# Algorithm Homework 2

龙肖灵
Xiaoling Long
Student ID.:81943968
email:longxl@shanghaitech.edu.cn

October 23, 2017

## Collecting Toys

There are $n$ types of toys that you wish to collect. Each time you buy a toy, its type is randomly determined from a uniform distribution (i.e., all possible types have equal probabilities). Let $p_{i,j}$ be the probability that just after you have bought your $i$th toy, you have exactly $j$ toy types in your collection, for $i \geq 1$ and $0 \leq j \leq n$.

(a) Find a recursive equation of $p_{i,j}$ in terms of $p_{i-1,j}$ and $p_{i-1,j-1}$ for $i \geq 2$ and $1 \leq j \leq n$.

(b) Describe how the recursion from (a) can be used to calculate $p_{i,j}$ .

*Solution.*

(a) Recursive equation of $p_{i,j}$: $p_{i,j} = \frac{j}{n}p_{i-1,j} + \frac{n-j+1}{n}p_{i-1,j-1}$.

(b) First we should initialize the initial probability $p_{i,j} = 0 s.t. i < j \ or \ i \neq 0 \& j = 0 \ and \ p_{0.0} = 1$.
Then we can calculate all probability $p_{i,j}$

Done. ■

## Knapsack II

Given $n$ objects and a knapsack, item $i$ weighs $w_i > 0$ kilograms and has value $v_i$ where $n > v_i > 0$. The knapsack has capacity of$W$ kilograms. The numbers $n$, $v_i$ are integers and $w_i$ , $W$ are real numbers. What is the maximum total value of items that we can fill the knapsack with? Design an efficient algorithm. For comparison, our algorithm runs in $O(n_3)$.

*Solution.* We can using DP algorithm to minimize the weight on constant value.

1) Compute $V = \sum_i^n v_i$ and we know that $V < n \times max(v_i) < n^2$.

2) Define $OPT(i, v)$ is min weights of items selected from $1, \cdots, i$ whose total value equal $v$ . And if $1, \cdots, i$ can make total value be $v$, then $OPT(i, v) = 0$.

3) Then we have

$$OPT(i, v) = \begin{cases} 0 & if \ i = 0 \\ w_i + OPT i - 1, v - v_i & if \ OPT(i-1, v) = 0 \\ min\{w_i + OPT(i-1, v - v_i), OPT(i-1, v)\} & otherwise \end{cases}$$

4) We can get all probability $p_{i,j}$.

And the run time is $O(nV) < O(n^3)$ Done. ■

## Counting Friends

There are n students and each student $i$ has 2 scores $x_i, y_i$ . Students $i, j$ are friends if and only if $x_i < x_j$ and $y_i > y_j$ . How many friends are there? Design an efficient algorithm. For comparison, our algorithm runs in $O(nlogn)$ time.

*Solution.*

1) First we sort based on score $x_i$. And we now that for all $i > j \ \rightarrow \ x_i > x_j$. ($O(n\ log\ n)$)

2) Use Divide-and-Conquer to counting inversions of $y_i < y_j$.

   (1) Divide: separate list two pieces.

   (2) Conquer: recursively count inversions in each half.

   (3) Combine: count inversions where $y_i$ and $y_j$ are in different halves(merge and count), and returns sum of three quantities.($O(n\ log\ n)$)

So total run time is $O(n\ log\ n + n\ log\ n) = O(n\ log\ n)$. Done. ■

## XOR Convolution

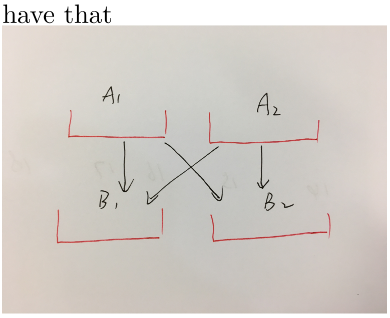
Given two arrays $A = a_0, a_1, \cdots, a_{n-1}$ and $B = b_0, b_1, \cdots, b_{n-1}$, return an array $C = c_0, c_1, \cdots, c_{m-1}$, where $c_i = \sum_{j \oplus k = i} a_i b_k$ . Design an efficient algorithm. For comparison, our algorithm runs in $O(n\ log\ n)$ time.
$\oplus$ is the bitwise XOR operator: $https://en.wikipedia.org/wiki/Bitwise_operation\#XOR$
Hint: Define $x^i \cdot x^j = x^{i \oplus j}$ , and imitate the Karatsuba algorithm.

