COMP3027 Assignment 1: 500468777

Problem 1

Solution 1:

1. The algorithm is incorrect as the numbers in A3 and A4 must come from either A1 or A2. Though the language used makes exactly which one unclear, the following is true:
$$(A1 \cup A2) \cap (A3 \cup A4) \neq \phi$$
Thus there are numbers shared between at least two arrays meaning the solution cannot be correct
2. As O(nlogn) > O(n) the algorithm does not meet the time requirement and is incorrect
3. ChatGPT claimed each step took O(n) time however, recursively sorting arrays takes O(nlogn) time
4. The concluded time of O(nlogn) is correct
5. The algorithm is unnecessarily complicated as once the two halves of the array are sorted after step two, the objective is fulfilled as
$$w < x < y < z,$$
$$\forall\, w \in A1[0:n], \forall\, x \in A1[n:2n], \qquad \forall y \in A2[0:n], \forall\, z \in A2[n:2n]$$

Solution 2:

1. The algorithm produces a correct output as sorting ensures items are increasing as they are distinct, meaning all items in an array will be smaller than in an array with a greater index as they contain items with a greater index in the sorted array but the time analysis is incorrect and the algorithm is not efficient enough to meet the requiements.
2. The claimed time of O(n) meets the time requirements but is incorrect and the given algorithm does not meet the time requirement but would still generate a correct solution.
3. Quicksorting takes O(nlogn) time as correctly stated. Thus, quicksorting an array of 4n elements would take O(4nlog4n) = O(n(logn + log4)) = O(nlogn) time.
4. The time analysis is incorrect as the time it takes to quicksort an array of length 4n is O(nlogn) time
5. The algorithm does not contain irrelevant information as it consists of just a sort and a partition.

Solution 3:

1. The algorithm produces correct output, correctly justified in the first paragraph of the solution but does not meet the time requirements.
2. The algorithm takes O(nlogn) time and thus does not meet the time requirements
3. O(nlogn) dominates O(n) in the runtime not the other way around and hence the algorithm takes O(nlogn) time
4. The algorithm takes O(nlogn) time and hence the time analysis of O(n) is incorrect

5. There are no irrelevant details in the solution as the solution consists of only sorting and partitioning the original array. The second paragraph could be discarded but it is not irrelevant.

Solution 4:

1. The solution is correct but does not meet the time requirements
2. The time complexity (without the note) is correct but O(nlogn) does not meet the time requirement
3. The note in the solution shows a basic misunderstanding of big-Oh notation as while the difference between O(nlogn) and O(nlogn + n) is not significant for large n, the O(nlogn) term is the one that dominates the other not O(n).
4. The time analysis (except for the note at the end) is correct
5. The note at the end is irrelevant and incorrect. If the note was discarded then the solution would be correct.

Solution 5:

1. The solution is correct as sorting the array and then partitioning based on the quartiles will result in 4 arrays where the elements in each are greater than the elements in any array before.
2. The function is effectively describing a merge sort which takes O(nlogn) time and thus doesn't meet the time requirements
3. At each step, the median is found and then the array is split based on elements smaller and larger than the median. This takes O(n) time. Thus the recurrence relation is $T(n) = O(n) + 2T(n/2)$ as the process is repeated for arrays half as large. It is clear the solution does not consider the complexity of the subproblems at each step.
4. The recurrence relation above solves to O(nlogn) and thus the claimed O(n) is incorrect.
5. The paragraph describing the partitioning of the arrays is ambiguous as it is not specified what the "next smallest" and "next largest" are in relation to. It would be simpler to state arrays of size n are formed. Further, the algorithm splits the arrays into leaves which are then combined but it can actually stop at leaves with arrays of size n, or at a depth of 3 as arrays A1, A2, A3 and A4 do not need to be sorted themselves.

