Question 1

Suppose you're consulting for a company that manufactures PC equipment and ships it to distributors all over the country. For each of the next n weeks, they have a projected *supply* s_i of equipment(measured in pounds), which has to be shipped by an air freight carrier.

Each week's supply can be carried by one of two air freight companies, A or B.

- **Company A charges a fixed rate r per pound(so it costs $r \cdot s_i$ to ship a week's supply s_i).**
- **Company B makes contracts for a fixed amount c per week, independent of the weight. However, contracts with company B must be made in blocks of four consecutive weeks at a time.**

A *schedule*, for the PC company, is a choice of air freight company (A or B) for each of the n weeks, with the restriction that company B, whenever it is chosen, must be chosen for blocks of four contiguous weeks at a time. The *cost* of the schedule is the total amount paid to company A and B, according to the description above.

Give a polynomial-time algorithm that takes a sequence of supply values $s_1, s_2, ..., s_n$ and returns a *schedule* of minimum cost.

Example. Suppose $r = 1$, $c = 10$, and the sequence of values is

$$11, 9, 9, 12, 12, 12, 12, 9, 9, 11.$$

Then the optimal schedule would be to choose company A f or the first three weeks, then company B for a block of four consecutive weeks, and then company A for the final three weeks.

Let M_i be the total cost when there a sequence of i supply values $s_1, ..., s_i$. Then we have

$$M_i = \begin{cases} 0, & i = 0 \\ min\{4c, M_{i-1} + rs_i\}, & 1 \leq i \leq 3 \\ min\{M_{i-4} + 4c, M_{i-1} + rs_i\}, & i \geq 4 \end{cases} \tag{1.1}$$

We design a list to record each step and the precursor message. The algorithm is showed in 1.0.1.

Algorithm 1.0.1 Optimal schedule

Input: a series supplies $s_1, ..., s_n$

```
Initialize array data to be a empty list
for i = 1 to n do
  if i == 0 then
    Set data[i] to (0, 0)
  else if 1 ≤ i ≤ 4 then
    if 4c < M_{i−1} + s_i then
      Set a to -4
      Set data[i] to (a, 4c)
    else
      Set a to -1
      Set data[i] to (a, M_{i−1} + s_i)
    end if
  else
    if M_{i−4} + 4c < M_{i−1} + s_i then
      Set a to -4
      Set data[i] to (a, M_{i−4} + 4c)
    else
      Set a to -1
      Set data[i] to (a, M_{i−1} + s_i)
    end if
  end if
end for
Initialize an array W
Initialize k as data[n][1]
Initialize m as n
Initialize i as 0
while m > 0 do
  if k == −3 then
    W[i] = B
    m = m − 4
  else
    W[i] = A
    m = m − 1
  end if
  i = i + 1
end while
Reverse W
Return data[n][2] and W
```

It is to see that the process of getting minimum value and backtracking route are both

$O(n)$.

Question 2

Suppose we want to replicate a file over a collection of n servers, labeled $S_1, S_2, ..., S_n$. To place a copy of the file at server S_i results in *placement cost* of c_i for an integer $c_i > 0$.

Now, if a user requests the file from server S_i, and no copy of the file is present at S_i, then the servers $S_{i+1}, S_{i+2}, S_{i+3}...$ are searched in order until a copy of the file is finally found, say at server S_j, where $j > i$. This results in an *access cost* of $j - 1$.(Note that the lower-indexed servers $S_{i-1}, S_{i-2}, ...$ are not consulted in this search.) The access cost is 0 if S_i holds a copy of the file. We will require that a copy of the file be placed at server S_n, so that all such searches will terminate, at the latest, at S_n.

We'd like to place copies of the files at the servers so as to minimize the sum of placement and access costs. Formally, we say that a *configuration* is a choice, for each server S_i with $i = 1, 2, ..., n - 1$, of whether to place a copy of the file at S_i or not.(Recall that a copy is always placed at S_n.) The *total cost* of a configuration is the sum of all access costs associated with all n servers.

Give a polynomial-time algorithm to find a configuration of minimum total cost.

The question is equal to that we reverse the sequence of $s_1, ..., s_n$. Formally, we denote $s_1, s_, ..., s_n$ as $t_n, t_{n-1}, ..., t_1$. Let $W(p, q)$ be that there are p servers and the q-th server with a copy of the file. Then we have $q \leq p$.

