COMP3027 Algorithm Design

Assignment 1

SID: 490210055

1 ChatGPT Algorithms Analysis

① the solution is correct

② the algorithm is correct, but does not meet the $O(n)$ time requirement

③ the solution shows basic misunderstanding of big-Oh notation

④ the time analysis is incorrect

⑤ the algorithm description is filled with irrelevant details

1.1 Merge Sort Variant

④:
The algorithm is not correct and confusing since A1 or A2 have two different assignments, which could be either divided halves or parts of the final result. As a direct consequence, the time analysis is far from being correct.
③:
The statement of algorithm complexity contradicts its claimed overall time complexity.

1.2 Quick Sort then Divide

② correctness:
the algorithm sorts A ascending before it splits A into A_1, A_2, A_3, A_4 of size n. Therefore, $\forall i, j \in \{1,2,3,4\}, i < j : max(A_i) < max(A_j)$ which implies that $\forall i, j \in \{1,2,3,4\}, i < j : \forall k, l \in [1, n] : A_i[k] < A_j[l]$.
②④ :
$O(n \log n)$ is the expected running time for a QuickSort. The worse time of a QuickSort for n items is of $O(n^2)$ especially if A has been already sorted. Therefore overall running time is $O(n^2) + O(n) \in O(n^2)$.
③:
$O(n \log n) \notin O(n)$ as $O(n \log n)$ dominates over $O(n)$ asymptotically.

1.3 Sort and Divide 1

②③④:
The reasons stay the same as those for subsection 1.2 if the sorting algorithm is implemented as a QuickSort. Even if using MergeSort, the algorithm still has a bounded performance of $O(n \log n) \notin O(n)$.
⑤:
To verify that A is an ordered array, at least $4n$ steps are required. However, this is for a best case. It's not practical to check whether A is ordered if no other heuristic information is provided since time complexity needs to cover worst scenarios. In other words, those $4n$ steps could be a waste of computational resource.

1.4 Sort and Divide 2

③:
When n is large enough, the difference between $O(n)$ and $O(n \log n)$ is, in fact, significant.
②:
Similar to analysis in 1.2 and 1.3.

1.5 Find the median then Divide and Conquer

④:
The time complexity analysis only claims running time for each step but fails to count overall performance as a sum of subtask running time.
⑤:
"If a node has no children, we add the value stored in the node to one of the four arrays.". The procedure is neither explicit nor definitive. Instructions of algorithms should be clear and decidable.

2 Broom Replacement

2.1 algorithm

The main idea is to adopt a greedy algorithm which MergeSorts W and B arrays firstly. Then the algorithm maps elements in W to elements in B pairwise from the beginning of sorted arrays to the end.

1. Reconstruct the data structure. That is to say, W and B will store a tuple of its index and the corresponding value of the element in that index position as (i, d_i). Notationally, $W[i] = d_i$ becomes $W[i] = (i, d_i)$, where i maps to a wizard, d_i is the distance for the wizard to get home. Similarly, $B[i] = (i, f_i)$ instead, where i maps to a boom and the distance its fuel covering f_i.

2. MergeSort W and B separately by d_i and f_i. After the sorting process, we have $W[i] = (w_{i1}, w_{i2}), B[i] = (f_{i1}, f_{i2})$. Arrays have such a property: $\forall i, j \in [1, n] : i < j \Rightarrow w_{i2} \leq w_{j2} \wedge f_{i2} \leq f_{j2}$

3. Pair the first values of the tuples in two arrays to create a mapping function π, mathematically, $\forall i \in [1, n] : \pi[f_{i1}] = w_{i1}$.

