# [What is the difference between memoization and dynamic programming?](https://stackoverflow.com/questions/6184869/what-is-the-difference-between-memoization-and-dynamic-programming)

# ------------------------------ short answer -----------------------------

**Dynamic programming** is technique for solving problems of recursive nature by store the previouse Computation results in a table. so Dynamic Programming is mainly an optimization of plain [recursion](https://www.geeksforgeeks.org/recursion/).

This problem can be solved by using 2 approaches

1. Memoization (Top Down) - Using recursion to solve the sub-problem and storing the result in some hash table.
2. Tabulation (Bottom Up) - Using Iterative approach to solve the problem by solving the smaller sub-problems first and then using it during the execution of bigger problem.

**How apply dynamic programming? (another answer)**

1. Find the recursion in the problem.
2. Top-down: store the answer for each subproblem in a table to avoid having to recompute them.
3. Bottom-up: Find the right order to evaluate the results so that partial results are available when needed.

# ------------------------------ detailed answer -----------------------------

***What is difference between memoization and dynamic programming?***

While a Greedy Algorithm is usually called naive, because it may run multiple times over the same set of data, Dynamic Programming avoids this pitfall through a deeper understanding of the partial results that must be stored to help build the final solution.

**Dynamic programming** is technique for solving problems of recursive nature.,so Dynamic Programming is mainly an optimization of plain [recursion](https://www.geeksforgeeks.org/recursion/)

a and Dynamic programming is *typically* implemented using tabulation, but can also be implemented using memoization. So as you can see, neither one is a "subset" of the other.

**Memoization** is a term describing an optimization technique where you cache previously computed results, and return the cached result when the same computation is needed again.

A reasonable follow-up question is: **What is the difference between tabulation (the typical dynamic programming technique) and memoization?**

When you solve a dynamic programming problem using tabulation you solve the problem "**bottom up**", i.e., by solving all related sub-problems first, typically by filling up an *n*-dimensional table. Based on the results in the table, the solution to the "top" / original problem is then computed.

If you use memoization to solve the problem you do it by maintaining a map of already solved sub problems. You do it "**top down**" in the sense that you solve the "top" problem first (which typically recurses down to solve the sub-problems).

A good slide from [~~here~~](http://web2.cc.nctu.edu.tw/~claven/course/Algorithm/unit13.ppt) (link is now dead, slide is still good though):

* If all subproblems must be solved at least once, a bottom-up dynamic-programming algorithm usually outperforms a top-down memoized algorithm by a constant factor
  + No overhead for recursion and less overhead for maintaining table
  + There are some problems for which the regular pattern of table accesses in the dynamic-programming algorithm can be exploited to reduce the time or space requirements even further
* If some subproblems in the subproblem space need not be solved at all, the memoized solution has the advantage of solving only those subproblems that are definitely required

**Additional resources:**

* Wikipedia: [Memoization](http://en.wikipedia.org/wiki/Memoization), [Dynamic Programming](http://en.wikipedia.org/wiki/Dynamic_programming)
* Related SO Q/A: [Memoization or Tabulation approach for Dynamic programming](https://stackoverflow.com/questions/12042356/memoization-or-tabulation-approach-for-dynamic-programming)

# which is better approach in terms of time and space complexity?

Short answer: **it depends on the problem!**

*Memoization* usually requires more code and is less straightforward, but has computational advantages in some problems, mainly those which you do *not* need to compute all the values for the whole matrix to reach the answer.

*Tabulation* is more straightforward, but may compute unnecessary values. If you do need to compute all the values, this method is usually faster, though, because of the smaller overhead.