Therefore we have

$$W(p,q) = \begin{cases} W(p-1,q) + p - q, & (q < p) \\ \min_{k=1}^{i-1}\{W(p-1,k)\} + c_p, & (q = p) \end{cases} \tag{2.1}$$

Algorithm 2.0.2 Minimize costs

Input: a series servers' copy-file cost $c_1, ..., c_n$ in reverse order as $(d_1, ..., d_n) = (c_n, ..., c_1)$

```
Initialize a lower triangular matrix [W_{p,q}]_{p>q}
for p from 1 to n do
  for q from 1 to p do
    if q < p then
      W[p][q] = W[p − 1][q] + p − q
    else
      W[p][q] = min_{k=1}^{p−1}{W[p − 1][k]} + d_p
    end if
  end for
end for
Return min_{k=1}^{n}{W[n][k]}
```

It is easy to see that the running time is $O(n^3)$.

Question 3

Suppose it's nearing the end of the semester and you're taking *n* courses, each with a final project that still has to be done. Each project will be graded on the following scale: It will be assigned an integer number on a scale of 1 to $g > 1$, higher numbers being better grades. Your goal, of course, is to maximize your average grade on the *n* projects.

You have a total of $H > n$ hours in which to work on the n projects cumulatively, and you want to decide how to divide up this time. For simplicity, assume H is a positive integer, and you'll spend an integer number of hours on each project. To figure out how best divide up your time, you've come up with a set of functions $f_i : i = 1, 2, ..., n$(rough estimates, of course) for each of your n courses; if you spend $h \leq H$ hours on the project for course i, you'll get a grade of $f_i(h)$.(You may assume that the functions f_i are f_i are *nondecreasing* if $h < h'$, then $f_i(h) \leq f_i(h')$.)

So the problem is: Given these functionsf_i, decide how many hours to spend on each project(in integer values only) so that your average grade, as computed according to the f_i, is as large as possible. In order to be efficient, the running time of your algorithm should be polynomial in n, g, and H; none of these quantities should appear as an exponent in your running time.

Let $F(i, h)$ be the total grades satisfying that there are i courses and there are h hours. We can easy to get the following equation:

$$F(i, h) = \begin{cases} f_1(h), & i = 1 \\ \max\limits_{k=1}^{h-i+1}\{F(i-1, h-k) + f_i(k)\}, & i \geq 2 \end{cases} \tag{3.1}$$

Then we have the algorithm showed as 3.0.3

The time costs is roughly $O(H^3)$

Algorithm 3.0.3 Maximum grades

Input: the quantity of courses n; a series $f_1, f_2, ...f_n$ of n courses; total hours h

Initialize a upper triangular matrix $[F_{i,h}]_{i<h}$
for h from 1 to H **do**
  $F[1][h] = f_1(h)$
end for
for i from 2 to n **do**
  for h from 1 to H **do**
    Set $F(i, h)$ as the maximum value of $F(i-1, h-k) + f_i(k)$ for $k = 1$ to $h - i + 1$
  end for
end for
Return $F(n, H)$

Question 4

Suppose you are given a directed graph $G = (V, E)$ with costs on the edges c_e for $e \in E$ and a sink t(costs may be negative). Assume that you also have finite values $d(v)$ for $v \in V$. Someone claims that, for each node $v \in V$, the quantity $d(v)$ is the cost of the minimum-cost path from node v to the sink t.

1. Give a linear-time algorithm(time $O(m)$ if the graph has m if the graph has m edges) that verifies whether this claim is correct.
2. Assume that the distances are correct, and $d(v)$ is finite for all $v \in V$. Now you need to compute distances to a different sink t'. Give an $O(m\log n)$ algorithm for computing distances $d'(v)$ for all nodes $v \in V$ to the sink node t'. (*Hint*: It is useful to consider a new cost function defined as follows: for edge $e = (v, w)$, let $c'_e = c_e - d(v) + d(w)$). Is there a relation between costs of paths for the two different costs c and c'?)

1. According to Bellman-Ford algorithm, we only need to test each edge whether all of them satisfying "$d(u) \leq d(v) + w(u, v)$". If not, there is at one vertex that not records the shortest path. This algorithm only test each edge one time, so the time cost is $O(m)$.

2. Notice that if $\overrightarrow{(v, w)}$ is in a shortest path, then $c'_e = c_e - d(v) + d(w) = 0$. If $\overrightarrow{(w, v)}$ is in a shortest path, then $c'_e = c_e - d(v) + d(w) = 2 \times c_e$. If $\overrightarrow{(v, w)}$ is not in a shortest path, then $c'_e = c_e - d(v) + d(w)$ satisfying $0 < c'_e < 2c_e$.

So we have $c'_e > 0$. Besides we have that when use c'_e to get a shortest path, it is also a shortest path for c_e. Then we can use Dijkstra algorithm which costs $O(m\log n)$.