Coin Change Problem using dynamic programming

```
#include <stdio.h>
#define M(x,y) (x>y?y:x)
Int C(int*c, int n, int a, int*i){
    int d[a+1],j=0;
    for(d[*i=0]=0;++*i<=a;)for(d[*i]=a+1,j=0;j<n;j++)if(c[j]<=*i)d[*i]=M(d[*i-c[j]]+1,d[*i]);
    return d[a]>a?-1:d[a];
}
int main(){
    int c[100], n, a, i; //ican
    printf("Coins: "); scanf("%d", &n);
    printf("Coin values: "); for (i=0; i<n; i++) scanf("%d", &c[i]);
    printf("Amount: "); scanf("%d", &a);
    printf("Min coins: %d\n", C(c, n, a, &i));
    return 0;
}
```

Longest Common Subsequence (LCS) using dynamic programming

```
#include <stdio.h>
#define N 1000

int n, m, i, j, f[N], g[N];
int s1[N], s2[N]; // Assuming the sequences are of integer type

int main() {
    printf("Enter the length of sequence 1: ");
    scanf("%d", &n);

    printf("Enter sequence 1: ");
    for (i = 1; i <= n; i++)
        scanf("%d", &s1[i]);
    printf("Enter the length of sequence 2: ");
    scanf("%d", &m);

    printf("Enter sequence 2: ");
    for (j = 1; j <= m; j++)
        scanf("%d", &s2[j]);

    for (i = 1; i <= n; i++) {
        g[0] = 0;
        for (j = 1; j <= m; j++) {
            if (f[j] > g[j - 1])
                g[j] = f[j];
            else
                g[j] = g[j - 1];

            if (s1[i] == s2[j])
                f[j] = g[j - 1] + 1;
            else
                f[j] = 0;
        }
    }

    printf("Longest common subsequence length: %d\n", g[m]);
    return 0;
}
```

Knapsack Problem using dynamic programming

```
#include <stdio.h>

#define N 1000


int n, m, i, j, f[N];


int main() {
    printf("Enter the number of items: ");
    scanf("%d", &n);

    printf("Enter the maximum weight: ");
    scanf("%d", &m);

    for (i = 1; i <= n; i++) {
        int w, v;
        printf("Enter the weight and value of item %d: ", i);
        scanf("%d%d", &w, &v);

        for (j = m; j >= w; j--)
            if (f[j] < f[j - w] + v)
                f[j] = f[j - w] + v;
    }

    printf("Maximum value: %d\n", f[m]);
    return 0;
}
```

Merge Sort using divide and conquer algorithm

```
#include <stdio.h>
#define N 100000

int n, a[N], t[N];

void mergeSort(int l, int r) {
    if (l == r)
        return;

    int m = (l + r) / 2;
    int i = l, j = m + 1, k = l;
    mergeSort(l, m);
    mergeSort(m + 1, r);

    while (i <= m && j <= r)
        t[k++] = a[i] < a[j] ? a[i++] : a[j++];

    while (i <= m)
        t[k++] = a[i++];

    while (j <= r)
        t[k++] = a[j++];
    for (i = l; i <= r; i++)
        a[i] = t[i];
}

int main() {
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    printf("Enter the elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);

    mergeSort(0, n - 1);

    printf("Sorted elements: ");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Quick Sort using divide and conquer algorithm (pivot at the first or last position)

```
#include <stdio.h>

#define q(p, r, t) \
    if (p < r) { \
        int i = p, j = r, x = t[(p + r) / 2]; \
        while (i <= j) { \
            while (t[i] < x) \
                i++; \
            while (x < t[j]) \
                j--; \
            if (i <= j) { \
                S(i, j, t); \
                t[i] = t[i] ^ t[j], t[j] = t[j] ^ t[i], t[i] ^= t[j]; \
                i++, j--; \
            } \
        } \
        q(p, j, t); \
        q(i, r, t); \
    }

#define S(a, b, t) q(a, b, t)

int main() {
    int n = 6;
    int arr[6] = {9, 3, 7, 1, 5, 2};

    S(0, n - 1, arr);

