**DYNAMMIC PROGRAMMING NOTES**

**Techniques and Theory:**

* DP Main Problem Types:
  + Maximization / Minimization
  + Measuring combinations
* DP Methods of Solving:
  + Recursion & Memoization
  + Bottom-Up
* DP Problem Solving Steps:
  + Determine how many parameters are there
  + Find the base case(s)
  + Think of sub-problems and find a pattern
  + Generalize the relationship among sub-problems
  + Choose to either use Bottom-up or memoization
  + Write code by implementing base cases first and then the general relationship model
* DP Helpful Approach Methods:
  + Visualize the problem using drawings and many examples
  + Think of the start and of the problem and search for a pattern there
  + Always have in mind the concept of dynamic programming problem solving and the main methods of solving a DP problem. (memo and bottom-up)

**Problem Types:**

1. Maximization – ‘Kadane Algorithm’:

* **Code Description Notes:**
  + ‘Sum’ is used a current variable and ‘ans’ is used as a final variable
  + The first condition sets the ‘ans’ and the second neglects any negative values as the answer will never be negative (minimum: 0)
* **Code Example Reference: (**[**https://ideone.com/lHLHcX**](https://ideone.com/lHLHcX)**)**

1. #include<iostream>
2. using namespace std;
4. int kadane(int arr[], int N){
5. int ans = 0, sum = 0;
6. for (int i = 0; i < N; i++) {
7. sum += arr[i];
8. if (ans < sum)
9. ans = sum;
10. if (sum < 0)
11. sum = 0;
12. }
13. return ans;
14. }
16. int main() {
17. int A[] = {-2, -3, 4, -1, -2, 1, 5, -3};
18. int max\_sum = kadane(A, 8);
19. cout << max\_sum;
20. return 0;
21. }

2. Measuring Combinations – ‘Example 2’:

* **Code Description Notes:** 
  + The general concept of this problem is to express the answer for N, as an answer for (X<N)+ Y, where X is another ‘N’ to be solved and Y is a constant mathematical procedure.
  + Build on the thought that for any N > 4, if a combination ends with ‘+1’, then the sum of combinations of that combination will be DP[N-1]. For example, if (N = 5), then the answer can also be expressed as the answer for {(N=4) + 1}. Also, for any combinations ending with ‘+2’ or ‘+3’ their sums will be {(N=3) + 2} and {(N = 2) + 3}. Therefore, we can say that DP[N] = DP[N-1] + DP[N-2] + DP[N-3].
  + So, we first initialize the first four known and needed positions of the array.
  + Then we run the above model to get the result.
* **Code Example Reference: (DP Round 2 - PowerPoint)**

#include <iostream>

using namespace std;

int dp[1001];

int main() {

int n;

cin>>n;

dp[0] = 0;

dp[1] = 1;

dp[2] = 2;

dp[3] = 4;

for (int i = 4; i<=n; i++)

dp[i] = dp[i-1] + dp[i-2] + dp[i-3];

cout<<dp[n];

return 0;

}