COMP3121
Assignment 1
A17S1N1

Evangelos Kohilas
z5114986

By submitting this document you are confirming that all the answers are your work and are not take from any other sources unless clearly mentioned.

Question 1:

```
for i in array:
    if i not in set:
        add i to set
    else
        print i
        break
```

Evangelos Kohilas
z5114986

Question 2a:

Split all the people into pairs of two.
for each pair:
    ask if they know the other person in the pair.

Two possible situations arise from each pair:
1) If x knows y, then x can't be a celebrity
2) If x doesn't know y, then y can't be a celebrity

Re-pair all the possible celebrities from the previous pairs into new pairs.
Continue this process until the only possible celebrity can be found.
After this process, n-1 questions have been asked.

Now, for every person in the room, ask if they know that celebrity.
If they all do, then it's still possible for them to be a celebrity.
After this process, n-1 questions have been asked.

Now, ask the possible celebrity if they don't know the other n-1 people.
This process requires n-1 questions, and will result in knowing if the person is a celebrity.

The sum of these steps takes 3*(n-1) questions, resulting in 3*n - 3 questions to find the possible celebrity.

Question 2b:

After the first set of questions have been asked, some of the second set of questions will have been already answered.
This is dependent on the size of n.
For example, if n is of size 2^k (for integer k), then log2(2^k) = k questions would have already been answered, making the total number of questions needed 3n - 3 - log2(n).

However, an example of the worst case would be when the size of n is one higher than a power of 2.
In this case, the total maximum number of questions needed becomes n-1, n-1, n-2 when the celebrity is the odd person out in the tree, and sums to 3n - 4.
This can be seen when n = 9, as ceil(log2(9)) = 4.
In which case, the worst case sum can be overtaken by the ceil(log2(n)), and so, the maximum possible number needed is 3n - ceil(log2(n))

Evangelos Kohilas
z5114986

Question 3a:

```
sort the array with merge sort
let a and b be two pointers at the start and end of the array
    respectively
if x - a > b:
    increment position of a
if x - b > a:
    increment position of b
```

Question 3b:

```
for i in array:
    if i not in set:
        add x-i to set
    else:
        return (x-i, i)
```

Question 4:

The time complexity of Radix sort using Counting sort as a sorting subroutine is O(d*(n+b)) where:

- b is the base for representing numbers.
- d is the maximum number of digits, calculated by log(b,k).
- k is the maximum possible value.

So the time complexity would be O((n+b) * log(b,k))
However, we can find the minimising limiting value of b to be n, and the resulting complexity becomes O(n * log(n,k)).
At this point, we let k = n^9, and we find that the complexity becomes O(n*9), or O(n)

Evangelos Kohilas
z5114986

Question 5:

Sort the numbers using bucket sort with n buckets of range 1/n without sorting the individual buckets.
The size of the input will compensates for the sum of the differences in the numbers, and so a large difference will never appear under one bucket.

e.g. (0.2099...-0.2)*99 ~= 0.99

This can also be stated as:
x1 = bucket / n
x2 = (bucket + 1 - epsilon) / n
(x2 - x1) * (n-1) < 2
Thus, the bound under 1 bucket is (1/n)*(n-1)

Furthermore, a large difference will never appear over multiple buckets
e.g. with 100 buckets, the largest differences between buckets would be;
(0.2199... - 0.2) ~= 0.02
this multiplied over 99 buckets is ~0.02 * 99 ~= 1.98

This can also be stated as:
x1 = bucket / n
x2 = (bucket + 2 - epsilon) / n
(x2 - x1) * (n-1) < 2
Thus, the bound of adjacent buckets becomes (2/n)*(n-1)

Thus, the sum will always be less than 2, as both conditions cannot exist at the same time.