*Solution.*

1) We have that $0 \oplus R = R$ , $R \oplus R = 0$ and $A \oplus X \oplus A \oplus Y = X \oplus Y$.

2) So we can use Divide-and-Conquer to solve this problem.

3) Before divide we have $n \times n$ cicle. So we divide the $A, B$ into two parts $(A_1, A_2), (B_1, B_2)$. And we have that




   If $j \oplus k = i$, we calculate together so we didn't need $n \times n$. We compute $\oplus$ first. $A_1 - B_2$ and $A_2 - B_1$ is same. And $A_2 - A_2$ we can convert to same as $A_1 - B_2$ because of $A \oplus X \oplus A \oplus Y = X \oplus Y$. So we just need to compute 2 parts($A_1 - B_1, A_1 - B_2$). And we recursively do these operation. Combine we just merge same $i = j \oplus k$. So sum all $a_j b_k$.
   And the run time is $O(n\ log\ n)$.

Done. ■

## DNA Pattern Recognition

There are four possible bases in a DNA sequence: $A, G, C, T$. Suppose we have two DNA sequences $S$ and $P$ with length $n$ and $m$ where $\sqrt{n} < m < n - \sqrt{n}$. Design an efficient algorithm to find out the minimum number of bases in $P$ that we have to change so that $P$ is a substring of $S$. For comparison, our algorithm runs in $O(n\ log\ n)$ time.
For instance, $S = AGCTAGGCTCT$, $P = AAGTCTC$. The answer is 2. We can change $P$ to $TAGGCTC$.
Hint: An application of FFT.

*Solution.* ■

## 2D Inversions

Given an array of 2D pairs $A = a_0, a_1, \cdots, a_{n-1}$ where $a_i = (x_i, y_i$, define $a_i > a_j$ as $x_i > y_j$ and $y_i > y_j$ .

(a) How many half-inversions are there? $a_i$ and $a_j$ are half-inverted if $i < j$, $x_i > x_j$ and $y_i \geq y' > y_j$ where $y'$ is a fixed constant. Design an efficient algorithm. For comparison, our algorithm runs in $O(n\ log\ n)$ time.

(b) How many cross-inversions are there? $a_i$ and $a_j$ are cross-inverted if $i < i' \leq j$ and $a_i > a_j$ where $i'$ is a fixed constant. Design an efficient algorithm. For comparison, our algorithm runs in $O(n\ log\ n)$ time.

(c) How many inversions are there? $a_i$ and $a_j$ are inverted if $i < j$ and $a_i > a_j$ . Design an efficient algorithm. For comparison, our algorithm runs in $O(n\ log^2\ n)$ time.

*Solution.*

(a) Use Divide-and-Conquer.

- Divide: separate list two pieces.
- Conquer: recursively sort $a_i$ based on $x_i$ and count inversions in each half.
- Combine: count inversions where $x_i > x_j$ and meanwhile satisfy that $y_i \geq y' > y_j$ are in different halves(merge and count), and returns sum of three quantities.($O(n\ log\ n)$).

The run time of algorithm is same as counting inversions. It's $O(n\ log\ n)$.

(b) Similar with question (a). But we don't recuisively count. We sort, merge and count inversion.

- Divide: separate list two pieces from $i'$.
- Sort: Sort $a_i$ based on $x_i$ in both two parts. ($O(n\ log\ n)$)
- Count: Merge and count. If $x_i > x_j$ and meanwhile satisfy that $y_i > y_j$ are in different halves, then the number of cross-inversions plus 1. ($O(n)$).

The run time of algorithm is same as counting inversions. It's $O(n\ log\ n)$.

(c) Similar with question (a).

- Divide: separate list two equal pieces.
- Conquer: recursively sort $a_i$ based on $x_i$ count inversions in each half.
- Combine: count inversions where $x_i > x_j$ and meanwhile satisfy that $y_i > y_j$ are in different halves(merge and count), and returns sum of three quantities.($O(n\ log\ n)$).

The run time of algorithm is same as counting inversions. It's $O(n\ log\ n)$.

Done. ■