A5Q1

(a)

u	v	Path	b(P)
a	b	aceb	4
a	c	ac	4
a	d	acd	4
a	e	ace	4
b	c	bec	5
b	d	becd	5
b	e	be	7
c	d	cd	6
c	e	ce	5
d	e	dce	5

(b)

Blackened edges form a maximum spanning tree. We modify Prim's algorithm:

```
prim(G(V, E) and edge_weights):
     choose any s in V
     let L={s} and R=V-{s}
     Let MWE be a priority queue, containing
          forall v in R, weight of the max weight edge
          (u, v) connecting v to L (and (u, v) as metadata)
     T = {}
     for i = 1 to n:
          (v, (v, u)) = MWE.extractMax()         # O(log n)
          remove v from R; add v to L
          add (u, v) to T
          for each (v, z) st z in R:             # O(m*(log n))
               if w(v, z) > z's current cost in MWE:
                    incrementKey in MWE
```

```
                  and keep(v, z) as metadata

    return T

// m = # edges
```

Alternate approach: Run a pre-processing step to negate weights of all edges. Resulting tree would be maximum spanning tree for original input.

Runtime
`O(n log n + m log n) = O(m log n) as m > n`

Correctness
Prove Prim outputs a spanning tree
Proof: At each iteration i, $T_{prim,i}$ spans L_i AND is acyclic.
By induction: Holds for base case s
Ind Hyp. $T_{prim,k}$ spans L_k and is acyclic.
Prim picks the max edge (u, v) from $L_k to\ R_k$.
By IH: u has a path to every vertex in $L_k$and vice versa
=> with addition of (u, v), v has a path to every vertex L_k and vice versa
=> $T_{prim,k+1}$ is spanning.
$T_{prim,k+1}$ is acyclic because (u, v) is the first edge from L_k to R_k (by the lonely edge corollary).

Prove T_{prim} is a MST
Using MST Cut Property:
Prim looks at n-1 different cuts. Add n-1 edges that every MST includes.
Since any spanning tree includes n-1 edges, T_{prim} is the MST
(with distinct edge weights this proof also proves the uniqueness of MST).

(c)

Assumption: Graph remains connected after removing edge e.
Case 1. e was not in MST. We don't need to do anything here.
Case 2. e was in MST. For the two vertices connected by u and v, we can find another edge with max weight that connects u and v which does not yield a cycle. We simply need to loop over all edges to find the new edge.

Runtime
O(|E|) as it simply involves looping over all edges.

Correctness
Case 1. We already have the optimal MST solution of which e is not a part. Removing e would not affect our solution.
Case 2. Since we choose the max weight edge between u and v, we should end up with an MST.

A5Q2

(a)

1. Construct a new graph G with vertices V and edges from all paths.
2. Use Dijkstra's algorithm on G with given s and t to find the shortest path.

(b)

Memory

- $P[i,j]$: length of shortest single s-t path that exists in graphs $G_i \dots G_j$.
- $C[i,j]$: cost of shortest path for graphs $G_i \dots G_j$ with the last switch of path at G_j and G_{j+1}.

Base Case

- $P[i,j] = length\big(partA(i,j)\big) \quad \# \, store \, None \, if \, path \, doesn't \, exist$
- $C[1..k,0] = P[1,1..k]$

Recurrence

$C[i,j] = \min(C[0..j,j] + (i-j) * P[j+1,i]) + \lambda \;\; for \; switch$
for i > j

Complexity

- To generate table P: two nested loops with Dijkstra's $O(E \log V)$ complexity. Total: $O(k^2 * O(E \log V))$
- Recurrence: $O(k^3)$
- Total complexity: $O(k^2 * O(E \log V) + k^3)$

Correctness

If a switch occurs between i and i+1, we know G_{i+1} to G_k are the same path. For < i, we solve the subproblem. We add λ for each switch.

And thus we're able to find the sequence of paths with minimal cost.