Problem 2

a) The greedy algorithm will work as follows. First sort both arrays in ascending order using merge sort creating arrays W* and B*. Then assign the broom at B*[i] to the wizard at W*[i] for all i = 1, 2, …, n. ie assign the unassigned broom with the least fuel to the unassigned wizard who needs to travel the shortest distance.

b) Assume an optimal solution S exists. If S = G then we are done.
If $S \neq G$ then there exists brooms $b_j, b_k \in B$ and wizards $w_j, w_k \in W$ such that $w_j < w_k$ and $b_j < b_k$ and $(w_j, b_k), (w_k , b_j) \in S$ and $(w_j, b_j), (w_k, b_k) \in G$. Define S* = S but with the pair above inverted to match G.

Now we are looking to prove:

$$0 \leq \ mismatch(S) \ - \ mismatch(S*)$$
$$\leftrightarrow 0 \leq \frac{1}{n}\sum differece\ in\ pairs\ in\ S - \frac{1}{n}\sum difference\ in\ pairs\ in\ S*$$
$$\leftrightarrow 0 < |w_j - b_k| + |w_k - b_j| - (|w_j - b_j| + |w_k - b_k|) = \Delta$$

Define:

$d = w_j - b_k$

$w_k = w_j + x,\ x > 0$

$b_j = b_k - y,\ y > 0$

Then:

$$\Delta = |w_j - b_k| + |w_j + x - b_k + y| - (|w_j - b_k + y| + |w_j + x - b_k|)$$
$$= |d| + |d + x + y| - |d + y| - |d + x|$$
$$= |d + x + y| - |d + x| - (|d + y| - |d|)$$

Consider the function f: R → R, f(z) = |z + k| - |z| for some constant k > 0

If z ≥ 0: f(z) = z + k – z = k

If z ≤ -k: f(z) = (k + c) – k – (k + c) (for some c > 0) = -k

If -k < z < 0: f(z) = 2z - k

Then $\Delta = f(d + x) - f(d)\ for\ k = y$

If d ≥ 0:

⇨ d + x > 0 (as x > 0)

⇨ Δ = y – y = 0

If d + x ≤ -y:

⇨ d < -y (as x > 0)

⇨ Δ = -y –(-y) = 0

If -y < x < d + x < 0:

⇨ Δ = 2(d + x) + y – (2d + y) = 2x > 0 as x > 0

If -y < d + x < 0 and d < -y:

⇨ Δ = 2(d + x) + y – (-y) = 2(d + x) + 2y > 0 (-y < d + x < 0 => |y| > |d + x|)

If d + x > 0 and -y < d < 0:

⇨ Δ = y – (2d – y) = 2y – 2d > 0 as (-y<d<0 => |y| > |d|)

If d + x > 0 and d < -y:

⇨ Δ = y – (-y) = 2y > 0

Thus Δ ≥ 0 and therefore the difference in the mismatch between S and S* is greater than 0. Therefore, an inversion towards the greedy solution in an optimal solution is at least as good or better. Hence if inversions are performed in S until G is achieved then G is at least as good or better than S. Hence G is an optimal solution.

c) The merge sort in the solution takes O(nlogn) time and assigning wizards to brooms in the sorted array can be done in linear time. Thus, the overall time complexity is O(nlogn).

d) Assume a solution S exists that minimizes the mismax of the solution. Then $mismax_S \leq mismax_G$. If S = G then G minimizes the mismax.
If S ≠ G then there exists brooms $b_j, b_k \in B$ and wizards $w_j, w_k \in W$ such that $w_j < w_k$ and $b_j < b_k$ and $(w_j, b_k), (w_k, b_j) \in S$ and $(w_j, b_j), (w_k, b_k) \in G$. Define S* = S but with the pair above inverted to match G.

