Maoting Ren
mren

**3-1.** Computing the Coefficients of a Polynomial Given its Roots. Please do problem 462 in the textbook draft (in the FFT chapter).

**Solution**: We can use divide-and-conquer algorithm for computing the coefficients.

1. Each time we recursively divide $A(x) = (x - r_1)(x - r_2)...(x - r_n)$ into two halves, until the length is one
2. Then merge the two halves by FFT. Finally, we can get the result.

As we know, the time complexity of FFT is O(n log n), and the time complexity of divide-and-conquer is O(log n), so the total time complexity is $O(n \log^2 n)$.

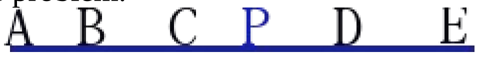
**3-2.** The Post Office Problem. Please do problem 60 (both parts) in chapter 3 (the sorting chapter) of the textbook draft.

**Solution**:

**a)** Firstly, we consider the one-dimensional variant where the businesses are points on a number line. In this condition, we should find a point (assume it is **A**) that the sum of distances between A and other points is minimum. I believe this point should be the median of all the points, and I will show you why is it by the following demonstration:

**1.** Suppose we have two points on a line, and want to find a point **P** that has a minimum sum of distances between **P** and **C**, **D**. It is obvious that **P** should between **C** and **D**, and the sum of distances is |**D** – **C**|. If **P** is not between **C** and **D**, the total distances is |**P** – **C**| + |**P** – **D**|, because **P** is not between **C** and **D**, so either |**P** – **C**| or |**P** – **D**| will large than |**D** – **C**|, and the total distances will absolutely large than |**D** – **C**|.

**2.** And now we have been known that if we want to minimum the sum of distances between **P** and **C**, **D**, we should find a point between **C** and **D**. Then we can apply this method to solve one-dimensional post office problem.



**3.** Suppose we want to find a point **P** that has a minimum sum of distances between **P** and all other points. Firstly, **P** should between **A** and **E**, this makes the sum of distances between **P** and **A**, **E** is minimum. Then we delete the points **A** and **E**, use the same way we know P should between **B** and **D**. By this way we know **P** should at the same position with **C**, which is median. If the number of points is even, then **P** can can be at any position between the middle two points.

**4.** And now we have been solved the one-dimensional post office problem. We can apply this algorithm to two-dimensional  post office problem. Because the distance between two points is "Manhattan" distance, that is the distance between two points $(x_1 , y_1)$ and$(x_2 , y_2)$ is given by $|x_1 - x_2| + |y_1 - y_2|$. So we can separately find the median in X coordinate(assume it is **px**) and Y coordinate(assume it is **py**), and we can build the post office at (**px**, **py**) coordinate.

**5.** The last thing is how to find the median in O(n) time. We know in quick-sort algorithm, we can find the **k**th element in O(n) time, and in this problem, we can find our median by this way too.

Done!

**b)** We can convert this problem to one-dimensional because they don't depend on each other. So we see how to solve it in one-dimensional first.

**1.** Assume we have many points on a line, they are $x_1, x_2, x_3,..., x_n$, and their weights are $w_1, w_2, w_3,..., w_n$. We want to find a point **x** that minimum the $f(x) = \Sigma(\mathbf{x} - x_i)*w_i, 1 <= i <= n$. That is we want a weighted median that satisfy the following condition:

$g(w) = \Sigma w_i < \mathbf{W}/2, x_i < \mathbf{x}$, **W** is the total weights

$g(w) = \Sigma w_i <= \mathbf{W}/2, x_i > \mathbf{x}$

**2.** Now we should find the weighted median in O(n) time. We can use the following method:

**1.** Find the median of all points, suppose it is $\mathbf{x_i}$ we know it can be solved in O(n) time according to **a)**

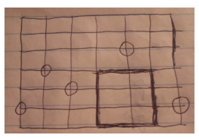
**2.** Then compute the total weights of both halves, suppose they are $W_{left}$ and $W_{right}$. If they satisfy $W_{left} < \mathbf{W}/2$ and $W_{right} <= \mathbf{W}/2$, then we know $\mathbf{x_i}$ is the weighted median. If $W_{left} > W_{right}$, we recursively search the left half, else we search the right half. By this way, we can find the weighted median

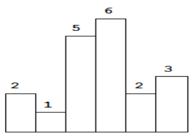
**3.** Apply this method to Y coordinate, we will find the weighted median in both X and Y coordinate, assume they are (**x**, **y**), and this is the best point to build our post office.

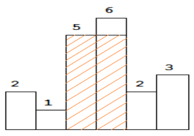
**3-3.** Building a House. You have just purchased a new rectangular-shaped lot of land, containing n trees. Given the (x, y) coordinates of each tree (think of each tree as a point), please describe an O(n log n) algorithm for finding a sub-rectangle of maximum area within your lot that (i) contains no trees inside it, and (ii) has at least one of its four edges coincide with the edge of the lot (i.e., at least one edge of the house must run directly along the edge of the lot for some amount of distance).

**Solution**: As we know from this problem, we should find a sub-rectangle of maximum area at least one edge of the house run directly along the edge of the lot. So we can separately search four edges, and record the maximum area. And I will show you how to find a maximum area in one edge. And then we can apply this method to other edges. Here we go! It will be a magic!

Firstly, we sort the points by x coordinate. Then we convert this problem to another problem:







In this problem, our task is to find the area of largest rectangle in the histogram. The height of histogram is the tree's coordinate (y -1).

we can maintain a non-decrease stack, and do the following steps:

**1.** if current bar is large than the top element of stack or stack is empty, this bar should push into stack.

**2.** if current bar(assume it is **k**) is small then the top element of stack, then we pop the top element (assume it's **p**) of stack until the top element is smaller than **p**. We can compute the maximum area based on the top element **p** as the highest bar, because we know that the stack is non-decrease, so we can find the adjacent smaller element in it's left side in O(1) time, that is the new top element. And we already know that **k** is smaller than **p**, and is **p**'s right adjacent smaller element. If **k** is large than the new top element, then we push **k** into stack, else we pop the top element of stack, and do the above operations.

**3.** Scan each bar of this histogram, and maintain a maximum area, after the stack is empty, we can get the maximum area.

After we apply this method to all edges, we can get the maximum area. Done!

**3-4.** Counting Dominated Points in 3D. In 3D, a point (x, y, z) dominates the point (x 0 , y0 , z0 ) if x ≥ x 0 , y ≥ y 0 , and z ≥ z 0 . Please give an O(n log n) algorithm for counting the number of dominated points in a 3D set of n input points.

**Solution**: We can do the following steps to counting the number of dominated points
**1.** Sort the points by X coordinate, and this operation will cost O(n log n) time

**2.** Scan the points from the point with maximum X coordinate to the point with minimum X coordinate

**3.** During scan the points, maintain a current maximum y coordinate **maxY**and z coordinate **maxZ**. Each time when scan a point, we can see if this point's Y coordinate or Z coordinate is large than the maximum value we scanned before. If the answer is yes, this point is Non-dominated point. By this way we can count the number of  Non-dominated points(assume it is **m**).

4. Subtract **n** by **m**, we can get the result, that is dominated points' number is (**n** - **m**)