Comp3121 assignment 2 Question 2
Chen En (z5335039)

Question 2

[30 marks] UNSW is competing in a programming varsity competition, and $2k$ students have registered. This competition requires students to compete in pairs, and so UNSW has run a mock competition to help determine the pairings. The results for each of the $2k$ students have been stored in an array L, and each result is a non negative integer. The score of each pair is the product of the marks of the two students, and the final score of each university is the sum of all pairs' scores.

For example, if the marks of the students are $L = [1, 4, 5, 3]$, then the pairs $(1, 3)$ and $(4, 5)$ give the largest total score of $1 \cdot 3 + 4 \cdot 5 = 3 + 20 = 23$.

2.1 **[10 marks]** Suppose the marks of 4 students are a, b, c, d where $a \leq b \leq c \leq d$. Show that $ab + cd \geq ac + bd \geq ad + bc$. You can assume that all marks are non-negative integers.

This subquestion should be solved independently of the overall question's context.

2.1
To prove $ab + cd \geq ac + bd \geq ad + bc$, we split it to two inequality equations $ab + cd \geq ac + bd$ and $ac + bd \geq ad + bc$ then prove them individually.

Proof.
Prove $ab + cd \geq ac + bd$:
Given by the question we know that $a \leq b \leq c \leq d$, which we can assume that $c - a \geq b - a$ and $d - b \geq d - c$. We combine those two inequality equations by product two sides that gives us

$$(c - a)(d - b) \geq (b - a)(d - c)$$
$$\Longrightarrow cd - bc - ad + ab \geq bd - bc - ad + ac$$
$$\Longrightarrow cd + ab \geq bd + ac$$

Therefore, we prove that

$$cd + ab \geq bd + ac$$

Prove $ac + bd \geq ad + bc$:
Use the similar approach as the proving above, we assume that $b - a \geq 0$ *and* $d - c \geq 0$.

$$(b - a)(d - c) \geq 0$$
$$\Longrightarrow bd - bc - ad + ac \geq 0$$
$$\Longrightarrow bd + ac \geq bc + ad$$

Therefore, we proved $ac + bd \geq ad + bc$.

By proving $ab + cd \geq ac + bd$ and $ac + bd \geq ad + bc$, we have proved the inequality of $ab + cd \geq ac + bd \geq ad + bc$. *QED*

2.2 **[12 marks]** Design a $O(k \log k)$ algorithm which determines the maximum score that UNSW can achieve to ensure it has the best chance of winning.

For any set of 4 students $\{(L_i, L_j), (L_k, L_l)\}$, the score is the largest when $L_i \leq L_j \leq L_k \leq L_l$.
Let L be a sorted list of the scores of 2n student $\{L_1, L_2 \ldots, L_{2n}\}$ where $L_1 \leq L_2 \leq \ldots \leq L_{2n}$.
And let L' be a permutation of L, S_{2k} be the final score of the first 2k students in L and S'_{2k} be the final score of the first 2k students in L'.

$$S_{2k} = \sum_{i=1}^{k} L_{2i-1}L_{2i}\,, S'_{2k} = \sum_{i=1}^{k} L'_{2i-1}L'_{2i}$$

When k =2 (i.e. there are 4 students), we know that the score the largest when $L_1 \leq L_2 \leq L_3 \leq L_4$, that is: $S_4 \geq S'_4$.
Suppose when k = 2n-2, $S_{2n-2} \geq S'_{2n-2}$.
Then for $S_{2n} = S_{2n-2} + L_{2n-1}L_{2n,}\, S'_{2n} = S'_{2n-2} + L'_{2n-1}L'_{2n}$.
Since L is sorted , $L_{2n-1}L_{2n,} \geq L'_{2n-1}L'_{2n}$ *and* $S_{2n-2} \geq S'_{2n-2}$.

$$\Longrightarrow S_{2n-2} + L_{2n-1} * L_{2n} \geq S'_{2n-2} + L'_{2n-1} * L'_{2n}$$

Therefore, the score is the largest when L is sorted.
From the induction proof, we know that we can find the maximum sum of pairs from forming the pair by $\{(L_1, L_2),\ (L_3, L_4)\ldots.,\ (L_{2n-1}, L_{2n})\}$ even when there are k students. As we always get the largest score from each pair, we will get the score as maximum as we can when we sum all the pairs, which applies the greedy method.

That is, the score of L is largest when L is in ascending order and we pair the students by the approach we mentioned above. We can use any sorting algorithm to sort L and make it an ascending array. Since our goal is to find an O(klogk) algorithm, we apply merge sort to array L. We traverse the array L by iterating over two elements at a time. (If Li, L(i+1) is the first set, then the next set will be L(i+2), L(i+3)) During the traversal until reaching the end of the array, we calculate the sum of the product from each pair, which is the maximum score.

Time complexity
By applying merge sort the time complexity O(2k log 2k) = O(k log k).
And we traverse k pairs in the array that the time complexity is O(2k) = O(k).
Thus, the time complexity is O(k logk) + O(k) = O(k logk).

2.3 [**8 marks**] USYD is also competing in this competition and they have also run a mock competition. UNSW gets to choose USYD's team pairings. Design a $O(k \log k)$ algorithm which determines the minimum score that USYD can achieve.

For any set of 4 students $\{(L_i, L_l), (L_k, L_j)\}$, the score is the smallest when $L_i \leq L_j \leq L_k \leq L_l$. We take S as the sum of those two pairs. We use induction to proof that we can find the minimum score when the pair is form by (L_1, L_{2k}), (L_2,L_{2k-1}), …., (L_k,L_{k+1}).

Let L be a sorted list of the scores of 2n student $\{L_1, L_2 \ldots, L_{2n}\}$ where $L_1 \leq L_2 \leq \ldots \leq L_{2n}$.
And let L' be a permutation of L, S_{2k} be the final score of the first 2k students in L and S'_{2k} be the final score of the first 2k students in L'.

$$S_{2k} = \sum_{i=1}^{k} L_i L_{k+1-i}, S'_{2k} = \sum_{i=1}^{k} L'_i L'_{k+1-i}$$

When k =2 (i.e. there are 4 students), we know that the score the largest when $L_1 \leq L_2 \leq L_3 \leq L_4$, that is: $S_4 \leq S'_4$.
Suppose when k = 2n-2, $S_{2n-2} \geq S'_{2n-2}$.
We use the similar approach as in 2.2, we know that $S_{2n-2} \geq S'_{2n-2}$.
That is, the score of L is minimum when L is in ascending order and pairs the largest one with the smallest one.

We can use any sorting algorithm to sort L and make it an ascending array.
Since our goal is to find an O(klogk) algorithm, we apply merge sort to array L.
As a result, we set the first and the last element as a pair, and move inward both first and last elements' indexes. That is, (L_1, L_{2k}), (L_2,L_{2k-1}), (L_3,L_{2k-3}) and so on. During the traversal until reaching the middle two elements of the array, we calculate the sum of the product from each pair and return the minimum.

Time complexity
By applying merge sort the time complexity O(2k log 2k) = O(k log k).
And we traverse k pairs in the array that the time complexity is O(2k) = O(k).
Thus, the time complexity is O(k logk) + O(k) = O(k logk).