Evangelos Kohilas
z5114986

Question 6:

By doubling the total number of buckets available, the bucket sizes half, (or the range of numbers decreases).
Thus, our previous differences are halved. e.g.
With 100 numbers in 1 bucket, the total range between numbers decreases from 1/n to 1/(2n)
When n = 100, ~0.01 * 99 ~= 0.99
With 50 numbers over 2 buckets, the multiplication value decreases from (n-1) to (n/2-1)
When n = 100, ~0.02 * 49 ~= 0.98

As before, this can be stated as:
x1 = bucket / n
x2 = (bucket + 2 - epsilon) / n
(x2 - x1) * (n-1) < 1

Therefore, the sum will always be less than 1.0001

Alternatively, creating 10000n buckets would extend the range of each bucket to 1/10000n, and so the possible range under one bucket would be,
1/10000n * (n-1) = 1/10000
Additionally, a large difference will never appear between multiple buckets, as the largest possible difference would now be,
2/10000n * (n/2-1) = 1/10000.
Since maximum possible sum while sorted is 1, then 1 + 1/10000 = 1.0001
And therefore, the sum will always be less than 1.0001.

Question 7:

Take your first child, and quicksort the array of shoes using that child's foot as a pivot, moving all shoes smaller than to the left, equal to in the middle, and greater than to the right.

Now, use the shoes that fit the child's foot as a pivot to quicksort the children themselves, moving as above but with children instead.

Repeat this process for every child in the array, noting that you will only need to compare half of the shoes depending on which side of the original child you continue with.

Doing so will result in an O(n*log(n)) algorithm, as for every child whose shoes are different, you are only comparing half of the above array each time.

Evangelos Kohilas
z5114986

Question 8a:

Sort both arrays with merge sort.
Pop an element off both arrays.
    Compare both elements
    Replace smallest element with the next element from its array.
Repeat until either array is empty or a similar element is found.

Question 8b:

```
for every element in first array:
    add said element to a set
for every element in the second array:
    if element is in the set:
        return element
```

Question 9a:

Split the values of the array into pairs of 2, making 2^(n-1) pairs.
Compare each pair, adding the smallest and largest elements into their own list.
This takes 2^(n-1) comparisons.

For both lists, split the values into pairs again.
  For the lower valued list, discard all the larger elements
  For the higher valued list, discard all the smaller elements
Repeat until only 1 element in each list remains, which would be the lowest and highest elements.
This process takes 2^(n+1) - 2 comparisons

Thus, 2^(n-1) + 2^(n+1) - 2 = 3 * 2^(n-1) - 2

Evangelos Kohilas
z5114986

Question 9b:

Split the values of the array, making 2^(n-1) pairs.
For all pairs of that level, find the highest number, creating new 2^(n-2) pairs (in a tree like manner)
Continue until the largest number has been found.
This would take 2^n - 1 comparisons.

Back track from the largest number, keeping a list of all the numbers that the largest number has been compared with.
Now find the max of those numbers iteratively (or again through tree-like pairs).
This would take log2(2^n) - 1 comparisons.

Thus, 2^n - 1 + log2(2^n) - 1 = 2^n + n - 2

Question 10a:

f(n) = n^2
g(n) = (n - 2*log2(n))(n + cos(n))

Limit of f(n)/g(n) as n -> infinity = n^2/(n^2 + n*cos(n) - 2*n*log2(n) - 2*log2(n)*cos(n))

Dividing by n^2 on top and bottom gives:
1/(1 + cos(n)/n - 2*log2(n)/n - (2*log2(n)*cos(n))/n^2)

n grows quicker than log2(n) and cos(n) <= n, so the limit becomes:
1/(1 + 0 - 0 - 0)

Thus, the limit is 1.
Therefore, f(n) = Theta(g(n)).

Evangelos Kohilas
z5114986

Question 10b:

f(n) = log2(n)^2
g(n) = log2(n^log2(n)) + 2*log2(n)

Limit of f(n)/g(n) as n -> infinity = log2(n)^2/(log2(n^log2(n)) + 2*log2(n))

Using log laws gives:
log2(n)^2/(log2(n)*log2(n) + 2*log2(n))
log2(n)^2/(log2(n)^2 + 2*log2(n))

Dividing through by log2(n)^2 gives:
1/(1 + 2/log2(n))

Hence:
1/(1 + 0)

Thus, the limit is 1
Therefore, f(n) = Theta(g(n))

Question 10c:

f(n) = n^100
g(n) = 2^(n/100)

Limit of f(n)/g(n) as n -> infinity = n^100/2^(n/100)

Using log laws gives:
100*log2(n))/(n/100)
10000*log2(n))/n

As n grows quicker than log2(n), the limit is equal to 0
Hence, f(n) = O(g(n)).