A5Q3

(a)

MAXCUT-DECISION instance
An undirected graph G = (V, E) where each edge e = uv has a positive integer weight w(uv), and a target T.

Return YES if a partition V_1 and V_2 of V exists with sum of all w(uv) where u is in V_1 and v is in $V_2 \geq T$.

MAXCUT-DECISION is in NP
Looping through all edges is complexity O(E). Therefore there can be a way to verify the certificate (solution) in polynomial time.

```
verify-MCD-cert(V1, V2, T):
  sum = 0
  for u in V1:
    for v in V2:
      sum += w(uv)

  if sum >= T:
    return YES
  else:
    return NO
```

Therefore MAXCUT-DECISION is in NP.

(b)

We can find largest T (in range) in polynomial time by doing binary search using MAXCUT-DECISION and different values of T (given solution for MAXCUT-DECISION can be found in polynomial time). Given a T, we can find partitions V_1 and V_2 in polynomial time. For every edge in G, we remove it and pass G through MAXCUT-DECISION to see if the result changes. If it doesn't, the edge was not a part of the partition. If it does, the edge was a part of the partition and we put the vertices in different sets.
Therefore MAXCUT can be solved in polynomial time if MAXCUT-DECISION can be.

A5Q4

(a)

Meeting Scheduling is NP since the certificate (solution) can be verified in polynomial time to see if it satisfies the problem.

Following is a reduction from NAE-3SAT to Meeting Scheduling to show it is NP-complete (NAE-3SAT is NP-complete). Let S be the family of subsets $S_1, \dots, S_l$.

NAE-3SAT consists of m clauses $\{C_1, \dots, C_m\}$ over n variables $\{x_1, \dots, x_n\}$. We create our instance (S, M, K_1, K_2). (We only consider two values for $K = \{1, 2\}$ as it suffices.)

Let:
$M = \{x_1, \dots, x_n, \sim x_1, \dots, \sim x_n\}$ # All variables and their negations. ~ denotes a negation
S consist of two sub-collections:

1. $\{\{x_1, \sim x_1\}, \dots, \{x_n, \sim x_n\}\}$ # Collection of sets of all variables each with their negation
2. {{Literals in C_1}, …, {Literals in C_m}} # Collection of literals from every clause

M can be split in the required way iff the NAE-3SAT problem has an assignment that makes it yield true.

There exists a partition of M into two subsets K_1, K_2 such that all $S_1, \dots, S_l$ are split by this partition. Let K_1 correspond to variables in NAE-3SAT problem that are $true$ and K_2 to ones that are $false$.

Therefore the Meeting Scheduling problem is NP-complete.

(b)

Since we reduced NAE-3SAT to Meeting Scheduling in Part 1, we can reduce 3SAT to NAE-3SAT to show Meeting Scheduling is NP-complete (3SAT is NP-complete).

3SAT consists of m clauses $\{C_1, \ldots, C_m\}$. For each clause $(x_i \text{ } or \text{ } x_j \text{ } or \text{ } x_k)$, have a variable c_l and add two clauses to our instance as follows:

$(x_i \text{ } or \text{ } x_j \text{ } or \text{ } c_l) \text{ } and \text{ } (x_k \text{ } or \sim c_l \text{ } or \text{ } F)$
where F = False

If $x_i \text{ } or \text{ } x_j$ are true, then c_l can be false and $\sim c_l$ will make the second clause true.
If x_k was true, then c_l can be true.
If all of x_i, x_j, x_k, are true, then c_l can be false.

To replace constant F with a variable, define x_T and x_F and add another clause:
$(x_T \text{ } or \text{ } x_T \text{ } or \text{ } x_F)$ where one of $\{x_T, \ x_F\} = F$ and $x_T \neq x_F$.

This yields an acceptable NAE-3SAT solution and therefore NAE-3SAT is NP-complete.
Thus, Meeting Schedule is NP-complete.