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
```

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 $3n-3$ 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 8, then $\log_2 8 = 3$ questions would have already been answered, making the total number of questions needed $3n-3-\log_2 n$ however, an example of the worst case would be when the size of n is one higher than a power of 2. in which 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 this sums to $3n-4$, which is equivalent to $\log_2 9 = 4$ and so, generalised, the maximum possible number needed is $3n - \log_2 n$

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, the resulting complexity becomes $O(n \log_n k)$ at this point, we let $k = n^9$, and we find that the complexity becomes $O(9n)$ or rather, $O(n)$

Question 5:

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

i.e. $(0.20\dot{9} - 0.2) * 99 \approx 0.99$ $x1 = bucket/n$ $x2 = (bucket + 1 - \epsilon)/n$ $(x2 - x1)(n-1) < 2$ and so, the bound under 1 bucket is $(1/n)(n-1)$

additionally, a large difference will never appear over multiple buckets i.e. with 100 buckets, the largest differences between buckets would be $0.21\dot{9} - 0.2 \approx 0.02$ this multiplied over 99 buckets is $0.02 * 99 \approx 1.98$

$x1 = bucket/n$ $x2 = (bucket + 2 - \epsilon)/n$ $(x2 - x1)(n-1) < 2$ and so, the bound of adjacent buckets becomes $(2/n)(n-1)$

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

Question 6:

by doubling the total number of buckets available, the bucket sizes half, (or the range of numbers decreases) and thus, our previous difference is halved with 100 numbers in 1 bucket, the total range between numbers decreases from $1/n$ to $1/2n$ e.g. where $n = 100,\ 0.01 * 99 \approx 0.99$ with 50 numbers over 2 buckets, the multiplication value decreases from $n - 1$ to $n/2 - 1$ e.g. where $n = 100,\ 0.02 * 49 \approx 0.98$

as before: $x1 = bucket/n\ x2 = (bucket + 2 - epsilon)/n\ (x2 - x1)(n - 1) < 1$

thus, 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 $1 + 1/10000 = 1.0001$

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

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, $2^{n-1} + 2^{n+1} - 2 = 3 * 2^{n-1} - 2$