We are now looking to prove:

$$mismax_s \geq mismax_G$$

$$\leftrightarrow \max(|w_j - b_k|, |w_k - b_j|) \geq max(|w_j - b_j|, |w_k - b_k|)$$

For $|W_j - b_j|$:

If $w_j > b_j$:

$| w_j - b_j | = w_j - b_j < w_k - b_j$

If $b_j > w_j$:

$| w_j - b_j | = b_j - w_j < b_k - w_j$

If $w_j = b_j$:

$| w_j - b_j | = 0 \leq | x |$ for all x

$\Rightarrow | w_j - b_j | \leq \max(|w_j - b_k|, |w_k - b_j|)$

Thus $| w_j - b_j | \leq \max(|w_j - b_k|, |w_k - b_j|)$

For $|w_k - b_k|$:

If $w_j \geq b_k$:

$|w_j - b_k| = w_j - b_k < w_k - b_k < w_k - b_j$

If $b_k > w_j$:

If $b_k \geq w_k$:

$| w_k - b_k | < b_k - w_k < b_k - w_j$

If $w_k > b_k$:

$|w_k - b_j| - |w_k - b_k|$

$= w_k - b_j - (w_k - b_k)$

$= b_k - b_j > 0$

Thus $|w_k - b_j| > |w_k - b_k|$

Thus $|w_k - b_k| \leq \max(|w_j - b_k|, |w_k - b_j|)$

Thus $| w_j - b_j | \leq \max(|w_j - b_k|, |w_k - b_j|)$
and $|w_k - b_k| \leq \max(|w_j - b_k|, |w_k - b_j|)$

Therefore

$$\max(|w_j - b_k|, |w_k - b_j|) \geq max(|w_j - b_j|, |w_k - b_k|)$$

Thus $mismax_{S^*} \leq mismax_S$ meaning an inversion in S towards a greedy solution will only make the mismax smaller or keep it the same. Hence, if inversions are

applied until G is obtained then $\text{mismax}_S \geq \text{mismax}_G$. Thus, the greedy solution does minimize the maximum mismatch in wizards and brooms.

Problem 3

a)

