HW3 DC1

Chengqi Huang chengqihuang@gatech.edu

Problem 1

Given:

A be an array of N distinct numbers(unsorted), k < n/2

Output:

The kth elements of A that are close to the median of A

Solution:

Since the Array is un-sorted array, and we are looking for specific positions, the linear-time median algorithm would be a good idea to start and find median of A. Then find the difference between median and all other elements and store this in a new array, and then using linear-time algorithm to find the kth smallest number for the new array.

Steps:

1: using linear-time median algorithm to find the median of A:

choose pivot: p

Partition A into: A1 < P, A2 = P, A3 > P

If median <= A1, using A1 as A, recursively run the algorithm

If A1 < median <= A1 + A2, return P

If n/2 - k/2 > P, using A2 as A, k - A1as k, recursively run the algorithm

2: subtract median from all elements in A, store the absolute results value in B: find the difference between all elements in A and the median

3: run the same linear-time algorithm at array B to find the kth smallest element: K.

4: traverse all elements in B, if B[i] < K, include A[i] in the output array

Runtime:

Step1: O(n)
Step2: O(n)
Step3: O(n)
Step4: O(n)
Overall runtime: O(n)

Proof of correctness:

First, find the median, then, find the k-th closet numbers to the median. Since we assume n and k are even numbers, the output gives k-th nearest neighbors to the median of array A.

Problem 2

(a)

$A(x) = a_0 * x^{(n-1)} + a_1 * x^{(n-2)} + \ldots + a_{n-1} * x^0$

$A(2) = a_0 * 2^{(n-1)} + a_1 * 2^{(n-2)} + \ldots + a_{n-1} * 2^0 = a$

(b)

Given:

2 n-bit integers a and b

Output:

Multiply them in O(nlogn) time

Solution:

$a = a_0 * x^{(n-1)} + a_1 * x^{(n-2)} + \ldots + a_{n-1} * x^0$, $b = b_0 * x^{(n-1)} + b_1 * x^{(n-2)} + \ldots + b_{n-1} * x^0$, using FFT to compute a(x1), a(x2), … , a(a2n), and b(x1), b(x2), …, b(x2n), then for i = 1->2n, c(xi) = a(xi) *b(xi), then convert c(xi) to $c_0, c_1, \ldots, c_{n-1}$ using reverse FFT

Steps:

1. Create polynomial expression for a and b. $a = a_0 * x^{(n-1)} + a_1 * x^{(n-2)} + \ldots + a_{n-1} * x^0$, $b = b_0 * x^{(n-1)} + b_1 * x^{(n-2)} + \ldots + b_{n-1} * x^0$
2. Using FFT to convert a and b to points. Run FFT(a, w2n), output: $(i_0, \ldots, i_{2n-1})$ and FFT(b, w2n), output: $(j_0, \ldots, j_{2n-1})$
3. For q = 0 -> 2n-1, $k_q = j_q * i_q$
4. Reverse FFT: 1/2n * FFT(k, $w2n^{(2n-1)}$), output: $c_0, \ldots c_{n-1}$
5. Output $c = c_0 * 2^{(n-1)} + c_1 * 2^{(n-2)} + \ldots + c_{n-1} * 2^0$

Run time:

Step1: O(n)

Step2: O(nlogn)

Step3: O(n)

Step4: O(nlogn)

Step5: O(n)

Overall runtime: O(nlogn)

Proof of correctness:

FFT: convert polynomial to points, because the multiply is faster, then convert back.