1. Asympotic running time of divide-and-conquer algorithms:

(a)

$$T(n) = c \cdot n^d + ac \cdot \left(\frac{n}{b}\right)^d + a^2 c \left(\frac{n}{b^2}\right)^d + \cdots + a^L c \left(\frac{n}{b^L}\right)^d$$

$$= cn^d \left(1 + a\left(\frac{1}{b}\right)^d + \cdots + a^L \left(\frac{1}{b^L}\right)^d\right)$$

$$= cn^d \left[1 + \frac{a}{b^d} + \cdots + \left(\frac{a}{b^d}\right)^L\right]$$

$$\leq cn^d \sum_{i=0}^{\log_b n - 1} (a/b^d)^i$$

Case I:

if $a < b^d$

$r = a/b^d$

$$T(n) \leq cn^d \sum_{i=0}^{\log_b n - 1} r^i \leq cn^d \sum_{i=0}^{\infty} r^i = cn^d \frac{r}{1-r}$$

$\Rightarrow T(n) = O(n^d)$

Case II:

if $a = b^d$

$$T(n) \leq cn^d \sum_{i=0}^{\log_b n - 1} (a/b^d)^i = cn^d \log_b n$$

$\Rightarrow T(n) = O(n^d \log n)$

Case III:

if $a > b^d$

$r = \frac{a}{b^d}$

$$T(n) \leq cn^d \sum_{i=0}^{\log_b n - 1} (a/b^d)^i = cn^d \frac{r^{\log_b n} - 1}{r-1} = cn^d \frac{n^{\log_b r} - 1}{r-1}$$

$$\leq cn^d \frac{n^{\log_b r}}{r-1} = \frac{c}{r-1} n^d \cdot n^{\log_b a - \log_b b^d} = \frac{c}{r-1} n^{\log_b a}$$

$\Rightarrow T(n) = O(n^{\log_b a})$

(b)

- Merge sort

$T(n) \leq 2T(n/2) + n = n \log_2 n = O(n \log n)$

$a=2 \quad b=2 \quad d=1$

$\log_2 2 = 1$

- Binary Search

$T(n) \leq T(n/2) + 1 = \log_2 n = O(\log n)$

$a=1 \quad b=2 \quad d=0 \quad \log_2 1 = 0$

- Matrix multiplication

$T(n) \leq 8T(n/2) + O(n^2) \Rightarrow T(n) = O(n^3)$

$a=8 \quad b=2 \quad d=2$

$\log_2 8 > d = 2$

- Fast Matrix multiplication

$T(n) \leq 7T(n/2) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7})$

$a=7 \quad b=2 \quad d=2$

$\log_2 7 > 2$

2. Capter 5 : More recursions.

(a) $T(n) = 2T(n/3) + 1$

$a=2;\ b=3;\ d=0$

$\log_3 2 > 0$

$T(n) = 2T(n/3)+1 \Rightarrow T(n) = O(n^{\log_3 2})$

(b) $T(n) = 5T(n/4) + n$

$a=5;\ b=4;\ d=1$

$\log_4 5 > 1$

$T(n) = 5T(n/4) + n \Rightarrow T(n) = O(n^{\log_4 5})$

(c) $T(n) = 9T(n/3) + n^2$

$a=9;\ b=3;\ d=2$

$\log_3 9 = 2$

$T(n) = 9T(n/3) + n^2 \Rightarrow T(n) = O(n^2 \log n)$

(d) $T(n) = 2T(n-1) + 1$

$= 2(2T(n-2)+1)+1 = 4T(n-2)+3$

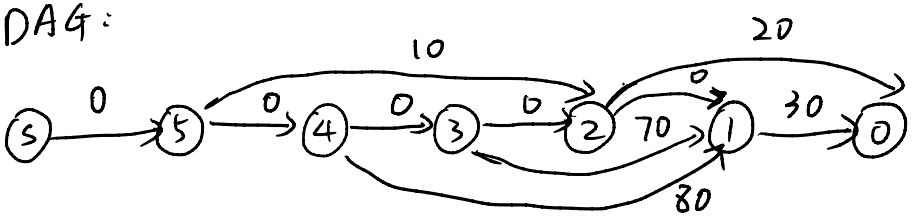
$= 4(2T(n-3)+1)+3 = 8T(n-3)+7$

$= 2^k T(n-k) + (2^k - 1)$

$T(n) = 2T(n-1)+1 \Rightarrow T(n) = 2^n - 1 = \theta(2^n)$.

3.

(a) DAG:



$p(5) = 2$ $p(3) = 1$ $p(1) = 0$

$p(4) = 1$ $p(2) = 0$

The running time of create the DAG: $O(n \log n)$

1. to calculate $p(i)$, because the intervals are sorted in an ascending order of finishing time. we can use binary-search for each interval i to find its $p(i)$ and the running time of find all intervals is $O(n \log n)$.

2. Adding edge, each interval has a $p(i)$, so the edge number is n. for the edge $(i, i-1)$ of length 0, the number is at most n.

The running time of adding edge is $O(n)$

So, the total running time is $O(n \log n)$.

(b) No, this problem need us to find the longest path in the graph. Dijkstra can deal with graph with positive edges for shortest path between two nodes.

what's more, it is a necessity for local optimun using Dijkstra.

In this question — finding longest path, the necessity isn't met.

