Downhill Skiing

Question 1 – Greedy Algorithm

Greedy algorithm fails on:

1 (Start)	9	9
1	2	9
8	9	1

The algorithm will pick (1,2) first, as 1 < 2. It will then have the choice of either 8 (1,3) or 9 (2,3). 8 is the smallest, so the algorithm will complete with {(1,1), (1,2), (1,3)}, a total danger of 1 + 1 + 8 = 10. This is highlighted with a red background. The optimal path (green) is {(1,1), (2,2), (3,3)}, giving a danger of only 1 + 2 + 1 = 4.

Question 2 - Recurrence

Recurrence for D(x,y) =

{ ∞ if x < 1 or x > n or y < 1 or y > n

d(x,y) if y = n

D(x-1,y+1) + d(x,y)* if D(x-1,y+1) < D(x,y+1) and D(x-1,y+1) < D(x+1,y+1)

D(x+1,y+1) + d(x,y)* if D(x+1,y+1) < D(x,y+1) and D(x+1,y+1) < D(x-1,y+1)

D(x,y+1) + d(x,y)* otherwise

}

(Where d(x,y) denotes the danger of position (x,y))

(*Alternatively: D(x,y) = d(x,y) + min[D(x-1,y+1), D(x+1,y+1), D(x,y+1)] otherwise)

My recurrence is correct as from the start vertex it compares which of the accessible moves are the lowest danger. If one of these vertices is outside of the bounds of the map, their value is infinity, and thus that vertex will never be chosen by the recurrence. Therefore, the recurrence will be called recursively on the accessible vertices, until reaching the base case where y = n. This base case gives its own value, without any further calculations needed. The least dangerous path will then be built as the values return upwards.

Question 3 – Memoized Recursive Algorithm

```
D(M,x,y) {
      if x < 1 || x > M.n || y < 1 || y > M.n
            return ∞
      if y = M.n
            return M.d(x,y)
      if M.D(x,y) undefined
            M.D(x,y) <- Min(D(M,x-1,y+1), D(M,x+1,y+1), D(M,x,y+1))
+ M.d(x,y)
      return M.D(x,y)
}
```

(M.d(x,y) = danger of position (x,y); M.n = size of M; M.D(x,y) = stored result of D(M, x, y))

My memoized algorithm runs in $O(n^2)$ because it only ever calculates D(M, x, y) once during the operation of the algorithm. The first time D is called on a given vertex, the returned value is stored in

the matrix M.D. The algorithm runs as per the recurrence specified previously, but now performs far less calculations of D (without memoization, the algorithm would have an exponential running time).

Question 4 – Bottom-Up Algorithm

The memoized algorithm will end up calculating D for the vertices starting at y = n, ending with y = 0. This is by virtue of the fact that the only vertices that can be calculated are those where the 3 vertices directly below have already got values of D. Initially, the only vertices with known values are those out of bounds, and those at the very bottom, where y = n. Thus, the following bottom-up algorithm arises:

```
D(M,x,y) {
      if x < 1 || x > M.n || y < 1 || y > M.n
            Error("Start out of bounds.")
      for c <- M.n to 1
            M.D(c,M.n) <- M.d(c,M.n)
      for r <- M.n to 1
            for c <- M.n - 1 to 1
                  M.D(c,r) <- Min(M.D(c-1,r+1), M.D(c+1,r+1),
M.D(c,r+1)) + M.d(c,r)
      return M.D(x,y)
}
```

Question 5 – Output Algorithm

```
D(M,x,y) {
      if x < 1 || x > M.n || y < 1 || y > M.n
            Error("Start out of bounds.")
      for c <- M.n to 1
            M.D(c,M.n) <- M.d(c,M.n)
      for r <- M.n to 1
            for c <- M.n -1 to 1
                  ((pX,pY), pD) <- Min(M.D(c-1,r+1), M.D(c+1,r+1),
M.D(c,r+1))
                  M.D(c,r) <- pD + M.d(c,r)
                  M.P(c,r) <- (pX,pY)
      while y <= M.n
            Output(M.P(x,y))
            x,y <- M.P(x,y)
}
```

I have modified the algorithm to output the path taken by modifying the supporting functions. Min() now returns a tuple containing both the minimum distance from the supplied arguments, as before, along with the x and y coordinates of the 'predecessor'. M.D is updated with the distance as usual, while the predecessor position is stored in M.P(x,y). The Output function prints the supplied coordinates. It is used in a loop at the end that traces the recorded predecessors to the bottom of the world. This print loop will run n times, so the algorithmic time complexity remains $O(n^2)$.

Question 6 – Partner Skiing

To tackle the two skier problem, we need to consider the best choice for skier A, for each possible choice of skier B. I have used the previous solution as a base, extending the main loop.

```
DD(M,ax,bx) {
      if ax < 1 || ax > M.n || bx < 1 || bx > M.n
            Error("Start out of bounds.")
      for a <- 1 to M.n -1
            for b <- 2 to M.n
                  M.D(a,b,M.n) <- M.d(a,M.n) + M.d(b,M.n)
      for y <- M.n to 1
            for a <- 1 to M.n – 1
                  for b <- a + 1 to M.n
                        M.D(a,b,y) <- Min(
                                          M.D(a,b,y+1),
                                          M.D(a,b-1,y+1),
```

```
                                                    M.D(a,b+1,y+1),
                                                    M.D(a-1,b,y+1),
                                                    M.D(a-1,b-1,y+1),
                                                    M.D(a-1,b+1,y+1),
                                                    M.D(a+1,b,y+1),
                                                    M.D(a+1,b-1,y+1),
                                                    M.D(a+1,b+1,y+1)
                                                    )
                                                + M.d(a,M.n) + M.d(b,M.n)
            return M.D(ax,bx,1)
        }
```

The algorithm sets the minimal danger of each point pair (a,y) and (b,y) to the sum of the minimum danger of the possible child vertices and the danger of the pairs. As the algorithm works up from the bottom row, the total safest paths are updated in M.D. This algorithm is $O(n^3)$, as the main loop checks every combination of x co-ordinates for every y co-ordinate.