Ans 11.2

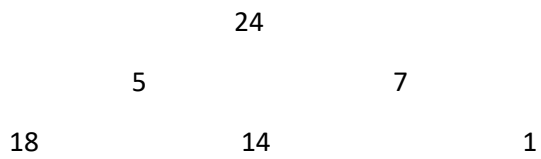
Ans b.

In the brute force approach, we would have generated all the possible paths, added the numbers along to path and compared the sums to each other to find the path which leads to the greatest sum. Since the number of paths from the peak to the triangle to the base is 2^n and since we generate all those paths and since that would be the highest complexity in the cost of the function, the overall complexity would have been $\theta(2^n)$. However, in the program written in sum_triangles.py, the highest complexity arises from the nested for loop which leads the overall complexity to be $\theta(n^2)$(since the inner loop repeats 1, 2, 3, 4……n times, and $\Sigma_{x=1}^{n} x = \frac{n(n+1)}{2}$). Thus, the runtime of the written program is significantly smaller than that of the brute force approach.

Ans c.

If a greedy algorithm is used, then it will always make the locally optimized decision. In the context of the sum in triangles this means that when we move from the peak of the triangle to the base, whenever a there are two possible paths, the algorithm will decide to go in a path that has the greater number. While this choice is locally optimum, it can also lead the program to leaving out those paths that have greater numbers, and potentially a greater sum along the path. Thus, a greedy algorithm doesn't deliver a globally optimum solution in this problem.

For instance: consider the following example



In this example, if a greedy algorithm is adopted, the solution produced would be 24 + 7 + 14 = 45, however, the correct and globally optimum solution would be 24 + 5 + 18 = 47