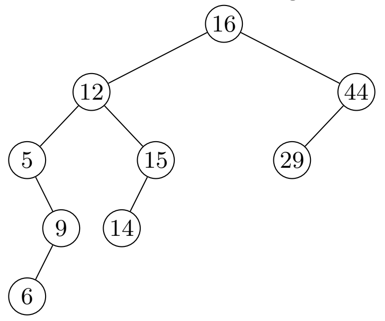
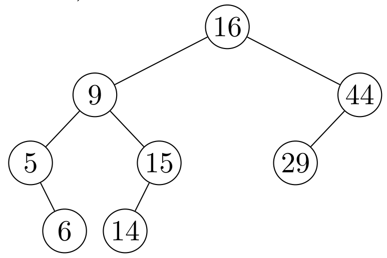
Test 2 Solutions

Question 1 [12 marks]
(a) Consider the following BST:



(2 marks) Delete key 12. **Answer:**



Alternatively, one can also use key 14 to replace key 12.

(2 marks) Is the original BST given above an AVL tree? If not, perform tree rotation(s) to make it AVL.

Answer: No, it is not AVL. Perform a double rotation on the subtree rooted at 5, which results in the subtree rooted at node 6 whose left child is 5 and right child is 9.

Marking note: The answer relies on a *double rotation*. But the class discussion and exercises are all about *single rotation*. We also said that students don't need to remember all rules of tree rotation. We therefore adopt the following marking scheme: An answer that says the given tree is not AVL gets 2 marks. Any correct rotations that result in an AVL-tree gets 1 bonus mark.

(b) (8 marks) Circle those statements that are true.

F (1) It is possible to develop a comparison-based sort algorithm that runs in $O(n \log\log n)$ time.

T (2) Given an AVL tree T of n nodes, the task of searching if a key is in T takes $O(\log n)$ time.

F (3) If a problem exhibits *optimal substructure*, then a greedy algorithm exists.

T (4) In a greedy algorithm, the choice at each step does not depend on how the future choices are made.

F (5) If a problem can be solved by a recursive algorithm, then a DP (dynamic programming) solution is guaranteed to exist so that running time can be improved.

T (6) There are problems that cannot be solved by the greedy method but can be solved by DP.

F (7) The DP solution to the Matrix-Chain Multiplication problem applies the divide-and-conquer strategy: split the given matrix-chain to two halves and solve them recursively.

T or F (8) In a hash table in which collisions are resolved by chaining, suppose the hash function is *simple uniform* - any element is equally likely to hash into any of the available slots. Also assume the hash function can be computed in $O(1)$ time. Then, the running time for searching an element is $O(1)$ on average.

Remark: For the statement to be true, there is acutually one more assumption: $n \in O(m)$, the number of keys is propositional in the number of buckets. In this case $1 + \alpha = 1 + n/m$, which is bounded by a constant. For this reason, you get a mark with either answer.

Question 2 [12 marks]
(a) (6 marks) You are given the following adjacency list representation of a digraph G.

vertex $\rightarrow$	out-neighbors
$A \rightarrow$	F, D
$B \rightarrow$	C, A
$C \rightarrow$	A
$D \rightarrow$	B
$E \rightarrow$	C
$F \rightarrow$	D, E

Draw the BFS tree starting at node A by following this adjacency list model. Then, for each non-tree edge, draw a directed, dotted line on the tree and indicate which type of non-tree edge it is (recall there are three types: *back edge*, *cross edge*, and *forward edge*).

Answer: Your drawing should correspond to the following edges and their types:

- tree edges: $(A, F), (A, D), (F, E), (D, B), (E, C)$
- cross edges: $(F, D), (B, C)$
- back edges: $(B, A), (C, A)$
- forward edges: none

(b) (6 marks) This is a question regarding "cup dropping" from Exercise #5: You have a manufacturing company who produces some form of glass that is more resistant than a typical glass. In your quality control section you do testing of samples and the setup for a particular type of cup produced is as follows. You have a ladder with n steps and you want to find the highest step from which you can drop a sample cup and not have it break. We call this the highest safe step.

In this question, you are asked to describe an algorithm to find the highest safe step while allowing to break **at most 3 cups.** Your algorithm should be asymptotically faster than the one which runs in $O(\sqrt{n})$ time and allows to break at most 2 cups.

Hint: For partial marks, you may describe an algorithm that allows to break at most 2 cups, whose running time is $O(\sqrt{n})$.

For part marks (at most 3 marks): If you don't know how to solve the given problem, as the hint says, you can get part marks by giving an algorithm with runtime $O(\sqrt{n})$ that breaks at most 2 cups. Such an algorithm is given in Exercise # 5. Recall that in the worst case we do $n/\ell + \ell$ many drops and the best solution is let $\ell = \sqrt{n}$ (e.g., if you let $\ell = n^{0.4}$, then $n/\ell + \ell = n^{0.6} + n^{0.4} \in O(n^{0.6})$, which is worse than $O(\sqrt{n})$.)

Answer to the given problem: We assume a number ℓ and try dropping a cup from step ℓ, then 2ℓ, and so on (up to n/ℓ) until the first value of i for which dropping from $i\ell$ does not break but it will break from $(i+1)\ell$. The highest safe step is between $i\ell$ to $(i+1)\ell - 1$. We treat the range ℓ as the subproblem, to which we apply the same algorithm that allows to break up at most 2 cups. Assume another interval ℓ'. Then, in the worst case we need to do $n/\ell + \ell/\ell' + \ell'$ many drops. We can let $\ell = n^{0.6}$ and $\ell' = n^{0.3}$, then $n/\ell + \ell/\ell' + \ell' = n^{0.4} + n^{0.3} + n^{0.3} \in O(n^{0.4})$, which is asymptotically faster than $\sqrt{n}$.