    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Matrix Chain Multiplication using dynamic programming

```
#define M 50
```

```
int n, m[M], f[M][M], i, j, k; int main() {     scanf("%d", &n);     for (i = 1; i <= n;
i++) scanf("%d", &m[i]), f[i][i] = 0;     for (i = n-1; i; i--) for (j = i+1; j <= n; j++)
for (k = i; k < j; k++) f[i][j] = f[i][j] ?
f[i][j] < f[i][k]+f[k+1][j]+m[i-1]*m[k]*m[j] : f[i][k]+f[k+1][j]+m[i-1]*m[k]*m[j];
printf("%d\n", f[1][n]);     return 0;
}
```

```
#include <stdio.h>
#define M 50

int n, m[M], f[M][M], i, j, k;

int main() {
    printf("Enter the number of matrices: ");
    scanf("%d", &n);

    printf("Enter the dimensions of the matrices:\n");
    for (i = 1; i <= n + 1; i++)
        scanf("%d", &m[i]);

    for (i = 1; i <= n; i++)
        f[i][i] = 0;

    for (i = n - 1; i >= 1; i--) {
        for (j = i + 1; j <= n; j++) {
            f[i][j] = __INT_MAX__;
            for (k = i; k < j; k++) {
                int cost = f[i][k] + f[k + 1][j] + m[i] * m[k + 1] * m[j + 1];
                if (cost < f[i][j])
                    f[i][j] = cost;
            }
        }
    }
    printf("Minimum number of multiplications: %d\n", f[1][n]);

    return 0;
}
```

Knuth-Morris-Pratt string matching algorithm

```
#include <stdio.h>
#include <string.h>

void kmp(char* t, char* p) {
    int n = strlen(t), m = strlen(p), i = 0, j = 0, k = -1;     int nxt[m];
nxt[0] = -1;     while (i < m) k = (k == -1 || p[i] == p[k]) ? ++k, nxt[++i]
= k : nxt[k];     i = j = 0;     while (i < n && j < m) j = (j == -1 || t[i] ==
p[j]) ? ++i, ++j : nxt[j];     if (j == m) printf("Pattern found at index
%d\n", i - m);     else printf("Pattern not found\n");
}

int main() {     char text[] = "abcxabcdabcdabcy", pattern[] =
"abcdabcy";     kmp(text, pattern);     return 0;
}
```

Alter

```
#include <stdio.h>
#include <string.h>

void kmp(char* t, char* p) {
    int n = strlen(t), m = strlen(p), i = 0, j = 0, k = -1;
    int nxt[m];
    nxt[0] = -1;
    while (i < m) {
        k = (k == -1 || p[i] == p[k]) ? ++k : nxt[k];
        nxt[++i] = k;
    }
    i = j = 0;
    while (i < n && j < m) {
        j = (j == -1 || t[i] == p[j]) ? ++j : nxt[j];
        i++;
    }
    if (j == m) {
        printf("Pattern found at index %d\n", i - m);
    } else {
        printf("Pattern not found\n");
    }
}

int main() {
    char text[100], pattern[100];
    printf("Enter the text string: ");
    scanf("%s", text);
    printf("Enter the pattern string: ");
    scanf("%s", pattern);
    kmp(text, pattern);
    return 0;
}
```

Rabin-Karp string matching algorithm

```
#include <stdio.h>

#include <string.h>


#define p 31


int main() {

    char t[100], p[100];


    printf("Enter the text: ");
```

```
scanf("%s", t);

printf("Enter the pattern: ");
scanf("%s", p);

int n = strlen(t), m = strlen(p), i, j, pow_p = 1, hash_p = 0, hash_t = 0;

for (i = 0; i < m - 1; i++) {
    pow_p = (pow_p * p);
}

for (i = 0; i < m; i++) {
    hash_p = (hash_p * p + p[i]);
}

for (i = 0; i < m; i++) {
    hash_t = (hash_t * p + t[i]);
}

for (i = 0; i <= n - m; i++) {
    if (hash_t == hash_p) {
        for (j = 0; j < m; j++) {
            if (t[i + j] != p[j]) {
                break;
                }
            }

            if (j == m) {
                printf("Pattern found at index %d\n", i);
            }
        }

        if (i < n - m) {
            hash_t = (hash_t - t[i] * pow_p) * p + t[i + m];
        }
    }

    return 0;
}
```

Job Sequencing with deadlines using greedy algorithm

```
#include <stdio.h>
#define S(a,b) t=a,a=b,b=t

int main() {
    int n, d[100], p[100], t, i, j, k, f=0;

    printf("Enter the number of items: ");
    scanf("%d", &n);

    printf("Enter the deadlines and profits of each item:\n");
    for(i=0; i<n; i++) {
        scanf("%d %d", &d[i], &p[i]);
    }

    for(i=0; i<n; i++) {
        for(j=i; j<n; j++) {
            if(p[i]<p[j]) {
                S(p[i],p[j]);
                S(d[i],d[j]);
            }
        }
    }

    for(i=0; i<n; i++) {
        for(j=d[i]-1; j>=0; j--) {
            if(!f[j]) {
                f[j]=p[i];
                break;
            }
        }
    }

    k = 0;
    for(i=0; i<100; i++) {
        if(f[i]) {
            k += f[i];
        }
    }

    printf("Maximum profit: %d", k);
    return 0;
}
```

