Clearly indicate the following spanning trees for the weighted graph in the problem sheets.

(a) (2.5 out of 10) A depth-first spanning tree rooted at $s$.

Solution:

(b) (2.5 out of 10) A breadth-first spanning tree rooted at $s$.

Solution:
(c) **(2.5 out of 10)** A shortest path tree rooted at $s$.

Solution:

(d) **(2.5 out of 10)** A minimum spanning tree.

Solution:
(a) **(4 out of 10)** The following greedy strategy *does not* lead to an optimal set of assignments.

Let skier $i$ and ski $j$ have the smallest height difference between any ski and skier. Assign ski $j$ to skier $i$ and recursively find the best assignment for the remaining skis and skiers.

Describe two small arrays $P[1..n]$ and $S[1..n]$ where the proposed strategy does not lead to an optimal set of assignments.

**Solution:** Let $P[1..2]$ contain 5 and 2 and let $S[1..2]$ contain 3 and 0. The greedy strategy assigns the 2 skier to the 3 ski and then the 5 skier to the 0 ski for a total cost of $1 + 5 = 6$. The only other assignment has cost $2 + 2 = 4$.

(b) **(6 out of 10)** The following greedy strategy *does* lead to an optimal set of assignments.

Assign the lowest ski to the lowest skier and recursively find the best assignment for the remaining skis and skiers.

**Prove** that the proposed strategy leads to an optimal set of assignments.

**Solution:** We first prove there is an optimal solution assigning the lowest ski to the lowest skier. Let $A$ be any assignment and let the lowest ski and skier have heights $h_1$ and $g_1$ respectively. Assume, without loss of generality, that $h_1 \leq g_1$. If $A$ assigns the lowest ski to the lowest skier, we are done. Otherwise, it assigns lowest ski to a skier of height $g_2 \geq g_1$ and then assigns a ski of height $h_2 \geq h_1$ to the lowest skier. If we swap the assignments for those two skis and skiers, the cost increases by

$$
(|g_1 - h_1| - |g_2 - h_1|) + (|g_2 - h_2| - |g_1 - h_2|) = (g_1 - g_2) + (|g_2 - h_2| - |g_1 - h_2|) \\
\leq (g_1 - g_2) + (g_2 - g_1) \\
= 0.
$$

So the cost does not increase and our new assignment is still optimal.

Therefore, the greedy strategy does an initial assignment that belong to an optimal solution and then, by induction, does an optimal assignment for the remaining skis and skiers.
1. *(5 out of 10)* Suppose we implement Dijkstra using a priority queue that performs INSERT in $\alpha$ time units, EXTRACTMIN in $\beta$ time units, and DECREASEKEY in $\gamma$ time units. What is the total running time of Dijkstra($s$) when run on a directed graph $G = (V,E)$ with non-negative edge lengths? Briefly justify your answer.

**Solution:** Dijkstra inserts and extracts each vertex at most once and relaxes each edge at most once in this setting. Therefore, the total running time is $O(\gamma E + (\alpha + \beta)V)$. ■

2. *(5 out of 10)* Suppose you are given an edge-weighted directed graph $G = (V,E)$ where edge weights could be positive, zero, or negative along with a source vertex $s \in V$ and target vertex $t \in V$. You are guaranteed two facts about $G$:
   - There are no negative cycles, and
   - the shortest path from $s$ to $t$ uses at most $k$ edges.

Using big-Oh notation in terms of one or more of $k$, $V$, and $E$, how quickly can you compute the shortest path from $s$ to $t$? You must briefly justify your answer, but you do not need to give complete details on the relevant algorithm.

**Solution:** We can compute the shortest path in $O(V + kE)$ time. ($O(kE)$ is good enough for full credit.) To do so, we simply run $k$ iterations of Bellman-Ford in $O(E)$ time per iteration. After those $k$ iterations, $\text{dist}(t)$ will be at most the length of the shortest walk from $s$ to $t$ with at most $k$ edges. But the second bullet above, though, such a walk is the overall shortest path. ■
Suppose are you given a set $S$ of $n$ points in the plane, represented as two arrays $X[1..n]$ and $Y[1..n]$. Describe and analyze an algorithm to compute the length of the longest monotonically increasing path with vertices in $S$. Assume you have a subroutine $\text{LENGTH}(x, y, x', y')$ that returns the length of the segment from $(x, y)$ to $(x', y')$. You do not need to justify correctness of your algorithm (time is short), but you do need to explain your running time analysis.

Solution: There are a couple ways to solve this problem.

- We could reduce to single source shortest paths in a DAG. Create a graph $G = (V, E)$ where $V$ is the set of input vertices $S$ and there is an edge $u \rightarrow v$ for every pair of vertices where $v$ is above and to the right of $u$. The graph is a DAG, because any cycle would have to have an edges going back to the left. Weight each edge $(x, y) \rightarrow (x', y')$ as $-\text{LENGTH}(x, y, x', y')$. Now any path in $G$ is a monotonically increasing path by construction and the longest such path in the plane is a shortest path in $G$. We compute single source shortest path trees from every vertex in $G$ in $O(V + E) = O(n^2)$ time each and return the shortest path found (in $G$). Computing all $n$ shortest path trees takes $O(n^3)$ time total.

- We also saw how to directly compute the longest path between any pair of vertices in a DAG. We could build the same graph as above but without negating the edge lengths and then run this longest path procedure $O(n^2)$ times to find the longest of the longest paths. This would take $O(n^4)$ time total. However, the dynamic programming formulation for that problem actually solved all longest paths to a given vertex $t$, so we could try that formulation over all choices of $t$ in $O(n^3)$ time total.

- Finally, we could directly do dynamic programming. Let $\text{longest}(x, y)$ be the length of the longest monotone path from vertex $(x, y) \in S$. If the path has any edges at all, it has to go to some vertex above and to the right of $(x, y)$ and then contain a longest path from that vertex. We get the following recursive definition:

$$\text{longest}(x, y) = \max_{(x', y') | x' > x, y' > y} [\text{LENGTH}(x, y, x', y') + \text{longest}(x', y')]$$

Here, the max returns 0 if there are no suitable choices of $(x', y')$. The answer we seek is the largest of all $\text{longest}(x, y)$ values.

Each subproblem $(x, y)$ depends upon subproblems for vertices above and to the right of $(x, y)$, so we can solve subproblems in decreasing sorted order by (say) $x$-coordinate. There are exactly $n$ subproblems and each depends upon the up to $n - 1$ other vertices in $S$. So we can solve all subproblems and find our answer in $O(n^2)$ time total.