1)

a)

```
def cluster(lst):

    dl = len(lst)*[0]
    prev = 0
    for i in range(len(lst)):
        if lst[i]+prev > lst[i]:
            prev = dl[i] = lst[i]+prev
        else:
            prev = dl[i] = lst[i]

    return max(dl)


lst = [3, -5, 2, 11, -8, 9, -5]
print("max profit: {0}, arr: {1}".format(cluster(lst), lst))
print("==================================================")
lst = [3, -5, 2, 11, -8, 9, -5, 100]
print("max profit: {0}, arr: {1}".format(cluster(lst), lst))
print("==================================================")
```

O(n)

	A	B	C	D	E	F	G
	3	-5	2	11	-8	9	-5
0	3	-2	2	13	5	**14(max)**	9

In the second row, it is maintained a dynamic table, and choosed max value.

2)

```
def candy(price_list, length_list, candy_size):

    dl = (1+len(price_list))*[0]
    dl[0] = 0
    candy_index = length_list.index(candy_size)
    # for i in range(1, len(length_list)):
    max_profit = price_list[candy_index]
    for j in range(len(length_list)):
        part1 = candy_size - length_list[j];
        if part1<=0:
            continue

        p1_price = price_list[length_list.index(part1)]
        p2_price = price_list[j]

        dl[j] = p1_price + p2_price

    return max(max_profit, max(dl))
```

First given candy price is calculated without cutting, then it is cut in smaller pieces by decreasing 1 in each step, if there is no corresponding length candy in a step, it does not count. After all calculations max value is selected among them.
Complexity of the algorithm is O(n). Because only one loop is recuired.

3)

```
def pack(box_w, prices, weights):

    if len(prices) == 0:
        return 0

    box = []
    max_price = 0
    while len(prices)>0 and box_w > 0:
        max_ind= fetch_most_valuable_ind(prices, weights)
        if weights[max_ind] > box_w:
            box.append((prices[max_ind]/weights[max_ind])*box_w)
            box_w = 0
        else:
            box.append(prices[max_ind])
            box_w = box_w - weights[max_ind]

        prices.pop(max_ind)
        weights.pop(max_ind)

    for price in box:
        max_price = max_price + price

    print(box)
    return max_price
```


Complexity of the algorithm is **O(n^2)**

In each step of the while loop most valuable cheese is selected and it is put in the box until box is full, if any portion of cheese could be fit in box, the cheese is cut and box is filled full of cheese with maximum price.

4)

```
def selection(courselist):
    csl = sorted(courselist, key=lambda a: a['finish_time'])

    res = [csl[0]]
    curr = 0
    for i in range(1, len(csl)):
        if csl[i]['start_time'] >= res[curr]['finish_time']:
            curr = curr+1
            res.append(csl[i])

    return res
```


Courses is sorted according to their finish time, then first course is selected.
Then by following the sorted list next course is selected if its start_time does not intersect with the previously selected course’s finish_time.