Activity Selection problem using greedy algorithm

```
#include<stdio.h>

int main(){
    int n, f=-1, s, e, i, j, t;
    printf("Enter the number of intervals: ");
    scanf("%d", &n);
    int a[n][2];
    printf("Enter the intervals:\n");
    for(i=0; i<n; i++){
        printf("Interval %d: ", i+1);
        scanf("%d %d", &a[i][0], &a[i][1]);
    }

    // Sort intervals based on ending time
    for(i=0; i<n; i++){
        for(j=i+1; j<n; j++){
            if(a[j][1] < a[i][1]){
                t = a[i][1];
                a[i][1] = a[j][1];
                a[j][1] = t;

                t = a[i][0];
                a[i][0] = a[j][0];
                a[j][0] = t;
            }
        }
    }

    // Find maximum number of non-overlapping intervals
    printf("Maximum number of non-overlapping intervals: ");
    for(i=0; i<n; i++){
        if(a[i][0] >= f){
            printf("%d ", i+1);
            f = a[i][1];
        }
    }
    return 0;
}
```

Job Scheduling Problem using greedy algorithm

```
#include <stdio.h>
#define N 100

int main() {
    int n, i, j, t, a[N], b[N], c[N];

    // Input
```

```
printf("Enter the number of intervals: ");
scanf("%d", &n);
printf("Enter the start and end times of each interval:\n");
for (i = 0; i < n; i++) {
    scanf("%d%d", &a[i], &b[i]);
}

// Initialize index array
for (i = 0; i < n; i++) {
    c[i] = i;
}

// Sort index array based on end times
for (i = 0; i < n-1; i++) {
    for (j = i+1; j < n; j++) {
        if (b[c[i]] > b[c[j]]) {
            t = c[i];
            c[i] = c[j];
            c[j] = t;
        }
    }
}

// Calculate and output start and end times of merged intervals
```

```
    for (t = i = 0; i < n; i++) {
        printf("%d %d ", t += a[c[i]], t + b[c[i]]);
    }

    return 0;
}
```

Fractional Knapsack using greedy algorithm

```
#include <stdio.h>
#define N 100

int main() {
    int n, i, j;
    double m, v[N], w[N], r[N], t;

    // Input the number of items and maximum weight
    scanf("%d%lf", &n, &m);

    // Input the value and weight of each item and calculate its ratio
    for (i = 0; i < n; i++) {
        scanf("%lf%lf", &v[i], &w[i]);
        r[i] = v[i] / w[i];
    }

    // Sort the items by decreasing ratio
    for (i = 0; i < n-1; i++) {
        for (j = i+1; j < n; j++) {
            if (r[i] < r[j]) {
                t = r[i], r[i] = r[j], r[j] = t;
                t = v[i], v[i] = v[j], v[j] = t;
                t = w[i], w[i] = w[j], w[j] = t;
            }
        }
    }

    // Calculate the maximum value that can be carried with the given weight limit
    for (t = i = 0; i < n && m > 0; i++) {
        t += w[i] < m ? v[i] : m / w[i] * v[i];
        m -= w[i] < m ? w[i] : m;
    }

    // Output the maximum value that can be carried
    printf("%.4lf", t);

    return 0;
}
```

Alternative

```
#include <stdio.h>
#define N 100

int main() {
int n, i, j;
    double m, v[N], w[N], r[N], t;
scanf("%d%lf", &n, &m);      for (i
= 0; i < n; i++) {
scanf("%lf%lf", &v[i], &w[i]);
r[i] = v[i] / w[i];
    }
    for (i = 0; i < n - 1; i++) {
for (j = i + 1; j < n; j++) {
if (r[i] < r[j]) {
                t = r[i], r[i] = r[j], r[j] = t;
t = v[i], v[i] = v[j], v[j] = t;                  t
= w[i], w[i] = w[j], w[j] = t;
            }
        }
    }
    for (t = i = 0; i < n && m > 0; i++) {
if (w[i] < m) {             t += v[i];
m -= w[i];
        } else {             t +=
v[i] * m / w[i];             m =
0;
        }
    }
printf("%.4lf", t);
return 0;
}
```

