I shall not provide you with the complete answers, just with hints:

A1. You have to compute which is the sum between any function g(n) <= c*f(n) and any function c1 * f(n) <= h(n) <= c2 * f(n). By summing up, you obtain:

(c+c1)*f(n)=<g(n) + h(n)<=c2*f(n)

This means that you have found two new constants c3 = c+c1 and c4=c2 so the result should be theta(f(n))

A2. Backtracking is inefficient because it tests all possible partial solutions until it finds a contradiction with previous assignments. It must be used to solve NP-hard problems or when one wants to find all the possible solutions to a given problem. The 3 heuristics are listed in the slides.

A3. Because you need to be sure you choose these roots in the topological order of G_SCC, meaning the reverse topological order of transpose(G_SCC). Therefore, no edge will go from the first SCC to the next ones and so on... More information in the slides.

A4. There are 3 methods explained in the class: arrays, binary heaps and Fibonacci heaps. Each has a different complexity for removing an element and modifying an element in the priority queue, thus influencing the overall complexity of Prim's algorithm.

A5. There is a theorem for that, but it's not for this year's exam. Starting from the theorem you have to find a simple method/algorithm.

B1a. Use master theorem directly, case 3
B1b. Guess that the solution is O(n) and prove it using induction (or the substitution method)
B1c. Replace n = 2^m and see what happens. Solve the new recurrence in m.

B2a. You can have various DFS traversals. One is the following:

node	d	f
1	1	16
4	2	13
7	3	12
8	4	11
6	5	8
3	6	7
5	9	10
2	14	15

B2b.
Q = {5}
d = [INF INF INF INF 0 INF INF INF] // nodes order is 1,2,...,8

Q = {1}
d = [2 INF INF INF 0 INF INF INF] // nodes order is 1,2,...,8

Q = {2, 4}
d = [2 7 INF 5 0 INF INF INF] // nodes order is 1,2,...,8

Q = {2, 7, 8}
d = [2 7 INF 5 0 INF 6 7] // nodes order is 1,2,...,8

(you can continue until all nodes are computed)

B2c.
D(0) =
0 3 INF 5 INF INF INF INF
INF 0 INF INF 2 3 INF INF
INF INF 0 INF INF 3 INF INF
INF INF INF 0 INF INF 2 1
2 INF INF INF 0 INF INF INF
INF INF 1 INF INF INF 0 INF
INF INF INF INF INF INF 0 1
INF INF INF INF 2 3 INF 0

D(1) =
0 3 INF 5 INF INF INF INF
INF 0 INF INF 2 3 INF INF
INF INF 0 INF INF 3 INF INF
INF INF INF 0 INF INF 2 1
2 7 INF 5 0 INF INF INF
INF INF 1 INF INF INF 0 INF
INF INF INF INF INF INF 0 1
INF INF INF INF 2 3 INF 0

(only elements D(1)[5][2] and D(1)[5][4] are changed from the previous matrix)

(you continue for D(2) and D(3))

C. There are several solutions, but the best one is to devise a dynamic programming solution by taking into consideration that at each step you need to choose between two directions (coming from up or coming from left). Therefore the recurrence is:

best[i][j] = A[i][j] + max(best[i-1][j], best[i][j-1]) if i > 1 and j > 1
A[i][j] + best[i][j-1] if i == 1 and j > 1
A[i][j] + best[i-1][j] if i > 1 and j == 1
A[i][j] if i==1 and j==1