1 Suppose that $G = (V, E)$ is an undirected graph and that each edge $e \in E$ is assigned a weight $w(e)$ such that $w(e) = 2^i$ for some $i \geqslant 0$. Suppose further that no two edges in E have the same weight.

Recall that a *matching* for G is a set of edges $M \subseteq E$ such that no two edges in M share a common vertex. Consider the following greedy algorithm for finding a maximum weight matching in G, where $E = \{e_1, \ldots, e_m\}$.

```
Sort the edges so that w(e_1) ⩾ w(e_2) ⩾ ... ⩾ w(e_m).
M ← ∅    # M is the current set of edges in the matching
W ← 0    # W is the current total weight of M
for i = 1,...,m:    # Loop Invariant: ...
    if e_i does not share a vertex with any edge in M:
        M ← M ∪ {e_i}
        W ← W + w(e_i)
return M
```

Give a Loop Invariant, and use it to write a detailed proof that the output M is a maximum matching (following the proof format outlined in class).

HINT: Recall that $2^0 + 2^1 + \cdots + 2^{a-1} < 2^a$, for all $a \geqslant 0$.

Sample Solution:

Loop Invariant: M can be extended to a maximum matching M', using edges from the set $\{e_i, e_{i+1}, \ldots, e_m\}$.

Note that it follows from this Loop Invariant that the output M of the program is a maximum matching. This is because after exiting from the loop, $M = M'$, where M' is a maximum matching which extends M using eges from the set $\{e_{m+1}, \ldots, e_m\}$, which is the empty set.

Thus it suffices to prove the Loop Invariant.

To prove the Loop Invariant, we use induction on i. When $i = 1$, the loop is first entered, so $M = \emptyset$, so the extension M' can be any maximum matching, since all that is required is $M' \subseteq \{e_1, \ldots, e_m\}$.

In case $i > 1$, by the induction hypothesis there is a maximum matching M'_{i-1} which extends the current value of M.

Case 1: e_i shares a common vertex with some edge in M. Then e_i is not added to M, and $e_i \notin M'_{i-1}$ because M'_{i-1} is a matching. Therefore we can let $M' = M'_{i-1}$.

Case 2: e_i does not share a common vertex with any edge in M. Then e_i is added to M. But then we claim that $e_i \in M'_{i-1}$, so again we can let $M' = M'_{i-1}$. This claim follows from the HINT above and our assumption that all the edge weights are distinct, since these imply that the sum $w_{i+1} + w_{i+2} + \cdots + w_m$ of all remaining edge weights is less than w_i. Thus we conclude that $e_i \in M'_{i-1}$, since otherwise the current weight of M already exceeds the total weight of M'_{i-1}, so M'_{i-1} could not be a maximum matching.

Marking scheme for Question 1:

2 points for proper proof structure
4 points for good loop invariant
1 point for properly proving the base case
1 point for proving case 2a

2 points for proving case 2b
2 points for having good details and justifications

Error codes:
E1a: exchanging just a single edge may result in conflicts with other edges
E1b: loop invariant not strong enough/not useful
E1c: base case needs more justification
E1d: not enough justification
E1e: need to show that e_{i+1} is not in the optimum solutions

2 Give a dynamic program to solve the following problem:

- **Input:** Positive integers a, b, c, d, n.
- **Question:** Are there nonnegative integers (w, x, y, z) such that $n = aw + bx + cy + dz$?

Your solution should include the following steps:

(a) Define a suitable array indexed by parameters that define subproblems.
(b) Give a recurrence for the array values, based on the recursive structure of the problem, and argue that your recurrence is correct.
(c) Write an iterative algorithm, based on the recurrence, to solve the problem for arbitrary inputs a, b, c, d, n.

Hint: There is nothing to optimize—use an array that simply stores whether or not a solution exists.

Sample Solution:

Let $m = \max\{a, b, c, d\}$.

For $-m \leqslant i \leqslant n$ define $A(i) = 1$ iff $i \geqslant 0$ and $i = aw + bx + cy + dz$ for some nonnegative integers w, x, y, z. Define $A(i) = 0$ if $A(i) \neq 1$.

Thus we want to determine whether $A(n) = 1$.

Here is the recurrence:

- If $-m \leqslant i < 0$ then $A(i) = 0$
- $A(0) = 1$
- If $i > 0$ then $A(i) = 1$ iff either $A(i-a) = 1$ or $A(i-b) = 1$ or $A(i-c) = 1$ or $A(i-d) = 1$. Otherwise $A(i) = 0$.

The correctness of the recurrence can be proved by induction in i.

The base cases are $-m \leqslant i \leqslant 0$. These cases hold by our intended definition of $A(i)$.