2.2 correctness

Suppose we have another mapping function Π different from π such that mismatch(Π) < mismatch(π). Since we have reconstructed the data structure, we have mismatch(π) = $\frac{1}{n}\sum_{i=1}^{n} |w_{i2} - f_{i2}|\big|_{\pi[f_{i1}]=w_{i1}}$.For simplicity, we assume $\exists i, j \in [1, n] : \Pi[f_{j1}] = w_{i1} \wedge \Pi[f_{i1}] = w_{j1}$. Without loss of generosity, we could further strict that $i < j$. The rest of the mapping table of Π and that of π are exactly the same.
Now we derive from the assumption that, mismatch(Π) < mismatch(π) $\Rightarrow$ mismatch(Π) $-$ mismatch(π) $< 0 \Rightarrow (|w_{i2} - f_{j2}| + |w_{j2} - f_{i2}|) - (|w_{i2} - f_{i2}| + |w_{j2} - f_{j2}|) < 0$ **(1)**
However, since the MergeSort procedure(Step 2 of 2.1) has sorted the second values of the tuples, we know that $w_{i2} < w_{j2} \wedge f_{i2} < f_{j2}$ because $i < j$ (For simplicity of the proof, equivalent w_{x2}s and f_{x2}s are omitted). Define a, b, c, d as $a := w_{i2}$, $b := w_{j2}$, $c := f_{i2}$, $d := f_{j2}$. Now, we can generate 6 cases, which are listed following, to remove the remaining norm operators on the LHS of the inequality **(1)**.

- $a < b < c < d$
- $a < c < b < d$
- $a < c < d < b$
- $c < a < b < d$
- $c < d < a < b$
- $c < a < d < b$

For each case, it is trivial to know that $(|w_{i2} - f_{j2}| + |w_{j2} - f_{i2}|) - (|w_{i2} - f_{i2}| + |w_{j2} - f_{j2}|) > 0$, which leads to a contradiction against **(1)**. Thus we have shown that there's no such function Π. Finally, we conclude that mismatch(π) is indeed the minimum value where π is defined in algorithm 2.1.

2.3 time complexity analysis

Step 1: $O(n)$ because of iterating through arrays
Step 2: $O(n \log n)$ because of MergeSort
Step 3: $O(n)$ because of iterating through again.
Overall: $O(n) + O(n \log n) + O(n) \in O(n \log n)$ as each step is independent.
The algorithm has a $O(n \log n)$ running time which satisfies the requirement to be of $O(n^2)$.

2.4 maximum mismatch

We use the same technique in 2.2 to prove that $mismax(\pi)$ is optimal.
Assume that there exists a Π such that $mismax(\Pi) < mismax(\pi)$ and $\Pi \neq \pi$.
$mismax(\pi) := |w_{k2} - f_{k2}|, k \in [1, n]$ and $mismax(\Pi) := max(|w_{l2} - f_{k2}|, |f_{l2} - w_{k2}|), l \in [1, n], k \neq l$.
By substituting $mismax(\Pi), mismax(\pi)$ we know that $mismax(\Pi) < mismax(\pi) \Rightarrow max(|w_{l2} - f_{k2}|, |f_{l2} - w_{k2}|) < |w_{k2} - f_{k2}|$ $\textcircled{\textbf{2}}$
$a := w_{l2}, b := w_{k2}, c := f_{l2}, d := f_{k2}$ then $\textcircled{\textbf{2}} \Rightarrow max(|a - d|, |b - c|) < |b - d|$ $\textcircled{\textbf{3}}$

- $a < c < d < b$
  either $\Rightarrow |c-b| > |b-d| \wedge |a-d| > |a-c| \Rightarrow max(|a-d|, |c-b|) > max(|b-d|, |a-c|) = |b - d|$, which contradicts with $\textcircled{\textbf{3}}$
- $c < a < b < d$
  The analysis is similar with the previous case.
- $a < b < c < d$ or $a < c < b < d$
  $|a - d| > |b - d| \Rightarrow max(|a - d|, |c - b|) > |b - d|$, which contradicts with $\textcircled{\textbf{3}}$
- $c < a < d < b$ or $c < d < a < b$
  $|b - c| > |b - d| \Rightarrow max(|a - d|, |c - b|) > |b - d|$, which contradicts with $\textcircled{\textbf{3}}$
- the rest cases of a, b, c, d permutations are invalid since either $c < d \wedge a < b$ or $d < c \wedge b < a$ must hold.

