Divide And conquer

```
#include  <stdio.h>

#include <conio.h>

typedef struct {

    int left;

    int right;

    int sum;

} Result;

Result css(int arr[], int l, int m, int h) {

    int sum, i, left_sum, left_i, right_sum, right_i;

    sum = 0;

    left_sum = -10000;

    left_i = m;

    for (i = m; i >= l; i--) {

        sum += arr[i];

        if (sum > left_sum) {

            left_sum = sum;

            left_i = i;

        }

    }

    sum = 0;

    right_sum = -10000;

    right_i = m + 1;

    for (i = m + 1; i <= h; i++) {

        sum += arr[i];

        if (sum > right_sum) {

            right_sum = sum;
            right_i = i;
        }
    }
    Result res = {left_i, right_i, left_sum + right_sum};
    return res;
}
Result maxs(int arr[], int l, int h) {
    int m;
    Result left_res, right_res, cross_res;
    if (l == h) {
        Result res = {l, h, arr[l]};
        return res;
    }
    m = (l + h) / 2;
    left_res = maxs(arr, l, m);
    right_res = maxs(arr, m + 1, h);
    cross_res = css(arr, l, m, h);
    if (left_res.sum >= right_res.sum && left_res.sum >= cross_res.sum)
        return left_res;
    else if (right_res.sum >= left_res.sum && right_res.sum >= cross_res.sum)
        return right_res;
    else
        return cross_res;
}
int main() {
    int n, k, i, arr[100];
    Result res;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements: ");
    for (i = 0; i < n; ++i)
        scanf("%d", &arr[i]);
    res = maxs(arr, 0, n - 1);
    printf("Maximum subarray:\n");
    for (k = res.left; k <= res.right; ++k)
        printf("%d ", arr[k]);
    printf("\nSum: %d\n", res.sum);
    getch();
    return 0;
}
```

Dynamic_Max_sub_array:

```
#include  <stdio.h>
#include <conio.h>
int main()
{
    int n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[100];
    printf("Enter the elements: ");
    for (i = 0; i < n; ++i)
        scanf("%d", &arr[i]);
    int curr_sum = arr[0], max_sum = arr[0];
    for (i = 1; i < n; i++)
    {
        if (curr_sum + arr[i] > arr[i])
        {
            curr_sum += arr[i];
        }
        else
        {
            curr_sum = arr[i];
        }
        if (max_sum < curr_sum)
            max_sum = curr_sum;
    }
    printf("The maximum sum of the subarray is: %d", max_sum);
    getch();
    return 0;
}
```

Greedy Knapsack:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
typedef struct
{
    int weight;
    int profit;
} Item;
float *greedy_knapsack(int n, int m, Item *arr, int *frac)
{
```

```
int i, j;

float pw[100], temp, *x;

Item itemtemp;

x = (float *)malloc(n * sizeof(float));

for (i = 0; i < n; i++)

{

    pw[i] = (float)arr[i].profit / arr[i].weight;

}

for (i = 0; i < n; i++)

{

    for (j = 0; j < n - i - 1; j++)

    {

        if (pw[j] < pw[j + 1])

        {

            temp = pw[j];

            pw[j] = pw[j + 1];

            pw[j + 1] = temp;

            temp = frac[j];

            frac[j] = frac[j + 1];

            frac[j + 1]  = temp;

            itemtemp  = arr[j];

            arr[j] = arr[j + 1];

            arr[j + 1] = itemtemp;

        }

    }

}

for (i = 0; i < n; i++)
```

```
    {
        x[i] = 0.0;
    }
    for (i = 0; i < n; i++)
    {
        if (m > arr[i].weight)
        {
            x[i] = 1.0;
            m -= arr[i].weight;
        }
        else
        {
            x[i] = (float)m / arr[i].weight;
            break;
        }
    }
    return x;
}
int main()
{
    int n, i, m, *frac;
    printf("Enter total weight of the knapsack = ");
    scanf("%d", &m);
    printf("Enter the number of items = ");
    scanf("%d", &n);
    Item arr[100];
    float *x, max_profit = 0;
```

