Homework 3.
Due: Monday, September 20, 2021, before 8am EDT.

Problem 1 (Closest element)

We are given as an input two arrays of n distinct natural numbers A = [a1; a2;……an] and B =[b1; b2; ……. bn]. For each element ai of A, we want to output the element bj of B that is closest to Ai.

Step1: -

Merge sort the array B. So now our array B is sorted where b1<b2<……bn.

Step2: -

We will start loop across array A=[a1,a2,3,..an] until n . For each element for array A[i] we will run modified binary search as per step 3, for A[i] in Sorted Array B= [b1,b2,b3…bn] .In this binary search, A[i] will be our target value for which we have to find the closed element in Sorted array B.

Step3: -

Let's start custom binary search by first getting mid element of array B. Now check base case, if mid is not less than 0 and it's not greater than n-1 . Here A[i] is our target closet element.

If mid element B[mid]=A[i] then we will return C[i]=A[i] and go to step 3.

If target is less than element at array B of current mid i.e., B[mid] > A[i]. Next check for condition if target is great than previous to mid i.e. B[mid-1] < A[i]. If this condition is true, then we will check if current mid or previous mid is closed to target and return closet element to array C[i]. Otherwise continue binary search on left subarray from start to mid and go to step 3.

If target is greater than B[mid] i.e.[mid] is < A[i]. Next check for condition if target is less than next to mid i.e. B[mid+1] > A[i]. If this condition is true, then we will check if current mid or next element to mid is closed to target and return closet element to array C[i]. Otherwise continue binary search from mid to n and go to step 3.

Step4.
Finally, if search is finished from step 3 we will return B[mid]

Step5.
Continue from step 2 to step 4 for next element at array A[i] until n and Finally return array C.

Correctness: -
As we have sort the Array B and we are using modified binary search to find closed element of array A by comparing depending upon whether B[mid],B[mid-1] or B[mid+1] is closet we will find the closet element for A[i] and storing it new Array C.

Runtime: -
For step 1 run time for merge sort is O(n log n) .

Then for step 2,3,4,5 for checking each element at array A [a1,a2,..an] and doing binary search on array B is O(n log n) . O(log n) for binary search of Array B [b1.b2,……bn] and O(n log n) for n elements of array A.

Return array C is o(1)

So total run time is O(n log n) + O(n log n) + O(1) = O(n log n)

Problem 2 (FFT application)

We are given a set S = f s1; s2; : : : ; sn} of n distinct natural numbers such that 0 <= si <=100n. We have to design an algorithm that takes as input S and a natural number N, and outputs True if the equation si + sj + sk = N has at least one solution, and return False otherwise.

Step1: -

For each element of Set S[i] of S we will convert them to three polynomial form such as where a[i]= a , b[i] = $b^{S[i]} and\ c$[i] = $c^{S[i]}$. Each element on array will act as exponents in polynomial E.g. for set S=[1,2,3,5,7] Polynomial equation a Can be written as:-

a=1 x+ 1 x^2 +1 x^3 + 1 x^5 + 1 x^7

Similarly, second polynomial equation b will be:

b=1 x+ 1 x^2 +1 x^3 + 1 x^5 + 1 x^7

and

c= 1 x+ 1 x^2 +1 x^3 + 1 x^5 + 1 x^7

Step2:-

We will do FFT multiplication of above equation. FFT_Multiply(a,b)

If a or b=1 then we will return ab.

Express a,b in n higher order vector coefficient form for a(x) and b(x).

a'=[a0,a1,.....an-1, 0, 0,0]

b'=[b0,b1,.....bn-1,0, 0,0]

FFT Evaluate ------→2n distinct points

y=FFT(a,Omega=2nd root of unity)

z=FFT(b,Omega=2nd root of unity)

M=y*z

d(x)=IFFT(M)

Step3: -

Now we Repeat FFT multiply for polynomial equation d(x) and c and from this we will get final polynomial E(x). Where E(x) is the polynomial multiplication of E(x)=c*(a*b)

Step4:-

We will check final multiplication form for e(x) if any of the exponents of polynomial is equal to N Then return True Otherwise Return False

Correctness: -

First we have convert all values of set S in three similar polynomial equation a ,b and c where exponents of equation are value of element S and all of them have coefficients of 1. Now we have used FFT as Blackbox to multiply both polynomial equation a , b and get final polynomial d . Then again, we multiply C to d and get the final polynomial equation E.

On polynomial equation E, we check if any of the exponents of this E is equal to value N , then we have solution of problem given otherwise return false. Because we have returned all positive sum from this three equation and repletion is allowed so this will always be true for any value of N. If on final equation none of exponents is equal to N then return False.

Runtime:-

Converting set S to each polynomial equation is o(1) .

Then doing FFT multiplication for Step 2 is O(n log n)

Similar doing FFT multiplication for Step 3 is O(n log n).

Then checking N for each exponent on final equation is O(1).

So total run time is o(1)+O(n log n) + O(n log n)+ O(1)=O(n log n)

References: -

https://cp-algorithms.com/algebra/fft.html#toc-tgt-9

https://math.stackexchange.com/questions/764727/concrete-fft-polynomial-multiplication-example