For the induction step, assume $i > 0$. If either $A(i-a) = 1$ or $A(i-b) = 1$ or $A(i-c) = 1$ or $A(i-d) = 1$ then there exist nonnegative integers w, x, y, z so that one of $i-a, i-b, i-c, i-d$ equals $aw + bx + cy + dz$. Then either $i = a(w+1) + bx + cy + dz$ or $i = aw + b(x+1) + cy + dz$ or $i = aw + bx + c(y+1) + dz$ or $i = aw + bx + cy + d(z+1)$. Therefore $A(i) = 1$.

Conversely, if $i > 0$ and $A(i) = 1$, then $i = aw+bx+cy+dz$ for some nonnegative integers w, x, y, z. Thus $i-a = a(w-1)+bx+cy+dz$ and $i-b = aw+b(x-1)+cy+dz$ and $i-c = aw+bx+c(y-1)+dz$ and $i-d = aw + bx + cy + d(z-1)$. Since $i > 0$ it follows that either $w > 0$ or $x > 0$ or $y > 0$ or $z > 0$, so either $i-a \geqslant 0$ or $i-b \geqslant 0$ or $i-c \geqslant 0$ or $i-d \geqslant 0$. Thus either $A(i-a) = 1$ or $A(i-b) = 1$ or $A(i-c) = 1$ or $A(i-d) = 1$.

Here is the iterative program solving the recurrence:

```
for i = −m ··· − 1
    A(i) = 0
end for
A(0) = 1
for i = 1 ... n
    if A(i − a) = 1 or A(i − b) = 1 or A(i − c) = 1 or A(i − d) = 1 then A(i) = 1
    else A(i) = 0
end for
Output A(n)
```

Marking scheme for Question 2:

Array definition (2 marks)
Base case of the DP (2 marks)
DP recurrence (3 marks)
Justification of correctness (1 mark)
Pseudocode implementing the given recurrence as an iterative algorithm (2 marks)

In general there where many greedy-like or brute force approaches. Many of them got 0/10, but if there was some hint of trying to define some array or recurrence, then it was 1 or 2 marks.

Incomplete solutions showing somewhat reasonable DP structure usually got 5-6 marks.

I didn't penalize for run time, i.e. a DP that has 5 dimensions $A[w,x,y,z,s]$ = true iff "s is $w*a+x*b+y*c+z*d$" would get full score if properly presented in DP framework with recurrence of style "$A[w,x,y,z,s] = A[w-1,x,y,z,s-a]$" for each coefficient, even though its complexity is even worse than simple brute force.

I didn't down-mark for "bugs" like reaching to the array out of its bounds.

3 Suppose that we are given a flow network based on the directed graph $G = (V,E)$ with source node s and sink node t, and with integer capacities $c(u,v)$ for each edge $(u,v) \in E$. Suppose f is a maximum flow in the network, and we are given the residual network G_f. Give an efficient algorithm for finding a minimum cut in G, analyse your algorithm's running time and prove that it is correct.

Solution:

Compute the set $S = \{v \in V :$ there exists a path from s to v in $G_f\}$.

Output (S,T), where $T = V - S$.

The set S can be computed by breadth first search from G_f in time $O(|E|)$.

Hence the time required to computed the cut, given G_f, is $O(|E|)$.

Obviously S and T partition the vertex set V into disjoiont sets, and $s \in S$. There cannot be a path in G_f from s to t, since this would be an augmenting path, and hence f would not be a maximum flow. Therefore $t \in T$. Thus (S,T) is a cut.

It remains to prove that (S,T) is a minimum cut.

Consider a pair of vertices $u \in S$ and $v \in T$.

If $(u,v) \in E$ then $f(u,v) = c(u,v)$, since otherwise $(u,v) \in E_f$ which would place v in the set S.

If $(v,u) \in E$, then $f(v,u) = 0$, since otherwise $c_f(u,v) = f(v,u)$ would be positive, so $(u,v) \in E_f$, so $v \in S$.

Thus the flow $f(u,v)$ across any edge (u,v) from S to T is the edge capacity $c(u,v)$, and there is no flow from T to S. Therefore $f(S,T) = c(S,T)$.

Since for any cut (S',T') we have $f(S,T) = f(S',T') \leqslant c(S',T')$ it follows that $c(S,T) \leqslant c(S',T')$. Thus (S,T) is a minimum cut.

Marking scheme for Question 3

The question is out of 8 marks: 5 for the idea of doing a BFS or DFS and separating the reachable from non-reachable nodes. 3 for the proof where 2 goes to arguing that the what they construct is an actual cut with capacity = flow, and 1 for arguing that it is a minimum cut.

If all they have correct is the idea of a cut, i.e. separating s from t, they get 1 mark. If they mention the max flow min cut theorem (MFMC) they get 1 to 2 marks depending on how they use it. If they claim that all they need is the theorem they get 1 mark, if they show they need to actually construct a cut, and not just state the theorem they get 2 marks.

If they just state they need to run a BFS on the residual graph and nothing else, they get 5 marks to 5.5 depending again on the level of details. If they give some sort of (incomplete) proof they get up to 6.5.

This is all. In general, most students got the idea, i.e. most of them got around 5.5-6.5.