Evangelos Kohilas
z5114986

Question 10d:

f(n) = n^100
g(n) = 2^n^(1/100)

Limit of f(n)/g(n) as n -> infinity = n^100/2^n^(1/100)

Using log laws gives:
100*log2(n))/(n^(1/100))

Let x = n^(1/100), n = x^100
10000*log2(x))/x

As x grows quicker than log2(x), and x grows quicker than n, the limit is equal to 0.
Hence, f(n) = O(g(n))

Question 10e:

f(n) = sqrt(n)
g(n) = 2^sqrt(log2(n))

Limit of f(n)/g(n) as n -> infinity = sqrt(n)/2^sqrt(log2(n))

Using log laws gives:
log2(sqrt(n))/sqrt(log2(n))

Squaring through top and bottom:
log2(sqrt(n))^2/log2(n)

Using log laws:
log2(n)^2/(4*log2(n))

Dividing through top and bottom by log2(n):
log2(n)/4

As log2(n) grows to infinity, the limit grows to infinity.
Hence, f(n) = Omega(g(n)).

Evangelos Kohilas
z5114986

Question 10f:

f(n) = n^100
g(n) = 2^log2(n)^2

Limit of f(n)/g(n) as n -> infinity = n^100/2^log2(n)^2

Using log laws gives:
100*log2(n)/log2(n)^2

Dividing top and bottom by log2(n) gives:
100/log2(n)

As log2(n) grows, the limit decreases to 0.
Hence, f(n) = O(g(n)).

Question 10g:

f(n) = n^1.001
g(n) = n*log2(n)

Limit of f(n)/g(n) as n -> infinity = n^1.001/(n*log2(n))

Dividing through by n gives:
n^(1/1000)/log2(n)

Using log laws gives:
1000*log2(n)/n

As n grows quicker than log2(n), the limit is 0
Hence, f(n) = O(g(n))

Evangelos Kohilas
z5114986

Question 10h:

f(n) = n^((1+sin((pi*n)/2))/2)
g(n) = sqrt(n)
Limit of f(n)/g(n) as n -> infinity = n^((1+sin((pi*n)/2))/2)/sqrt(n)

As sin exists in f(n), f(n) will continuously oscillate as n approaches infinity.
g(n) will continue to grow to infinity.
Thus, the limit does not exist.
Therefore, f(n) and g(n) cannot be bounded with a multiple of each other, so no asymptotic notation exists.

Evangelos Kohilas
z5114986

Puzzle 1a:

```
let @ be ON and O be OFF
Let game start as @ @
if light 1 is chosen, no ON lights remain
    0) @ @ -> O O
if light 2 is chosen, light 1 remains ON, and therefore 1 move
remains
    1) @ O -> O @ -> O O
if the game starts as O @, only light 1 remains
    2) O @ -> O O
therefore all combinations of 2 lights are shown to end

Let the game start as @ @ @
if light 1 is chosen, no ON lights remain
    @ @ @ -> O O O
if 2 is chosen, only 1 remains, which results in case above
    3) @ O O -> 4) O @ @ -> (0)
if 3 is chosen:
    @ @ O
    if 1 is chosen, fall back to the case above
        O @ @ -> (4)
    if 2 is chosen:
        @ O @
        if 1 is chosen, no options remain
            O @ O -> O O @ -> (2)
        if 3 is chosen:
            @ O O -> (3)
```

therefore, all combinations of lights can be represented as a combination of 2 and 3 lights

Puzzle 1b:

if there is an odd amount of off lights to the right of the left most ON light, alice will lose?

Puzzle 3:

```
traverse the perimeter
while block not found:
    follow first outgoing road
    turn at the first road, noting direction
        continue following direction
    until you have reached the perimeter or you cannot follow the
same direction

    if perimeter reached:
        end
    else:
        return to perimeter
```