1 Maximum Subarray

```cpp
// find the consecutive subarray that have largest sum of elements
const int maxn = int(1e6 + 5);
int dp[maxn][2];

pair<double, int> bruteForce(int array[], int n) {
    cout << "Running brute force\n";
    int res = INT_MIN;
    clock_t startTime = clock();
    for (int i = 0; i < n; ++i) {
        for (int j = i; j < n; ++j) {
            int sum = 0;
            for (int k = i; k <= j; ++k) {
                sum += array[k];
            }
            if (sum > res) res = sum;
            if (runTime(startTime) > TIME_LIMIT) return {TIME_LIMIT, res};
        }
    }
    return {runTime(startTime), res};
}

pair<double, int> bruteForceWithImprovement(int array[], int n) {
    cout << "Running brute force with improvement\n";
    int res = INT_MIN;
    clock_t startTime = clock();
    for (int i = 0; i < n; ++i) {
        int sum = 0;
        for (int j = i; j < n; ++j) {
            sum += array[j];
            if (sum > res)
                res = sum;
        }
        if (runTime(startTime) > TIME_LIMIT)
            return {TIME_LIMIT, res};
    }
    return {runTime(startTime), res};
}

pair<double, int> dynamicProgramming(int array[], int n) {
    cout << "Running dynamic programming\n";
    clock_t startTime = clock();
    // dp[0][i]: largest sum of subsequence of a[0]...a[i-1]
    // dp[1][i]: largest sum of suffix array of a[0]...a[i]
    dp[0][0] = 0;
    dp[0][1] = array[0];
    for (int i = 1; i < n; ++i) {
        dp[i][0] = max(dp[i - 1][0], dp[i - 1][1]);
        dp[i][1] = max(dp[i - 1][1], 0) + array[i];
    }
    return {runTime(startTime), max(dp[n - 1][0], dp[n - 1][1])};
}
```

2 Binomial Coefficient

```cpp
const int maxn = 55;
ll D[maxn][maxn] = {0};

ll recursion_C(int n, int k) {
    if (k == 0 || n == k)
        return 1;
    else {
        return recursion_C(n - 1, k) + recursion_C(n - 1, k - 1);
    }
}

ll recursion_with_memory_C(int n, int k) {
    for (int i = 0; i <= n; ++i) {
        D[i][0] = 1;
        D[i][i] = 1;
    }
    if (D[n][k] == 0) {
        D[n][k] = recursion_with_memory_C(n - 1, k - 1) + recursion_with_memory_C(
            n - 1, k);
    }
    return D[n][k];
}
```

3 Binary Sequence

```cpp
//print all binary strings of length n

#include <bits/stdc++.h>
using namespace std;

const int maxn = 20;
bool a[maxn];

void print(int n) {
    for (int i = 0; i < n; ++i) {
        cout << a[i];
    }
    cout << endl;
}

void Try(int k, int n) {
    for (int y = 0; y <= 1; ++y) {
        a[k] = y;
        if (k == n - 1) {
            print(n);
        } else
            Try(k + 1, n);
    }
}

int main() {
    int n;
    cin >> n;
    Try(0, n);
}
```

4 n-Queens

```cpp
// print all solutions of n-queens puzzle

#include <bits/stdc++.h>
using namespace std;

const int maxn = 10;
int col[maxn];

bool check(int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (abs(col[i] - col[j]) == j - i) return false;
            if (i + col[i] == j + col[j]) return false;
        }
    }
    return true;
}

int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i) col[i] = i;
    do {
        if (check(n)) {
            cout << "CASE:\n";
            for (int i = 0; i < n; ++i) {
                cout << i << " " << col[i] << endl;
            }
        }
    } while (next_permutation(col, col + n));
}
```

