# hw5

資工三 110590002 王熯竑

## 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.

```
cin = input()
dp = [['' for i in range(len(cin))] for j in range(len(cin))]
for i in range(len(cin)):
    dp[i][i] = cin[i]
for i in range(len(cin)-1, -1, -1):
    for j in range(i+1, len(cin)):
        if cin[i] == cin[j]:
            dp[i][j] = cin[i] + dp[i+1][j-1] + cin[j]
        else:
            dp[i][j] = max(dp[i+1][j], dp[i][j-1], key=len)
maxLength = len(dp[0][len(cin)-1])
maxpalindrome = dp[0][len(cin)-1]
```

### 3.a. time complexity

$O(n^2)$

## 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

$O(n^2)$