Fibonacci

Bottom Up

```c
#include<stdio.h>
int Fibonacci(int N)
{
    int Fib[N+1],i;
    Fib[0] = 0;
    Fib[1] = 1;

    for(i = 2; i <= N; i++)
        Fib[i] = Fib[i-1]+Fib[i-2];

return Fib[N];
}

int main()
{
    int n;
    scanf("%d",&n);

    if(n <= 1)
        printf("Fib(%d) = %d\n",n,n);
    else
        printf("Fib(%d) = %d\n",n,Fibonacci(n));

    return 0;
}
```

Top down

```c
#include<stdio.h>

int Fibonacci(int N)
{
    if(N <= 1)
        return N;
    return Fibonacci(N-1) + Fibonacci(N-2);
}

int main()
{
    int n;
    scanf("%d",&n);
    printf("Fib(%d) = %d\n",n,Fibonacci(n));

    return 0;
}
```

Knapsack

```c
#include<stdio.h>

void knapSack(int W, int n, int val[], int wt[]);
int getMax(int x, int y);

int main(void) {

  int val[] = {-1, 100, 20, 60, 40};
  int wt[] = {-1, 3, 2, 4, 1};

  int n = 4;
  int W = 5;

  knapSack(W, n, val, wt);

  return 0;
}

int getMax(int x, int y) {
  if(x > y) {
    return x;
  } else {
    return y;
  }
}

void knapSack(int W, int n, int val[], int wt[]) {
  int i, w;
  int V[n+1][W+1];

  for(w = 0; w <= W; w++) {
    V[0][w] = 0;
  }

  for(i = 0; i <= n; i++) {
    V[i][0] = 0;
  }

  for(i = 1; i <= n; i++) {
    for(w = 1; w <= W; w++) {
      if(wt[i] <= w) {
        V[i][w] = getMax(V[i-1][w], val[i] + V[i-1][w - wt[i]]);
      } else {
```

```
        V[i][w] = V[i-1][w];
      }
    }
  }

  printf("Max Value: %d\n", V[n][W]);
}
```

Dijkstra

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 4

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {
  int cost[MAX][MAX], distance[MAX], pred[MAX];
  int visited[MAX], count, mindistance, nextnode, i, j;

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (Graph[i][j] == 0)
        cost[i][j] = INFINITY;
      else
        cost[i][j] = Graph[i][j];

  for (j = 0; j < n; j++) {
    distance[j] = cost[start][j];
    pred[j] = start;
    visited[j] = 0;
  }

  distance[start] = 0;
  visited[start] = 1;
  count = 1;

  while (count < n - 1) {
    mindistance = INFINITY;

    for (i = 0; i < n; i++)
      if (distance[i] < mindistance && !visited[i]) {
        mindistance = distance[i];
```

```
        nextnode = i;
      }

    visited[nextnode] = 1;
    for (i = 0; i < n; i++)
      if (!visited[i])
        if (mindistance + cost[nextnode][i] < distance[i]) {
          distance[i] = mindistance + cost[nextnode][i];
          pred[i] = nextnode;
        }
    count++;
  }

  for (i = 0; i < n; i++)
    if (i != start) {
      printf("\nDistance from source to %d: %d", i, distance[i]);
    }
}
int main() {
  int Graph[MAX][MAX], i, j, n, u;
  n = 4;

  Graph[0][0] = 0;
  Graph[0][1] = 10;
  Graph[0][2] = 15;
  Graph[0][3] = 0;

  Graph[1][0] = 0;
  Graph[1][1] = 0;
  Graph[1][2] = 0;
  Graph[1][3] = 10;


  Graph[2][0] = 0;
  Graph[2][1] = 0;
  Graph[2][2] = 0;
  Graph[2][3] = 10;


  Graph[3][0] = 0;
  Graph[3][1] = 0;
  Graph[3][2] = 0;
  Graph[3][3] = 0;
```

```
  u = 0;
  Dijkstra(Graph, n, u);

  return 0;
}
```