Answer to Problem Set 13

- Student Name: Gaole Dai
- Student ID: S01435587

Problem 1

Let K be the numbers of nodes in a graph G. Then, the Undirected Hamiltonian Path could be converted to Longest Path with input $G(V_1, V_n)$, and output $G(V_1, V_n, k)$.

Assume the original graph $G(V, E)$ is provided with nodes V_1 and V_n has an Undirected Hamiltonian path, which traverses all the vertices, therefore $G(V, E, K)$ is true because any two nodes in G will be connected by a path of length equal to its nodes (K therefore Longest Path problem holds).

Then, assume the graph $G'(V, E, V_s, V_e, K)$ has a Lpath of length K from V_s to V_e, which implies G' contains a simple path of length K from V_s to V_e.

But, G contains K vertices, hence traverses all vertices starting at V_s and ending at V_e forming a hamiltonian path, $G'(V_s, V_e)$.

Let $V_1 \equiv B$ and $V_n \equiv D$

Now, G has an Undirected Hamiltonian Path $\equiv$ BCAD of $K = 4$.

Therefore, G contains an optimized path of length = 4, between B and D.

`refer to https://www.geeksforgeeks.org/optimized-longest-path-is-np-complete/`

Problem 2

a. We could use Dynamic Programming similar to Knapsack to solve this problem, the pseudocode for the algorithm is shown below:

```
def cut(P, L):
    if L = 0:
        return 0
    tmp = 0
    for i in range(1, L+1):
        tmp = max(tmp, P[i] + cut(L-i, P))
    return tmp
```

Python code for the algorithm is shown below:

```
import sys
# Returns the best obtainable price for a rod of length n and
# price[] as prices of different pieces
def cut(price, n):
    if n == 0:
        return 0
    val = [0 for x in range(n + 1)]
    val[0] = 0
    # Build the table val[] in bottom up manner and return
```

```python
    # the last entry from the table
    for i in range(1, n + 1):
        max_val = -sys.maxsize - 1
        for j in range(i):
            max_val = max(max_val, price[j] + val[i-j-1])
            val[i] = max_val
    return val[n]
```

b. The time complexity is $\mathcal{O}(n^2)$, since for outer loop, the range is between $(1, n+1)$ and for inner loop, the range is $(0, i)$, the total iterations are $1 + 2 + \cdots + n + 1$ times, thus the time complexity is $\mathcal{O}(n^2)$.

c.

```python
# with the n = 4
arr = [1, 5, 8, 9, 10, 17, 17, 20, 24, 30]
print("Maximum obtainable value for rod-cutting problem is " + str(cut(arr, 4)))

===> Output: Maximum obtainable value for rod-cutting problem is 10
```

i	j_i	$val[i]$	$price[j] + val[i-j-1]$
1	0	val[1] = 1	price[0] + val[0] = 1 + 0 = 1
2	0	val[2] = max(1, 2) = 2	price[0] + val[1] = 1 + 1 = 2
2	1	val[2] = max(2, 5) = 5	price[1] + val[0] = 5 + 0 = 5
3	0	val[3] = max(5, 6) = 6	price[0] + val[2] = 1 + 5 = 6
3	1	val[3] = max(6, 6) = 6	price[1] + val[1] = 5 + 1 = 6
3	2	val[3] = max(6, 8) = 8	price[2] + val[0] = 8 + 0 = 8
4	0	val[4] = max(8, 9) = 9	price[0] + val[3] = 1 + 8 = 9
4	1	val[4] = max(9, 10) = 10	price[1] + val[2] = 5 + 5 = 10
4	2	val[4] = max(10, 9) = 10	price[2] + val[1] = 8 + 1 = 9
4	3	val[4] = max(10, 9) = 10	price[3] + val[0] = 9 + 0 = 9