5 Permutations

```cpp
// print all permutations of (1, 2, ..., n)

#include <bits/stdc++.h>
using namespace std;

const int maxn = 10;
int a[maxn];

// void cheating() {
//     do {
//         for (int i = 0; i < 5; ++i) {
//             cout << a[i];
//         }
//         cout << endl;
//     } while (next_permutation(a, a + 5));
// }

void print(int n) {
    for (int i = 0; i < n; ++i) {
        cout << a[i];
    }
    cout << endl;
}

bool chk(int i, int k) {
    for (int j = 0; j < k; ++j) {
        if (a[j] == i) return false;
    }
    return true;
}

void Try(int k, int n) {
    for (int i = 0; i < n; ++i) {
        if (!chk(i, k)) continue;
        a[k] = i;
        if (k == n - 1)
            print(n);
        else
            Try(k + 1, n);
    }
}

void permutation(int n) {
    Try(0, n);
}

int main() {
    int n = 5;
    permutation(n);
}
```

6 Positive Solutions

```cpp
// find all positive solution of x_1+x_2+...+x_n = N
#include <bits/stdc++.h>
using namespace std;
```

```
const int maxn = 20;
int N, n, x[maxn];
int cnt = 0;

void print(int t) {
    for (int i = 0; i < t; ++i) cout << x[i] << " ";
    cout << endl;
    ++cnt;
}

void Try(int k) {
    int M = 0;
    for (int i = 0; i < k; ++i) M += x[i];
    if (k == n - 1) {
        x[k] = N - M;
        print(n);
    } else
        for (int i = 1; i <= N - M - n + k + 1; ++i) {
            x[k] = i;
            Try(k + 1);
        }
}

int main() {
    cin >> n >> N;
    Try(0);
    cout << "Number of solutions: " << cnt << endl;
}
```

7 m-Subsets

```
// print all m element subsets of {1, 2, ..., n}

#include <bits/stdc++.h>
using namespace std;

const int maxn = 20;
int subset[maxn];

void print(int m) {
    for (int i = 0; i < m; ++i) {
        cout << subset[i] << " ";
    }
    cout << endl;
}

void Try(int k, int n, int m) {
    if (k == 0) {
        for (int i = 0; i < n; ++i) {
            subset[k] = i;
            Try(k + 1, n, m);
        }
    } else
        for (int y = subset[k - 1] + 1; y < n; ++y) {
            subset[k] = y;
            if (k == m - 1) {
                print(m);
            } else {
                Try(k + 1, n, m);
            }
        }
}

void mSubset(int n, int m) {
    Try(0, n, m);
}

int main() {
    int n = 7, m = 3;
    mSubset(7, 3);
}
```

8 Subset Sum

```
// determine if exist a subset with given sum

#include <iostream>
using namespace std;

const int maxn = 20;
int S[maxn];

bool isSubsetSum(int S[], int n, int sum) {
    if (n == 0) {
        if (sum == 0)
            return true;
        else
            return false;
    }
    if (sum < 0) return false;
    return isSubsetSum(S, n - 1, sum - S[n - 1]) || isSubsetSum(S, n - 1, sum);
}

int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i) cin >> S[i];
    int sum;
    cin >> sum;
    cout << isSubsetSum(S, n, sum);
}
```

9 Binary Search

```
/*
    binary search
    given a non-decrease array a[0..n-1]
    find the index of the first element which have value >= key
*/

#include <bits/stdc++.h>
using namespace std;

int main() {
    int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int n = 10, key = 5;
    int k = 0;
    for (int b = n / 2; b >= 1; b /= 2) {
        while (k + b < n && a[k + b] < key) k += b;
    }
    cout << k + 1 << endl;
    cout << lower_bound(a, a + n, key) - a;
}
```

10 Linked List

```
#include <bits/stdc++.h>
using namespace std;

struct node {
    int value;
    node *next;
};

node *makeNode(int v) {
    node *p = new node;
    p->value = v;
    p->next = NULL;
    return p;
}

void print(node *head) {
    while (head != NULL) {
        cout << head->value << " ";
        head = head->next;
    }
}

void insertToHead(node **head, int x) {
    node *newNode = makeNode(x);
    newNode->next = *head;
    *head = newNode;
}

node *insertToHead(node *head, int x) {
    node *newNode = makeNode(x);
    newNode->next = head;
    head = newNode;
    return head;
}

void insertAfter(node *cur, int x) {
    node *newNode = makeNode(x);
    newNode->next = cur->next;
    cur->next = newNode;
}

int countNode(node *head, int v) {
    int ans = 0;
    while (head != NULL) {
        if (head->value == v) ans++;
        head = head->next;
    }
    return ans;
}

int main() {
    node *head = NULL;
    int n, val;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cin >> val;
        insertToHead(head, val);
    }
    print(head);
    cout << countNode(head, 5);
}
```