Combining all possible situations listed above, we know that there's no such Π fulfilling $mismax(\Pi) < mismax(\pi)$. Thus, $mismax(\pi)$ is indeed the minimum of $mismax(x)$, where x is a mapping from a wizard to a boom.

3 Pony Tour

3.1 number of edges

$|E| = 0$ for $G(2)$
$|E| = 8$ for $G(3)$
$|E| = 24$ for $G(4)$

3.2 closed form of number of edges

$4(n - 1)(n - 2)$

3.3 $|E| + |V|$ asymtote

$|E| + |V| = n^2 + 4(n - 1)(n - 2) \in O(n^2)$

3.4 expand *Friendly* Pony Tour strategy

Given is a *friendly* pony tour of dimension length n, $PT(n)$.
Divide a $PT(2n)$ into 4 subsets which we call $PT(TL), PT(TR), PT(BL), PT(BR)$, (T for top, L for left, B for bottom, R for right). The subsets are all of $n \times n$ and *friendly.* Now we show that $PT(2n)$ is *friendly* by giving a concrete sequence of moves which covers all 8 moves at the corners of $PT(2n)$.
Starting from the top left corner of $PT(TL)$, we could assume that the last move of $PT(TL)$ is $(1, n-2) \rightarrow (2, n)$.
Then we travel to $PT(BL)$ by moving $(2, n) \rightarrow (1, n+2)$. Since $(1, n+2)$ is a square on an end of one of the 8 moves, the square $(n, 2n-1)$ could be reached via a *friendly* tour. The following internal moves of $PT(n)$s are omitted. We only explicitly focus on travels between different $PT(n)$s and show that travels are complete(cover all corners of $PT(2n)$) and are not repeated(each square is visited less than twice). Here is a summary including previous two transitions.

1. $PT(TL) \rightarrow PT(BL)$
   $(1, 3) \rightarrow$ ...$PT(TL)$ internal moves... $\rightarrow (2, n) \rightarrow (1, n+2)$
2. $PT(BL) \rightarrow PT(BR)$
   $(1, n+2) \rightarrow$ ...$PT(BL)$ internal moves... $\rightarrow (n, 2n-1) \rightarrow (n+2, 2n)$
3. $PT(BR) \rightarrow PT(TR)$
   $(n+2, 2n) \rightarrow$ ...$PT(BR)$ internal moves... $\rightarrow (2n-1, n+1) \rightarrow (2n, n-1)$
4. $PT(TR)$
   $(2n, n-1) \rightarrow PT(TR)$ internal moves

Therefore, a concrete iteration of moves covers all 8 moves at the 4 corners of $PT(2n)$, in other words, $PT(2n)$ is *friendly.* In addition, the iteration has exactly 8×4 moves. (8 moves for each $PT(n)$, 4 $PT(n)$s in total) It is of a constant time $O(1)$.

3.5 D&C for Pony Tour

- divide
  separate the graph of size $(n \times n)$ into 4 even sub-graphs of size $(\frac{n}{2} \times \frac{n}{2})$ in a way of the previous section(3.4).
- delegate
  recursively find a *friendly* pony tour of each of the 4 sub-graphs
- combine
  combine 4 *friendly* pony tours of subgraphs into a single *friendly* pony tour.

3.6 Analysis

- correctness
  "divide" and "combine" steps are always viable as $n = 2^k, k \geq 3, k \in \mathbb{Z}$.
  "delegate" step is correct as we have proved that 4 *friendly* pony tours of graphs of size n can be combined in to a *friendly* pony tour of a graph of $2n \times 2n$ and we are given an assumption that *there exists a Friendly Pony Tour for chessboards of size n = 8.*

- time complexity
  $T(n) = 4T(n/2) + O(1)$ since "divide" and "combine" part consume a constant time. By the master theorem, $n^{\log_b^a} = n^{\log_2^4} = n^2$. $f(n)$ is $O(n^{1-\epsilon})$ for $\epsilon = 1$. Thus $T(n) \in \Theta(n)$.

3.7 performance

As shown in 3.6, $T(n) \in \Theta(n)$, which is a linear time if we assume the input is n(size of a dimension of the graph).