```
frac = (int *)malloc(n * sizeof(int));
for (i = 0; i < n; i++)
{
    frac[i] = i + 1;
}
for (i = 0; i < n; i++)
{
    printf("\nEnter profit of %d element = ", i + 1);
    scanf("%d", &arr[i].profit);
    printf("Enter weight of %d element = ", i + 1);
    scanf("%d", &arr[i].weight);
}
x = greedy_knapsack(n, m, arr, frac);
for (i = 0; i < n; i++)
{
    max_profit += (x[i] * arr[i].profit);
}
printf("Maximum profit is = %.2f", max_profit);
printf("\nItems inserted in the knapsack are:");
for (i = 0; i < n; i++)
{
    if (x[i] > 0)
    {
        printf("%d ", frac[i]);
    }
}
free(x);
```

```
    getch();
    return 0;
}
```

Rod_Cutting:

```
#include  <stdio.h>
#include <conio.h>
#include <limits.h>
int rod_cutting(int price[], int n)
{
    int dp[100 + 1];
    dp[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        int max_profit = INT_MIN;
        for (int j = 1; j <= i; j++)
        {
            max_profit = (price[j - 1] + dp[i - j] > max_profit) ? price[j - 1] + dp[i - j] : max_profit;
        }
        dp[i] = max_profit;
    }
    return dp[n];
}
int main()
{
    int n;
    printf("Enter the length of the rod: ");
    scanf("%d", &n);
    int price[100];

    printf("Enter the prices for each length:\n");

    for (int i = 0; i < n; i++)

    {

        printf("Price for length %d: ", i + 1);

        scanf("%d", &price[i]);

    }

    int max_profit = rod_cutting(price, n);

    printf("Maximum profit is: %d\n", max_profit);

    getch();

    return 0;

}
```

N_Queens:

```
#include  <stdio.h>

#include <conio.h>

#define N 10

int board[N][N], num_solution = 0;

int is_safe(int board[N][N], int row, int col, int n) {

    int i, j;

    for (i = 0; i < row; i++) {

        if (board[i][col])

            return 0;

    }

    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {

        if (board[i][j])

            return 0;

    }
    for (i = row, j = col; i >= 0 && j < n; i--, j++) {
        if (board[i][j])
            return 0;
    }
    return 1;
}

int n_queens(int board[N][N], int row, int n) {
    int i, col, j;
    if (row == n) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                printf("%c ", board[i][j] ? 'Q' : '.');
            }
            printf("\n");
        }
        printf("\n");
        num_solution += 1;
        return 1;
    }

    int has_solution = 0;
    for (col = 0; col < n; col++) {
        if (is_safe(board, row, col, n)) {
            board[row][col] = 1;
            has_solution = n_queens(board, row + 1, n) || has_solution;
            board[row][col] = 0;
        }
    }
    return has_solution;
}

int main() {
    int n, i, j;
    printf("Enter the number of queens (less than 10): ");
    scanf("%d", &n);

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

    if (!n_queens(board, 0, n)) {
        printf("No solution exists\n");
    } else {
        printf("Number of Solutions = %d", num_solution);
    }

    getch();
    return 0;
}
```

Matrix chain Multiplication:

```
#include <stdio.h>
#include <conio.h>
#include <limits.h>
#define MAX 100

void matrixChainOrder(int p[], int n, int m[MAX][MAX], int s[MAX][MAX]) {
    for (int i = 1; i <= n; i++) {
        m[i][i] = 0;
    }

for (int l = 2; l <= n; l++) {
        for (int i = 1; i <= n - l + 1; i++) {
            int j = i + l - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k < j; k++) {

                int q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (q < m[i][j]) {
                    m[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
    }
}

void printOptimalParenthesis(int s[MAX][MAX], int i, int j) {
    if (i == j)
        printf("A%d", i);
    else {
        printf("(");
        printOptimalParenthesis(s, i, s[i][j]);
        printOptimalParenthesis(s, s[i][j] + 1, j);
        printf(")");
    }
}

int main() {
    int p[MAX], m[MAX][MAX], s[MAX][MAX];
    int n;

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

    printf("Enter the dimensions of matrices (p[0] to p[n]): \n");
    for (int i = 0; i <= n; i++) {
        scanf("%d", &p[i]);
    }

    matrixChainOrder(p, n, m, s);

    printf("Minimum number of multiplications is: %d\n", m[1][n]);

    printf("Optimal Parenthesization is: ");
    printOptimalParenthesis(s, 1, n);
    printf("\n");
    getch();
    return 0;


}
```