11 Stack with Linked List

```
#include <bits/stdc++.h>
using namespace std;

struct StackNode {
    float item;
    StackNode *next;

    StackNode() {
        this->item = 0;
        this->next = NULL;
    }
};
struct Stack {
    StackNode *top;
    int size;
    int maxSize;
```

```
    Stack() {
        this->top = NULL;
        this->size = 0;
        this->maxSize = 5;
    }
};

Stack *StackConstruct() {
    Stack *s = new Stack();
    return s;
}

bool StackEmpty(Stack *s) {
    return (s->size == 0);
}

bool StackFull(Stack *s) {
    return (s->size == s->maxSize);
}

int StackPush(Stack *s, float item) {
    if (!StackFull(s)) {
        StackNode *newNode = new StackNode;
        newNode->next = s->top;
        newNode->item = item;
        s->top = newNode;
        s->size++;
        return 0;
    } else
        return 1;
}

float StackPop(Stack *s) {
    if (!StackEmpty(s)) {
        float t = s->top->item;
        StackNode *top = s->top;
        s->top = s->top->next;
        delete top;
        s->size--;
        return t;
    }
    return 0;
}

void Disp(Stack *s) {
    StackNode *p = s->top;
    while (p != NULL) {
        cout << p->item << " ";
        p = p->next;
    }
    cout << endl;
}

int main() {
    Stack *s = StackConstruct();
    StackPush(s, 5678);
    StackPush(s, 1234);
    StackPop(s);
    Disp(s);
}
```

12 Doubly Linked List

```
#include <bits/stdc++.h>
using namespace std;

struct dblist {
    int data;
    dblist *prev;
    dblist *next;
};
dblist *head, *tail;

void printListFromHead(dblist **head) {
    cout << "List from head: " << endl;
    dblist *p = *head;
    while (p != NULL) {
        cout << p->data << " ";
        p = p->next;
    }
    cout << endl;
}

void printListFromTail(dblist **tail) {
    cout << "List from tail: " << endl;
    dblist *p = *tail;
    while (p != NULL) {
        cout << p->data << " ";
        p = p->prev;
    }
    cout << endl;
}

void insertAfter(int x, dblist *p) {
    dblist *newNode = new dblist;
    newNode->data = x;
    newNode->next = p->next;
    newNode->prev = p;
    if (p != tail)
        p->next->prev = newNode;
    else
        tail = newNode;
    p->next = newNode;
}

void deleteNode(dblist *p) {
    if (p != head)
        p->prev->next = p->next;
    else
        head = head->next;

    if (p != tail)
        p->next->prev = p->prev;
    else
        tail = tail->prev;

    delete p;
}

int main() {
    dblist *p = new dblist;
    p->prev = NULL;
    p->data = 12;
    p->next = NULL;
    head = p;
    tail = p;
    insertAfter(34, p);
    insertAfter(56, p->next);
    deleteNode(p->next);
    printListFromHead(&head);
    printListFromTail(&tail);
}
```

