COMP3011 Assignment2

P1

(a)

Prove: every time we get the 3 front elements of each array and push the minimum one into result $(O(1))$. And this process will execute for $3 * (n/3)$ times, so the time is $O(n)$。

(b)

By master theorem, we have $T(n) = 3O(n/3) + n$

(c)

So $c = \log_3^3 = 1, \therefore n^c = n$

$\therefore$ the time complexity is $O(n \log n)$。

Another solution: we can consider it as a divide and conquer tree, for every step, its size will become $1/3$, so there are $\log_3 n$ layers, and for each layer, the sum of length is n, so it's $O(n \log n)$。

P2

The below is a C++ implementation, it output every DP status and value.

```
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
const int N = 1005;
int n, m, f[N][N];
char s1[N], s2[N];
int main(){
    scanf("%s%s", s1 + 1, s2 + 1);
    n = strlen(s1 + 1), m = strlen(s2 + 1);
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);
            if(s1[i] == s2[j])
                f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);
            cerr << i << ", " << j << "::" << f[i][j] << endl;
        }
    }
    printf("%d\n", f[n][m]);
    return 0;
}
```

```
quantity
inequality
```

```
-
1, 1::0
1, 2::0
1, 3::0
1, 4::1
1, 5::1
1, 6::1
1, 7::1
1, 8::1
1, 9::1
1, 10::1
2, 1::0
2, 2::0
2, 3::0
2, 4::1
2, 5::2
2, 6::2
2, 7::2
2, 8::2
2, 9::2
2, 10::2
3, 1::0
3, 2::0
3, 3::0
3, 4::1
3, 5::2
3, 6::3
3, 7::3
3, 8::3
3, 9::3
3, 10::3
4, 1::0
4, 2::1
4, 3::1
4, 4::1
4, 5::2
4, 6::3
4, 7::3
4, 8::3
4, 9::3
4, 10::3
5, 1::0
5, 2::1
5, 3::1
5, 4::1
5, 5::2
5, 6::3
5, 7::3
5, 8::3
5, 9::4
5, 10::4
6, 1::1
6, 2::1
6, 3::1
6, 4::1
6, 5::2
```

```
6, 6::3
6, 7::3
6, 8::4
6, 9::4
6, 10::4
7, 1::1
7, 2::1
7, 3::1
7, 4::1
7, 5::2
7, 6::3
7, 7::3
7, 8::4
7, 9::5
7, 10::5
8, 1::1
8, 2::1
8, 3::1
8, 4::1
8, 5::2
8, 6::3
8, 7::3
8, 8::4
8, 9::5
8, 10::6
```

P3

We view rides as a segment $(l_i, r_i) = (start_i, end_i)$ and it's value is $w_i = end_i - start_i + tip_i$. So the problem is transformed to finding the maximal weight of some segments and no intersection in any length (It doesn't matter if the points intersect).

So the problem is very classic and we can use the algorithm shown in class. (Like we sort segments in increasing order of r_i , and for each segment, we use binary search to find a prefix z_i such that it has no intersection with i ($r_{z_i} \leq l_i$), our DP equation is $f(i) = max(f[i-1], w_i + f[z_i])$。 It's $O(n \log n)$ solution。

Bonus: there are more efficient and easier $O(n)$ solutions because the segment value is bounded by $[1, n]$. So we can just use f_i denoted by max value consider segment that $r \leq i$。 So we have $f_i = \max(f_{i-1}, \max_{r_j = i}(f_{l_j} + w_j))$。 From that equation, we know that each segment only contributes once, So if we implement carefully (Like using a vector in C++ or radix sorting), the time complexity is $O(n)$

P4

Lemma 1

Assume one rectangle (a, b) can be put inside another (c, d)。

For symmetry, we set $a \leq b, c \leq d$。

Then it must have ①: $a \leq c, b \leq d$。

Assume it has rotated in 90°, so $a < d, b < c$. From that we have $a < b < c, b < c < d$. So ① is sill true。

So it indicated that like for each rectangle $(a, b), a < b$, we can judge whether it can be put inside just by a single situation.

Sol

From Lamma 1, we set that for rectangle i, its two edges are $x_i, y_i, (x_i \le y_i)$. We sort the rectangle first key is x_i is increasing, second key is y_i is decreasing. (In other words, $rec_i < rec_j$ if and only if $(x_i < x_j) \vee ((x_i = x_j) \wedge (y_i > y_j))$。

So if j can be put inside i if and only if $j < i$ and $y_j < y_i$. (The corner case is $x_i = x_j$, but it's y in decreasing order so $j > i$) .

So the problem becomes the LIS (Longest increasing subsequence). We can use simple $O(n^2)$ solution in class ($f_i = \max(1, 1 + \max_{j<i, y[j]<y[i]} f[j])$)

Bonus: there are $O(n \log n)$ solutions, like using Fenwick tree or segment tree to optimize the above solution or changing the DP definition and using binary search.