Question 4

Zeal Liang

Z5325156

First, we use a divide and conquer idea to keep bisecting the array S until only one number x remains in each subarray.

Then suppose there is a polynomial P_i which is equal to 0 when x is substituted in. i.e., let $P_i(x) = 0$.

We can simply design this polynomial as x minus itself, e.g., $P_i(x) = x - a$, where $a = x$. This is done for each number at the bottom, and then each polynomial is merged upwards, specifically by multiplying upwards. i.e., $P_1 \times P_2$. For each polynomial multiplication we can use the fast Fourier transform algorithm to speed up the computation time.

Then, the final polynomial at the top is the answer we are looking for. We can get the same result by substituting any number from the array S into this polynomial. i.e., $P(t_1) = P(t_2) = \cdots = P(t_n)$.

Let's take an example, if S = [1, 2, 3, 4], first we divide the array into [1, 2] and [3, 4], and then divide the left subarray once more. Then calculate the corresponding polynomial in each case. Do the same for the right subarray when you have finished with the left case. At the bottom we can get these polynomials separately:

$$P_1(1) = 0, P_1 = x - 1; \quad P_2(2) = 0, P_2 = x - 2;$$

$$P_3(3) = 0, P_3 = x - 3; \quad P_4(4) = 0, P_4 = x - 4;$$

Of course, we have already finished merging the left subarray case before merging the right one, but here we just show the results of all merges to the upper level directly:

$$\boldsymbol{P_l = P_1 \times P_2 = x^2 - 3x + 2,}$$

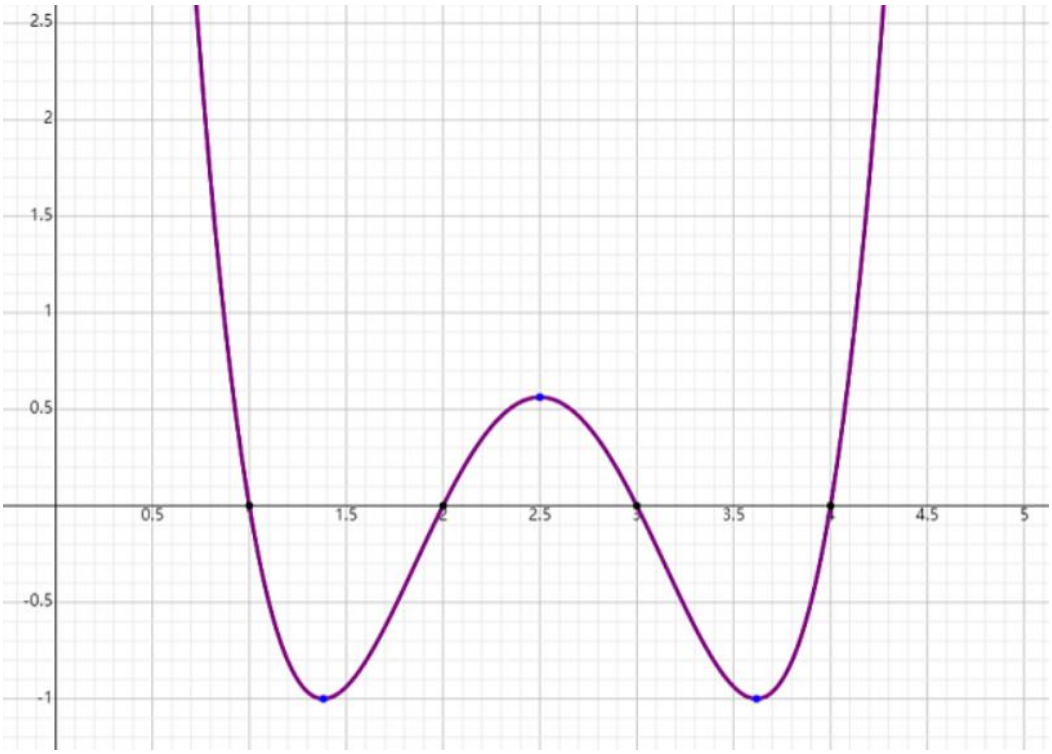
$$\boldsymbol{P_r = P_3 \times P_4 = x^2 - 7x + 12.}$$

Finally, merging one more layer up gives:

$$\boldsymbol{P_{ans} = P_l \times P_r = x^4 - 10x^3 + 35x^2 - 50x + 24.}$$

Since it is already at the top, $\boldsymbol{P_{ans}}$ is the answer we are looking for, and by looking at the graph of $\boldsymbol{P_{ans}}$ below we can clearly see that:

$$\boldsymbol{P_{ans}(1) = P_{ans}(2) = P_{ans}(3) = P_{ans}(4) = 0.}$$



This satisfies the requirement of the question: $\boldsymbol{P(t_1) = P(t_2) = \cdots = P(t_n).}$

The divide-and-conquer part of the algorithm requires a time complexity of $O(log\ n)$, and for each polynomial multiplication done with the fast Fourier transform algorithm requires a time complexity of $O(n\ log\ n)$. Therefore, the final time complexity of this algorithm is $O(n \log^2 n)$, which satisfies the requirements of the question.