13 Expression Evaluation

```
#include <bits/stdc++.h>
using namespace std;

stack<double> num;
stack<char> operators;

string trim(string s) {
    // remove the spaces at the start and end of string
    s.erase(s.find_last_not_of(" \n\r\t") + 1);
    s.erase(0, s.find_first_not_of(' '));
    return s;
}

string takeBeforeSpaceAndRemove(string &input) {
    // input = "   A       B.......     "
    // -> return A, input := B.......
    string A;
    input = trim(input);
    size_t found = input.find(" ");
    if (found == string::npos) {
        A = input;
        input = "";
    } else {
        A = input.substr(0, found);
        input = input.substr(found + 1, input.size());
    }
    A = trim(A);
    input = trim(input);
    return A;
}

bool isNum(string c) {
    // check if a string c is a number
    for (int i = 0; i < c.size(); ++i)
        if (c[i] != '.' && (c[i] < '0' || c[i] > '9')) return false;
    return true;
}

bool isOperator(string c) {
    if (c == "+" || c == "-" || c == "*" || c == "/" || c == "^")
        return true;
    else
        return false;
}

double stringToDouble(string c) {
    char *pEnd;
    return strtod(c.c_str(), &pEnd);
}

int priority(char c) {
    // return the priority of an (operator or bracket) c
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return 0;
}

void process() {
    // process top 2 elements in num stack with top element in operators stack
    double a = num.top();
    num.pop();
    double b = num.top();
    num.pop();
    char op = operators.top();
    operators.pop();
    if (op == '+')
        num.push(b + a);
    else if (op == '-')
        num.push(b - a);
    else if (op == '*')
        num.push(b * a);
    else if (op == '/')
        num.push(b / a);
    else if (op == '^')
        num.push(pow(b, a));
}

int main() {
    string input;
    getline(cin, input);
    input = trim(input);
    while (input != "") {
        string c = takeBeforeSpaceAndRemove(input);
        if (isNum(c)) {
```

```
            num.push(stringToDouble(c));
        } else if (isOperator(c)) {
            while (operators.size() > 0 && priority(operators.top()) >= priority(c
                [0]))
                process();
            operators.push(c[0]);
        } else if (c == "(") {
            operators.push(c[0]);
        } else if (c == ")") {
            while (operators.top() != '(') {
                if (operators.size() == 0) {
                    cout << "bieu thuc khong hop le";
                    return 0;
                }
                process();
            }
            operators.pop();
        } else {
            cout << "bieu thuc khong hop le";
            return 0;
        }
    }
    while (operators.size() > 0) {
        process();
    }
    cout << "ket qua: " << num.top();
}
```

14 Polynomial Product

```
// compute a polynomial product between f(x) and (1+2x)

typedef struct Polynom {
    int coeff;
    int pow;
    struct Polynom *link;
} poly;

void print(poly *t) {
    for (poly *cur = t; cur != NULL; cur = cur->link) {
        cout << cur->coeff << " " << cur->pow << endl;
    }
}

poly *tichdathuc(poly *px) {
    poly *px2x = NULL, *px2xtail;
    for (poly *pp = px; pp != NULL; pp = pp->link) {
        poly *newnode = new poly;
        newnode->coeff = 2 * pp->coeff;
        newnode->pow = pp->pow + 1;
        newnode->link = NULL;
        if (px2x == NULL) {
            px2x = newnode;
            px2xtail = newnode;
        } else {
            px2xtail->link = newnode;
            px2xtail = px2xtail->link;
        }
    }
    poly *ans = NULL, *anstail;
    for (poly *pp = px, *px2xcur = px2x; pp != NULL || px2xcur != NULL;) {
        poly *newnode = new poly;
        newnode->link = NULL;
        if (pp != NULL && px2xcur != NULL && pp->pow == px2xcur->pow) {
            newnode->coeff = pp->coeff + px2xcur->coeff;
            newnode->pow = pp->pow;
            pp = pp->link;
            px2xcur = px2xcur->link;
        } else if (px2xcur == NULL || (pp!=NULL&&pp->pow < px2xcur->pow)) {
            newnode = pp;
            pp = pp->link;
        } else {
            newnode = px2xcur;
            px2xcur = px2xcur->link;
        }
        if (ans == NULL) {
            ans = newnode;
            anstail = newnode;
        } else {
            anstail->link = newnode;
            anstail = anstail->link;
        }
    }
    return ans;
}
```

