
     coin = 5;
     diff = LIGHTER;
    }
   }else if(result0 == RIGHT){
    // 1a, 1b are lighter or 2c is heavier.
    l.clear();
    r.clear();
    l.push_back(1);
    l.push_back(7);
    r.push_back(9);
    r.push_back(10);
    int result6 = w.weigh_action(l, r);
    if(result6 == SAME){
     std::cout << "coin 2 is lighter" << std::endl;
     coin = 2;
     diff = LIGHTER;
    }else if(result6 == LEFT){
     std::cout << "coin 7 is heavier" << std::endl;
     coin = 7;
     diff = HEAVIER;

    }else if(result6 == RIGHT){
     std::cout << "coin 1 is lighter" << std::endl;
     coin = 1;
     diff = LIGHTER;
    }
   }
  }


 }


 return std::make_pair<int, int>(coin, diff);
}
```