

hw5

資工三 110590002 王燦竑

1.

1.a. worse case time complexity: $\Theta(n^2)$

when all element place in the same bucket,
and the bucket is sorted by selection sort,
then the time complexity is $\Theta(n^2)$

1.b. simple change make worst case time complexity to $O(n \log(n))$

use merge sort to sort the bucket,
then the worst-case time complexity is $O(n \log(n))$

2.

2.a. algorithm

get length from 1 to n,
get max price in each length by split each length into two parts and get the max price in each part,
then get the max price in length n.

```
maxPriceInLength[n]
for i in 1 to len(rod):
    for j in 1 to i:
        maxPriceInLength[i] = max(maxPriceInLength[i], price[j] + maxPriceInLength[i-j]+cuttingCost)
return maxPriceInLength[n]
```

2.b. time complexity

$O(n^2)$

3.

```
let dp[n][n] = [ -1 for every cell]
function lps(string s1, string s2, int n1, int n2):
    if n1==0 or n2==0:
        return 0
    if dp[n1][n2] != -1:
        return dp[n1][n2]
    if s1[n1-1] == s2[n2-1]:
        dp[n1][n2] = 1+lps(s1,s2,n1-1,n2-1)
    else:
        dp[n1][n2] = max(lps(s1,s2,n1-1,n2), lps(s1,s2,n1,n2-1))
    return dp[n1][n2]
```

```
input = input()
ans = lps(input, input[::-1], len(input), len(input))
```

3.a. time complexity

$O(n) * O(n^2) = O(n^3)$

4.

4.a. algorithm

dp store when the max length is index, most min cost.

```
dp[n]= []
dp[0] = 0
dp[1] = cost[1]

for i in 2 to n:
    dp[i] = inf
    for j in 0 to i:
        costJ = cost[i] + ((1+(i-j))*(i-j)/2) +dp[j]
        dp[i] = min(dp[i], costJ)

return dp[n]
```

4.b. time complexity

$O(n^2)$