Question 3 [12 marks]

Recall the *Activity Selection Problem*: Given a set of n activities $S = \{a_1, a_2, \ldots, a_n\}$ where each a_i has a *start time* s_i and a *finish time* f_i $(0 \le s_i < f_i < \infty)$ (if selected, a_i occupies the time interval $[s_i, f_i)$), our goal is to select a maximum-size set of mutually compatible (non-overlapping) activities.

(a) (6 marks) Instead of selecting the first activity to finish, let us select **the last activity to start** that is compatible with all previously selected activities. Describe a greedy algorithm based on this strategy. In the following, we provide a skeleton. You just need to complete it. It is fine if you write your own algorithm from scratch.

```
procedure Activity-Selection2 ($S$)
   Sort activities in $S$ such that $s_1 \ge s_2 \ge \ldots \ge s_n$
   $A \leftarrow \emptyset$;  $e \leftarrow \infty$;
   for $i \leftarrow 1$ to $n$ do
      if $f_i \le e$ then
         $A \leftarrow A \cup \{i\}$; $e \leftarrow s_i$
   return $A$
```

(b) (6 marks) Answer the following questions regarding the optimality of your algorithm.

- Let $Q = (q_1, q_2, \ldots, q_m)$ $(1 \le m \le n)$ be an optimal solution to the problem (ordered in non-increasing start times) and $J = (j_1, j_2, \ldots, j_k)$ $(1 \le k \le n)$ be the output of your algorithm. Argue why $Q' = (j_1, q_2, \ldots, q_m)$ is an optimal solution.

  Answer: By the nature of our algorithm, the first selected activity j_1 always has the latest start time. If $Q = (q_1, q_2, \ldots, q_m)$ $(1 \le m \le n)$ is an optimal solution to the problem, observe that the start time of q_1 is no latter than the start time of j_1, therefore, replacing q_1 with j_1 gives us another feasible solution with size $|Q|$, i.e, another optimal solution.

- Suppose by induction you have shown that all greedy choices by your algorithm are included in an optimal solution and you want to argue that $m = k$. That is, let $Q = (j_1, j_2, \ldots, j_k, q_{k+1}, \ldots, q_m)$ be an optimal solution and $J = (j_1, j_2, \ldots, j_k)$ be the output of your algorithm, where $m > k$. Why is this case not possible?

  Answer: This case is not possible, again, by the nature of our algorithm. For the sake of contradition, suppose this is the case. Since $Q = (j_1, j_2, \ldots, j_k, q_{k+1}, \ldots, q_m)$ is an optimal solution, q_{k+1} does not overlap with any of $j_1, j_2, \ldots, j_k$; therefore our algorithm will include at least q_{k+1}. This is a contradiction to the assumption that our algorithm terminates and produces J.

Question 4 [14 marks]

(a) (6 marks) Consider the longest common sub-sequence (LCS) problem: Given two sequences, $x_1x_2 \ldots x_n$ and $y_1y_2 \ldots y_m$, find a maximum-length common sub-sequence of them.

E.g., *dog* is a common sub-sequence of *dynamicprogram* and *dough*. The following is a recursive definition for solving the problem:

$$\begin{aligned} LCS(x_1...x_n; y_1...y_m) = \max\{ & LCS(x_1...x_{n-1}; y_1...y_m), \\ & LCS(x_1...x_n; y_1...y_{m-1}), \\ & (\text{if } x_n = y_m)\ 1 + LCS(x_1...x_{n-1}; y_1...y_{m-1})\} \end{aligned}$$

The bases cases are: if $n = 0$ or $m = 0$, then LCS yields 0 as the result.

You are asked to solve the problem by DP. For this, you should define a table $D[i, j]$ $(0 \le i \le n, 0 \le j \le m)$, and describe how to fill its entries (this description is considered your algorithm).

Answer:

$$D[i,j] = \max \begin{cases} D[i-1,j], \\ D[i,j-1], \\ D[i-1,j-1]+1, & \text{if } x_i = y_j \end{cases}$$

And the base cases are $D[i, 0] = D[0, j] = 0$ for any i, j.

We fill the table bottom-up, i.e., we do the base cases first; and then use the above definition from right to left and fill the table row-by-row and from left to right; this is because the value of $D[i, j]$ only depends on those of $D[i-1, j], D[i, j-1], D[i-1, j-1]$.

(b) (6 marks)

Given sequences, `DFCG` and `CDEFG`, we then have a table $D[0..4; 0..5]$, as drawn below. You are asked to fill the table based on your algorithm.

	C	D	E	F	G
	0 0	↖0	0	0	0
D	0 0	1	1	↖1	1
F	0 ↖0	1	1	2	2
C	0 1	1	1	2	↖2
G	0 1	1	1	1	3

			C	D	E	F	G
	i\j	0	1	2	3	4	5
	0	0	0	0	0	0	0
D	1	0	0	↖ 1	← 1	1	1
F	2	0	0	1	1	↖ 2	2
C	3	0	1	1	1	↑ 2	2
G	4	0	1	1	1	2	↖ 3

(c) (2 marks) For the problem instance above and by the way you filled the table, indicate how the longest common sub-sequence is identified from your table.

Answer: Any reasonable way to indicate will be accepted, e.g., you can draw arrows like the above. Or, you just explain: each time we apply the case $D[i-1, j-1] + 1$ (when $x_i = y_j$), we identify one common character. The last time this is G when we are filling $D[4, 5]$, before that it is F when filling $D[2, 4]$, and the first time it is D when filling $D[1, 2]$. Thus, DFG is a LCS.