Group Members:

- Norah Albeshri
- Salman Ashraf
- Charles Charles-Fredrick
- Olive Kanengoni
- Yousef Morfeq
- Alexander Tate

Computational Thinking

Coursework 2 - Kidney Donations

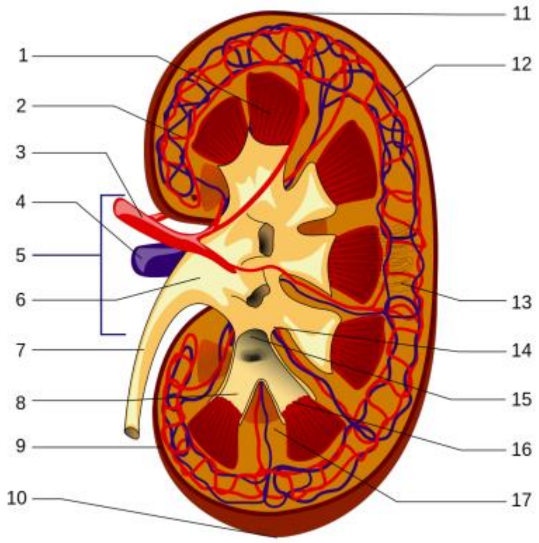


Figure 1: A Kidney

Task 1

Assumptions:

- "n" is the number of pairs in the input set
- "`outList`" is an initially empty list that contains all the possible matches of pairs.
- "`For x = 1 to n`" loops from 1 to n while keeping track of the current iteration in var x.
- "`Matches`" refers to the check carried out by the black box to determine if a donor and recipient are compatible.
- `pair(x)` returns the pair of the donor-recipient pair at location x in the input set.
- `donor(x)` returns the donor of the donor-recipient pair at location x in the input set.
- `recipient(x)` returns the recipient of the donor-recipient pair at location x in the input set.
- `append(x)` means to convert x to string and add as the next element to the end of the list.
- "`For x in list`" loops through the list whilst storing the current instance in variable x.

Pseudocode:

```
outList = []
For i = 1  to n:
      For j = i + 1 to n:
            If donor(i) matches recipient(j) and donor(j) matches recipient(i):
                  Append(pair(i) , pair(j)) to outList
For output in outList:
      Print output
```

Description:

This program loops through all the possible matches of two pairs using a nested for loop and uses the black box to find a list of all possible matches of donor-recipient pairs.

This outputs only one matching per pair of donor-recipient pairs, for example pair(4) and pair(6) may be an output of this whilst pair(6) and pair(4) is not as these are the telling us the same information so it would serve no purpose to also output the second matching, and it is a waste of storage space and processing power for the computer to work these out.

Task 2

Assumptions:

- "n" is the number of pairs in the input set
- "`outList`" is an initially empty list that contains all the possible matches of pairs.
- "`For x = 1 to n`" loops from 1 to n while keeping track of the current iteration in var x.
- "`Matches`" refers to the check carried out by the black box to determine if a donor and recipient are compatible.
- `pair(x)` returns the pair of the donor-recipient pair at location x in the input set.
- `donor(x)` returns the donor of the donor-recipient pair at location x in the input set.
- `recipient(x)` returns the recipient of the donor-recipient pair at location x in the input set.
- `append(x)` means to convert x to string and add as the next element to the end of the list.
- "`For x in list`" loops through the list whilst storing the current instance in variable x.

Pseudocode:

```
outList = []
For i = 1  to n:
      For j = i + 1 to n:
            For k = j + 1 to n:
                  If donor(i) matches recipient(j) and donor(j) matches
                  recipient(k) and donor(k) matches recipient(i):
                        Append(pair(i) , pair(j) , pair(k)) to outList
                  If donor(i) matches recipient(k) and donor(k) matches
                  recipient(j) and donor(j) matches recipient(i):
                        Append(pair(i) , pair(k) , pair(j)) to outList
For output in outList:
      Print output
```

Description:

This program loops through all the unique possible matchings of three pairs. It takes three unique pairs and uses the black box to check in both a clockwise and anticlockwise direction if these three pairs are matching and if so, add it to the output list.

This outputs only one matching per set of three, for example pair(4), pair(6), and pair(7) may be an output of this whilst pair(6), pair(7), pair(4) and pair(7), pair(4), pair(6) are not as these are the telling us the same information so it would serve no purpose to also output these matchings, and it is a waste of storage space and processing power for the computer to work these out.

Task 3

For task 1, to identify all donor-recipient pair matches, there is a nested FOR loop. The first loop executes **n** times, and the second **<n** times, so the run-time complexity is ***n^2***, meaning that it never takes more time than **n^2**.

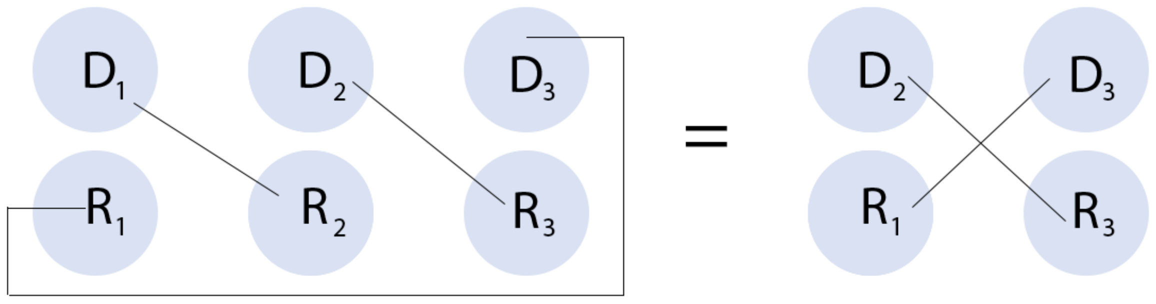
For task 2, to identify all three-way pooled donor-recipient pair matches, there is a triple nested loop. Each of the loops do not always execute **n** times, but the number of times that they do execute still increases with **n**, which means that the complexity is ***n^3***.

Generally, a kidney donation is performed between close relatives since they are more likely to have the same blood and tissue type. We can use these factors to make the program run faster by organising the pair into multiple groups based on kidney matching criterias (e.g. Blood group and tissue type). Run the program to find matches within each group. With some changes to the code, this could decrease the number of comparisons to be made between the pairs and will increase the efficiency and speed of the algorithm.

Pools with *n* donor-recipient pairs can be broken down into two problems:

1) R_n has to match with D_{n-1}, R_n matches with D_{n-2}, etc.
2) D_n matches with R_1

The recursive looping of part (1) eventually simplifies any size pool down into just a two-pair exchange which can be solved as normal, as shown in the image below.



To maximise overall matches, we should have an output list with all possible matches, e.g. all possible doubles, triples, and quadruples in one list.

Then, the donor-recipient pairs with the fewest matches in the output list should take priority, meaning if a pair only has one potential match, this match should be done first and both pairs removed from the output list.

Repeat for the pairs with the lowest number of possible matchings until there are no possible matches left.