4. Chapter 5: Variations of knapsack Problem: multi dimensional knapsack

(1) $z(i, w, z)$ denotes the optimal solution to the 2-dim kP when only item $i = 1, \cdots n$ are considered.

$$z(i,W,Z) = \begin{cases} 0 & \text{if } i = 0 \\ z(i-1, W, Z) & \text{if } w_i > W \text{ or } z_i > Z \\ \max\{ z(i-1, W, Z),\ v_i + z(i-1,\ W-w_i,\ Z-z_i)\} & \text{if } w_i \le W \text{ and } z_i \le Z \end{cases}$$

(2)

Because of the $z(i, W, Z)$ is consists of all of its subproblems, suchas $z(i-1, W, Z)$ or $z(i-1, W-w_i, Z-z_i)$, the answer recursively makes all the subproblems the optimal results. So the final results are optimal.

(3) Because the problem means that we need to fill a table of size $O(nWZ)$ like the 1-dimension question on class. each block of the table is $O(1)$. The overall complexity is $O(nWZ)$

(4) Program:

```java
public class Main {
    public static void main(String[] args) {
        int[][][] array = new int[6][12][16];
        int[] value ={0,1,6,18,22,28};
        int[] weight={0,1,2,5,6,7};
        int[] volumn={0,1,10,3,2,2};
        for(int i=0;i<6;i++){
            for(int j=0;j<12;j++){
                for(int z=0;z<16;z++){
                    if(i==0){
                        array[i][j][z]=0;
                        continue;
                    }
                    if(j==0){
                        array[i][j][z]=0;
                        continue;
                    }
                    if(z==0){
                        array[i][j][z]=0;
                        continue;
                    }
                    if(weight[i]>j||volumn[i]>z){
                        array[i][j][z]=array[i-1][j][z];

                    }
                    else{
                        array[i][j][z]=Math.max(array[i-1][j][z],value[i]+array[i-1][j-weight[i]][z-volumn[i]]);
                    }
                }
            }
        }
        System.out.println(array[5][11][15]);
    }
}
```

we get maximum $z(5,11,15) = 40$ achieved by selecting items $\{3,6\}$.

5. Chapter 5: Variations of Knapsack Problem : Knapsack with Repetition

1a)

Knapsack with reptition can be solved by the following Bellman-Ford equation.

$$OPT(w) = \begin{cases} 0 & \text{if } w = 0 \\ \max_{i,\, w_i \le w} \left(OPT(w - w_i) + v_i \right) & w \le W \end{cases}$$

The running time of the algorithm is $O(nW)$

Weight	0	1	2	3	4	5	6	7	8	9	10
Value	0	0	9	14	18	23	30	32	39	44	48

The optimal solution has value 48 and items $\{1, 4, 4\}$.

The program is as follows.

```
package KP;

public class KP_Repitition {
    public static void main(String[] args) {
        int []array=new int[11];
        int []weight={0,6,3,4,2};
        int []value={0,30,14,16,9};
        for(int i=0;i< array.length;i++){
            if(i==0){
                array[i]=0;
                continue;
            }
            else{
                int Maxx=0;
                for(int j=1;j<=4;j++){

                    if(weight[j]<=i){
                        Maxx=Math.max(Maxx,array[i-weight[j]]+value[j]);
                    }
                }
                array[i]=Maxx;
            }
        }
        System.out.println(array[10]);

    }
}
```

if n=1 or n=0

no item can be put

value is 0

if n=2

item 4 can be put

value is 9

if n=3

item 2 can be put

value is 14

if n=4

two items 4 can be put

value is 18

⋮

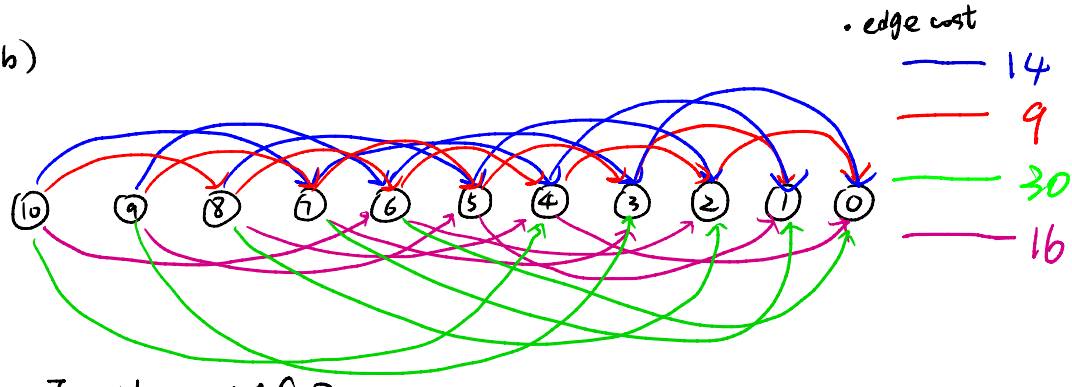
if n=10

two item 4 and one item 1

can be put

value is 48.

(b)



- Is it a DAG?

This is a DAG. The nodes in the DAG represent the subproblems, the larger subproblems point to smaller subproblems.

- Can you re-state this particular variant of the KP problem as finding a path on this graph?

In my view, this problem can be changed into finding the longest path and the cost of each edge is differentiated by distinct colors. We can draw the solution obtained with the top-down approach. As long as we obtain the longest distance from a node to any other node in the DAG, we will get the answer.