Exercise 1

(a)

T : text in length of n

P : pattern in length of m

Exact expected number of comparisons between characters in terms of d,n,m

Assume that

C(d,m,n) is the expected number of comparisons when d = size of alphabet Σ,

m = length of pattern P, n = length of text T.

think of when comparing first character in P and T,

$C(d,m,n) = 1 + \frac{d-1}{d}C(d,m,n-1) + \frac{1}{d}(1 + \frac{d-1}{d}C(d,m,n-1) + \left(1 + \frac{d-1}{d}C(d,m,n-1) + \cdots\right) + d^{-m}C(d,m,n-1)$

1 for the first comparison,

$\frac{d-1}{d}C(d,m,n-1)$ for when that is mismatch, comparing next character in T,

rest is for checking the next charter in P and T.

$$C(d,m,n) = \left(1 + \frac{1}{d} + \frac{1}{d^2} + \cdots + \frac{1}{d^{m-1}}\right) + (1-d^{-1})C(d,m,n-1)\left(1 + \frac{1}{d} + \frac{1}{d^2} + \cdots + \frac{1}{d^{m-1}}\right) + d^{-m}C(d,m,n-1)$$

$C(d,m,n) = \frac{1-d^{-m}}{1-d^{-1}} + C(d,m,n-1)$

for 1 shift, the $\frac{1-d^{-m}}{1-d^{-1}}$ times of comparisons are needed, there are (n-m+1) shifts,

$C(d,m,n) = \frac{1-d^{-m}}{1-d^{-1}}(n-m+1)$

the expected number of comparisons $=\frac{1-d^{-m}}{1-d^{-1}}(n-m+1)$

(b)

I used my brute-force algorithm implementation with given example text file

The length of text file = 1254136

I put finding pattern = "all"

The number of comparisons were 1339033

The length of text = n = 1254136

The length of pattern = m = 3

Numbers of character from English-keyboard = d =95 expected number of comparisons = $\frac{1-d^{-m}}{1-d^{-1}}(n-m+1)$

the expected number of comparisons = 1267474

the actual number of comparisons = 1339033 from the code execution.

The analysis from (a) is confirmed although the text is not random.

Exercise 2

<2dimensional pattern P>

if m=2

	0	1	2
0	a	b	a
1	a	a	b
2	b	a	a

<2dimensional text T>

if n=5

	0	1	2	3	4
0	a	b	a	f	s
1	e	a	b	d	g
2	b	a	a	e	b
3	c	d	a	e	d
4	t	r	q	w	u

What was instructed in exercise is like this (don't care about the characters in array).
P is mxm array, and T is nxn array (not 2 and 3, that is just an example)

(a)

```
void BruteForceIn2D( T, P, m, n)
    int i, j, k, h;
    bool mismatch;
    for(i=0 ; i<=n-m ; i++)
        for(j=0 ; j<=n-m; j++)
            mismatch = false;
            for(k=0; k<m&&!mismatch; k++)
                for(h=0; h<m&&!mismatch; h++)
                    if T[i+k][j+h]!=P[k][h],   // if character doesn't match
                        mismatch = true;
            if !mismatch,
                print (i,j) // left-up corner point where it matches
```

Analysis

Worst case

In worst case of pattern and text made with all-same-characters,

We have $(n-m+1)^2$ times of shifts,

per one shift,

we have m^2 comparisons.

In worst case,

Time complexity = $\theta(m^2(n-m+1)^2)$

When n>>m,

Time complexity = $\theta(n^2m^2)$

(b)

original Rabin-Karp-algorithm computed P interpret P as a d-ary

(from class)

like $p = p_o d^{m-1} + p_1 d^{m-2} + \cdots + p_o$

($p_i = assigned\ number\ to\ character\ P[i]\ \ when\ P\ was\ linear\ array$, d = |∑|)

in this exercise,

I will modify Rabin-Karp-algorithm to use that computing method twice.

First, I will use it vertically

($p[0,i] = assigned\ number\ to\ character\ P[0][i]$, $t[0,i] = assigned\ number\ to\ character\ T[0][i]$)

(d = |∑|)

for i from 0 to m-1

$p'[i] = p[0,i]d^{m-1} + p[1,i]d^{m-2} + \cdots + p[m-1,i]$

now we changed pattern m*m array into linear size m array of numbers.

This equation needs $\theta(m^2)$, $\theta(m)$ for getting each p[] value and $\theta(m)$ for computing(

(using Horner's Scheme from class change

the equation like $p' = p[m-1] + d(p[m-2] + d(\ldots.(p[1] + dp[0]) \ldots))$.

It is done m times, so to make p', it needs $\theta(m^3)$

Now for the first check, for i from 0 to m-1,

$t'[i] = t[0,i]d^{m-1} + t[1,i]d^{m-2} + \cdots + t[m-1,i]$

to make 1-time-shifted data t', we need $\theta(m)$ time by using t' from previous,

when it is shifted right, all data in t'[i] = t'[i+1] except t'[m-1]

and computing $t'[m-1]$ it takes $\theta(m)$ time

when it is shifted down, for each $t'[i]$, $t'[i]$=$d(t'[i] - t[0,i]d^{m-1}) + t[m,i]$

and this equation needs constant time because it is for all i, shifting needs $\theta(m)$

now from each comparison of p' and t' array,

I will make another d-ary like

$pData = p'[0]d^{m-1} + p^{'}[1]d^{m-1} + \cdots + p^{'}[m-1]$,

$tData = t'[0]d^{m-1} + t^{'}[1]d^{m-1} + \cdots + t^{'}[m-1]$,

starting from (a,b)=(0,0),

then if pData == tData, print (a,b) as an occurance,

if it's not correct then shift right a to a+1,

if a==m-1, then shift down b to b+1, and then now horizontal shifting is toward left(a->a-1) (it changes again when a reach 0)

in this order we have $(n-m+1)^2$ shifts and comparisons of pData and tData,

the algorithm will return all expected match spots(assumed).

Runtime

In worst case

(1) For computing p' and t' at first place, it is $\theta(m^3)$

(2) For every shift changes of p' and t' takes $\theta(m)$ and it is done $(n-m+1)^2$ times.

(2) needs $\theta(m(n-m+1)^2)$

(3) For computing pData and tData from p' and t'

it takes $\theta(m)$ (use techniques in (1))

it is done $(n-m+1)^2$ times, so

(3) needs $\theta(m(n-m+1)^2)$

(5) comparing pData and tData needs $\theta(1)$.

So

(1)+(2)+(3)+(4)+(5) = $\theta(m^3) + \theta(m(n-m+1)^2) + \theta(m(n-m+1)^2) + \theta(1)$

= $\theta(m^3) + \theta(m(n-m+1)^2)$

= $\theta(m(m^2 + (n-m+1)^2)$

the runtime in worst case = $\theta(m(m^2 + (n-m+1)^2)$

when n>>m,

runtime = $\theta(mn^2)$