Minimum Spanning Tree using greedy algorithm (Prim's algorithm)

```
#include <stdio.h>
#define N 100

int main() {     int n, i, j, u, v, w, m = 0, t = 0, d[N], p[N], a[N][N];
scanf("%d%d", &n, &u);     for (i = 0; i < n; i++) for (j = 0; j < n; j++) scanf("%d",
&a[i][j]);     for (i = 0; i < n; i++) d[i] = 1e9, p[i] = -1;     for (d[u] = 0, i = 0; i < n;
i++) {         for (v = -1, j = 0; j < n; j++) if (d[j] < 1e9 && (v == -1 || d[j] < d[v]))
v = j;         if (v == -1) break; for (j = 0; j < n; j++) if (a[v][j] < d[j]) d[j] = a[v][j],
p[j] = v;
    }
    for (i = 0; i < n; i++) if (p[i] >= 0) m += a[i][p[i]], t++;
printf("%d", t == n-1 ? m : -1);     return 0;
}
```

Minimum Spanning Tree using greedy algorithm (Kruskal's algorithm)

```
#include <stdio.h>
#define N 100
```

```
int main() {     int n, m, i, j, u, v, w,
c = 0, p[N]; scanf("%d%d", &n,
&m);
    for (i = 0; i < n; i++) p[i] = i;
    for (i = 0; i < m; i++) scanf("%d%d%d", &u, &v, &w), p[u] != p[v] && (c += w, j
= p[u], p[u] = p[v], v = j, j = i);
printf("%d", j == n-1 ? c : -1);     return
0;
}
```

Huffman code using greedy algorithm

```
#include <stdio.h>
#include <stdlib.h>
#define N 256

typedef struct node{int f;char c;struct node*l,*r;}node; typedef
struct list{int n;node*a[N];}list;

void ins(list*l,node*n){int i=l->n++;while(i>0&&n->f<l->a[i-1]->f)l->a[i]=l->a[-
i];l->a[i]=n;}

node*pop(list*l){return l->n>0?l->a[--l->n]:0;}

node*huff(char*s){     list l={0};int
f[N]={0},i;for(i=0;s[i];i++)f[(int)s[i]]++;
for(i=0;i<N;i++)if(f[i])ins(&l,&(node){f[i],i,0,0});while(l.n>1){
node*lft=pop(&l),*rgt=pop(&l),*prnt=malloc(sizeof(node));
prnt->f=lft->f+rgt->f;prnt->l=lft;prnt->r=rgt;ins(&l,prnt);}
return l.n>0?l.a[0]:0;
}

void enc(node*t,char*s,char*b){if(!t)return;if(t->c)*b=*s,enc(t->l,s+1,b+1);else
enc(t->l,s,b),enc(t->r,s,b+1);}
```

```
void dec(node*t,char*b){if(!t)return;putchar(t->c?t->c:*b?dec(*b++=='0'?t->l:t-
>r,b):0);}

int main(){char
s[N],b[N]={0};scanf("%s",s);node*t=huff(s);enc(t,s,b);puts(b);dec(t,b);return 0;}
```

Alter

```
#include<stdio.h>
#include<stdlib.h>
#define N 26
#define M 100

int freq[N],n,i,j,a[N],b[N],k,t,x[M],y[M],z[M],l[M],s[M],c[M],o[M];

int main(){
scanf("%d",&n);
for(i=0;i<n;i++){
scanf("%d",&freq[i]);
a[i]=i;
    }
    for(i=0;i<n-1;i++){         t=i+1;
for(j=i+2;j<n;j++) if(freq[t]>freq[j]) t=j;
k=a[i],a[i]=a[t],a[t]=k;
k=freq[i],freq[i]=freq[t],freq[t]=k;
k=a[i+1],b[k]=i,i=i+1;
        for(j=i+1;j<n;j++) if(freq[i]>freq[j]) i=j;
k=a[i],a[i]=a[t],a[t]=k;
k=freq[i],freq[i]=freq[t],freq[t]=k;          b[k]=i,i=i-1;
    }
    for(i=0;i<n;i++){          k=a[i];
while(k!=n){
c[k]=s[k],s[k]=s[k]+1,k=b[k];
        }
    }
    for(i=0;i<n;i++){
        l[c[i]]=i;          for(j=c[i];j>0;j--){              y[j-
1]=y[j]+(z[j]+1)/2,x[j-1]=x[j]+z[j]/2,z[j-1]=2*z[j];
if((l[j-1]-l[j])==1){                   z[j-1]=z[j-1]+z[j]+1;
            }else{                  z[j-
1]=z[j-1]+z[j];
            }
        }
        o[i]=x[0],z[c[i]]=1;
    }
    for(i=0;i<n;i++){
printf("%d %d\n",i+1,o[i]);
    }
    return 0;
}
```