15 Sudoku

```
// 2d linked list and check sudoku condition

typedef struct _List {
    int info;
    struct _List *linkhang;
    struct _List *linkcot;
} listt;

void print(listt *t) {
    cout << "t";
    for (listt *cur = t; cur != NULL;) {
        cout << cur->info << " ";
        if (cur->linkhang == NULL)
            cur = cur->linkcot;
        else
            cur = cur->linkhang;
    }
}

listt *CreateSudokulist(int **p) {
    listt *SDK = NULL;
    listt *cottail, *hangtail, *hangtailprev = NULL;
    for (int hang = 0; hang < 9; hang++) {
        for (int cot = 0; cot < 9; cot++) {
            listt *newnode = new listt;
            newnode->info = p[hang][cot];
            newnode->linkhang = NULL;
            newnode->linkcot = NULL;
            if (SDK == NULL) {
                SDK = newnode;
                cottail = newnode;
                hangtail = newnode;
            } else if (cot == 0) {
                hangtailprev = hangtail;
                hangtail->linkhang = newnode;
                cottail = newnode;
                hangtail = hangtail->linkhang;
            } else {
                if (hangtailprev != NULL) {
                    hangtailprev = hangtailprev->linkcot;
                    hangtailprev->linkhang = newnode;
                }
                cottail->linkcot = newnode;
                cottail = cottail->linkcot;
            }
        }
    }
    return SDK;
}

bool check3x3(listt *t) {
    bool chk[9];
    chk[t->info - 1] = 1;
    chk[t->linkcot->info - 1] = 1;
    chk[t->linkcot->linkcot->info - 1] = 1;
    chk[t->linkhang->info - 1] = 1;
    chk[t->linkhang->linkcot->info - 1] = 1;
    chk[t->linkhang->linkcot->linkcot->info - 1] = 1;
    chk[t->linkhang->linkhang->info - 1] = 1;
    chk[t->linkhang->linkhang->linkcot->info - 1] = 1;
    chk[t->linkhang->linkhang->linkcot->linkcot->info - 1] = 1;
    for (int i = 0; i < 9; ++i)
        if (!chk[i])
            return 0;
    return 1;
}

int CheckSudokuCondition(listt *head) {
    for (listt *cur = head; cur != NULL; cur = cur->linkhang) {
        bool chk[9];
        for (listt *curcot = cur; curcot != NULL; curcot = curcot->linkcot) {
            if (curcot->info > 9 || curcot->info < 1) return 0;
            chk[curcot->info - 1] = 1;
        }
        for (int i = 0; i < 9; ++i) {
            if (!chk[i]) return 0;
        }
    }
    for (listt *cur = head; cur != NULL; cur = cur->linkcot) {
        bool chk[9];
        for (listt *curhang = cur; curhang != NULL; curhang = curhang->linkhang) {
            if (curhang->info > 9 || curhang->info < 1) return 0;
            chk[curhang->info - 1] = 1;
        }
        for (int i = 0; i < 9; ++i)
            if (!chk[i]) return 0;
    }
    if (check3x3(head) &&
        check3x3(head->linkhang->linkhang->linkhang) &&
        check3x3(head->linkhang->linkhang->linkhang->linkhang->linkhang->linkhang)
            &&
        check3x3(head->linkcot->linkcot->linkcot) &&
        check3x3(head->linkcot->linkcot->linkcot->linkhang->linkhang->linkhang) &&
        check3x3(head->linkcot->linkcot->linkcot->linkhang->linkhang->linkhang->
            linkhang->linkhang->linkhang) &&
        check3x3(head->linkcot->linkcot->linkcot->linkcot->linkcot->linkcot) &&
        check3x3(head->linkcot->linkcot->linkcot->linkcot->linkcot->linkcot->
            linkhang->linkhang->linkhang) &&
        check3x3(head->linkcot->linkcot->linkcot->linkcot->linkcot->linkcot->
            linkhang->linkhang->linkhang->linkhang->linkhang->linkhang))
        return 1;
    return 0;
}

int main() {
    int **p = new int *[9];
    for (int i = 0; i < 9; ++i) p[i] = new int[9];
    for (int i = 0; i < 9; ++i) {
        for (int j = 0; j < 9; ++j) cin >> p[i][j];
    }
    cout << CheckSudokuCondition(CreateSudokulist(p));
}
```

16 Master theorem

Assume that $a \geq 1, b \geq 2, c > 0, k \geq 0, f(n) = \Theta(n^k)$ and $T(n) = aT(n/b) + f(n)$

- If $a > b^k : T(n) = \Theta(n^{log_b a})$
- If $a = b^k : T(n) = \Theta(n^k \log n)$
- If $a < b^k : T(n) = \Theta(n^k)$