n	|E|
2	0
3	8
4	12

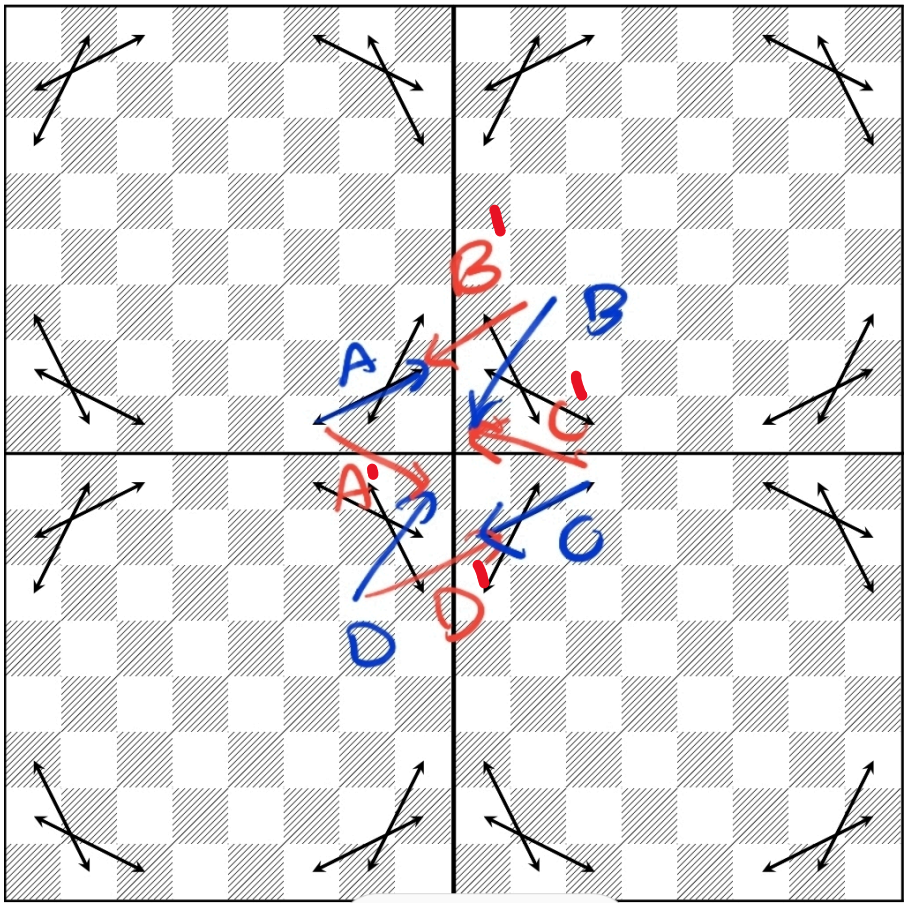
b) 5

c) $|v| = n^2$, $|E| = 4(n-1)(n-2)$
Thus $O(|V| + |E|) = O(n^2 + 4n^2 - 32n + 16)$
$= O(n^2)$

d)

Given the friendly knights tours for the 4 n x n sub-board we know the moves in blue (ignoring direction as the tour is closed) below will be included in each (as A and C are required for it to be friendly and B and D must be included as there are only two moves at each corner. Replacing the moves, A, B, C and D with the moves in A', B', C', D' will form a closed, friendly tour around the (2n) x (2n) board.

As this only makes 4 changes to each subtour, this can be done in constant time. Proof of correctness in part f.



e) We are given a graph of size $n = 2^k$.

If k = 3, return the closed friendly pony tour on that board
If k > 3, set W, X, Y, Z to be the tours returned when recursed on the four quadrants of the board (size $n = 2^{k-1}$).
We know the moves A, B, C and D will exist in W, X, Y and Z respectively.
Replace moves A, B, C and D with A', B', C' and D' combining the 4 separate tours into one closed, friendly tour.
Return that tour.

(A, B, C, D, A', B', C', D' in part d)

f) Proof by induction:
Base case: $n = 2^3$:

It is given that a closed friendly tour exists for a board of size n = 8.

Assume for $n = 2^k$ that the algorithm correctly returns a closed, friendly pony tour

When $n = 2^{k+1}$:

Each friendly tour is a closed loop and thus can start and end at any point on the board. Further, each subtour can be completed in either direction as they are closed.

The first change is the move A is changed to send the pony to the bottom left sub-board. As there is a closed tour in that sub-board, we know both legal moves (that keep the pony on that sub-board) from the top right corner must be made. As the order does not matter, we chose to send the pony in the direction that ensures that subtour ends with the move D. However, D' now sends the pony to the bottom right sub-graph. The bottom right subtour begins there and we define the direction to finish with the move C (which must occur as the tour is friendly). However, that move is replaced by C' which sends the night to the bottom left corner of the top right sub-board. Again, we know a closed tour exists here and we chose to send the night in the direction that would end in B, which is replaced with B'. As B' sends the pony to where A would have, the tour in the top left can continue. As each subtour is friendly, we know the tour across the 2n x 2n board will be too.

Further we know each square is visited once as each new move sends the night to the position a replaced move would have. Therefore, as all the other squares are known to be visited, the pony must visit all $(2n)^2$ squares.

Thus, if a closed, friendly pony tour exists for $n=2^k$ so too does one for $n = 2^{k+1}$. As a closed, friendly pony tour exists for $n = 2^3 = 8$, there exists a closed, friendly pony tour for all $n=2^k$, $k \geq 3$.

The algorithm has the following recurrence relation:
$T(n) = O(1) + 4T(n/2)$
Which unravels to:

$O(1 + 4 + 16 + \ldots + 4^{\log n/\log 2})$
$= O((4^{\log n/\log 2 + 1} - 1) / (4 - 1))$
$= O(4^{\log n/\log 2 + 1})$
$= O(4*4^{\log n/\log 2})$
$= O(2^{2 * \log n/\log 2})$
$= O(2^{\log(n^2) / \log 2})$
$= O(n^2)$
⇨ Algorithm runs in $O(n^2)$ time

g) The algorithm takes a pony graph as an input which has size $O(|V| + |E|) = O(n^2)$ Hence, although the algorithm is quadratic in n (the edge size of the graph), it is linear in the input size as both are $O(n^2)$