Travelling Salesman Problem using dynamic programming

```
#include <stdio.h>
#define N 16

int n, m, d[N][N], i, j, k, p[N][1<<N];

int main() {     scanf("%d", &n);     for (i = 0; i < n; i++) for (j = 0; j < n;
j++) scanf("%d", &d[i][j]);     for (k = 1; k < n; k++) for (i = 0; i < n;
i++) for (j = 0; j < 1<<n; j++) p[i][j|(1<<k)] =
!k?d[i][k]:j&(1<<k)?p[k][j]+d[i][k]<p[i][j|(1<<k)]?p[k][j]+d[i][k]:p[i][j|(1<<k)]:p[i][j|(1
<<k)];     for (m = i = 1; i < n; i++) m
*= i+1; for (k = 1; k < m; k++) for (i
= j = 0; j < n; j++) i += p[j][k]<<j;
printf("%d", i);
    return 0;
}
```

Shortest Paths using greedy algorithm

```
#include <stdio.h>
#define N 1000

int n, m, g[N][N], d[N], v[N], i, j, x, y;

int main() {     scanf("%d%d", &n, &m);     for (i = 0; i < m; i++)
scanf("%d%d%d", &x, &y, &g[x][y]);     for (i = 1; i < n; i++) d[i] =
1e9;     for (i = 0; i < n; i++) {         for (x = -1, j = 0; j < n; j++) if (!v[j]
&& (x < 0 || d[j] < d[x])) x = j;         if (d[x] == 1e9) break;         v[x] =
1;         for (y = 0; y < n; y++) if (g[x][y]) d[y] = d[y] < d[x]+g[x][y] ?
d[y] :
d[x]+g[x][y];
    }
    for (i = 1; i < n; i++) printf("%d ", d[i]);
return 0;
}
```

Sum of Subsets using backtracking

```
#include <stdio.h>
#define N 100

int n, w, a[N], s, i;

void dfs(int x, int t) {
    if (x == n) { if (t == w) for (i = 0; i < n; i++) if (a[i]) printf("%d ", i); }
    else { a[x] = 1, dfs(x+1, t+s-x), a[x] = 0, dfs(x+1, t); }
}

int main() {     scanf("%d%d", &n, &w);     for (i
= 0; i < n; i++) scanf("%d", &s), a[i] = 0;
dfs(0, 0);     return 0;
}
```

8-Queen problem using backtracking

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 8

int queen[MAX];
int n_solutions = 0;

int is_valid(int row, int col) {
for (int i = 0; i < row; i++)
    if (queen[i] == col || abs(queen[i] - col) == abs(i - row))
return 0;
  return 1;
}

void print_solution() {
  printf("\nSolution %d:\n", ++n_solutions);
  for (int i = 0; i < MAX; i++) {     for (int j =
0; j < MAX; j++)       printf("%s ", queen[i]
== j ? "Q" : "-");     printf("\n");
  }
}

void solve(int row) {
if (row == MAX) {
print_solution();
    return;
  }

  for (int col = 0; col < MAX; col++) {
if (is_valid(row, col)) {
queen[row] = col;
      solve(row + 1);
```

```
   }
  }
}

int main() {
solve(0);   return
0;
}
```

Alternative int

```
a[99],i,j,k;

int ok(int r,int c){     for(i=0;i<r;++i){
if(a[i]==c||a[i]-i==c-r||a[i]+i==c+r)
return 0;
    }
    return 1;
}

void queen(int r){
if(r==8){
        for(k=0;k<8;++k) printf("%d",a[k]+1);
printf("\n");         return;
    }
    for(c=0;c<8;++c){
        if(ok(r,c)){
a[r]=c;
queen(r+1);
        }
    }
}

int main(){
queen(0);     return
0;
}
```

Fifteen Puzzle problem using branch and bound

```
#include <stdio.h>
#define T(a,b)(a^=b,b^=a,a^=b)
int S(int*b,int l,int u){   if(u-l<2)
    return!memcmp(b,(int[]){1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0},64);
int m=u-l>>1,z=S(b,l,l+m),y=S(b,l+m,u);   if(z|y){     int s=0,i=0,j=l;
for(;j<u&&b[j];j++);     for(;i<m+j;b[j++]=s)       if((s+=b[i])>0)
        s-=16;
  }
  return z|y;
}
int main(){
int b[16],i;
for(;i<16;++i)
    scanf("%d",b+i);
  printf(S(b,0,16)?"SOLVABLE\n":"UNSOLVABLE\n");
return 0;
}
```