Programing Basic Knowledge LaTeX

Rayhanul Islam Mukul

15 December 2022

Contents

Chapter 1

Data Structure

Topic

- Arrays
- Linked Lists
- Stacks
- Queues
- Trees
- Heaps
- Hash table
- Graphs

1.1 Array

Code 1.1: Array of C++ programming language

```cpp
int arr[10] = {1, 12, 3, 24, 35, 16};
sort(arr, arr+10, [](int i, int j){
   return i > j;
});
```

1.2 Stack

LIFO = Last in Rirst Out

Code 1.2: Stack of C++ programming language

```
stack <int> mystack;    // push, pop, top, empty : O(1)
mystack.push(10);
mystack.pop();
mystack.empty();
```

1.3 Queue

FIFO = First in First Out

Code 1.3: Queue of C++ programming language

```
queue <int> myqueue;
myqueue.push(10);                  //Enqueue (push): O(1) - inserting
myqueue.pop();                     //Dequeue (pop): O(1) - removing
int front = myqueue.front();       //Front (front): O(1) - accessing
int back = myqueue.back();
int size = myqueue.size();         //O(1)
bool empty = myqueue.empty();      //Empty (empty): O(1) - checking
```

1.4 Priority Queue

Code 1.4: Priority Queue of C++ programming language

```
priority_queue<int> myPriorityQueue;
myPriorityQueue.push(30);                           //O(log n)
myPriorityQueue.pop();                              //O(log n)
int highestPriority = myPriorityQueue.top();        //O(1)
```

```
priority_queue <int, vector<int>, greater<int>> myPriorityQueue;  //O(log n)
```

```
if (myPriorityQueue.empty()) {
    // the priority queue is empty
}
```

1.5 DeQueue

Code 1.5: Priority Queue of C++ programming language

```
deque <int> mydeque;
mydeque.push_back(10);           //Insertion (push_back, push_front): O(1)
mydeque.push_front(20);
mydeque.pop_front();             //Deletion (pop_back, pop_front): O(1)
mydeque.pop_back();
```

```
mydeque.erase(mydeque.begin());  //Insertion/Deletion (insert, erase): O(n)
mydeque.insert(mydeque.begin(), 10);
int frontElement = mydeque.front();
int backElement = mydeque.back();
bool empty = mydeque.empty();
mydeque.clear();
```

Chapter 2

Standard Template Library (STL)

Topic

- Containers: vector, deque, list, set, multiset, map, multimap
- Iterators:
- Algorithms:
- Function:
- Function adapters:

2.1 Map

Code 2.1: Map

```
map <string, int> myMap;
myMap["meow"] = 1;          //Insertion, Deletion, Search: O(log n)
myMap.insert({"meeow", 2});
myMap.erase("meow");
for(auto it = myMap.begin(); it != myMap.end(); it++){
    cout << it->first << " " << it->second << endl;
}
```

```
if(myMap.count("meow")){
    // key exists
}
```

```
map <int, vector <int>> myMap;
map <int, vector <int>> :: iterator it;
for(it = myMap.begin(); it != myMap.end(); it++){
    sort(it->second.begin(), it->second.end());
}
for(auto it1 : myMap){
    x = it1.first;
    for(int i = 0; i < myMap[x].size(); i++){
        cout << myMap[x][i] << gap;
    }
}
```

2.2 Vector Pair

Code 2.2: Pair

```
    pair <int, int> myPair;
    myPair = make_pair(1, 2);
```

```
    vector <pair <int, int>> vecPair;
    vecPair.push_back(make_pair(1, 2));        //Access, insert O(1)
    sort(vecPair.begin(), vecPair.end());      //sort O(n log n)
    for(int i = 0; i < vecPair.size(); i++){   //Find element: O(log n) Bin_Sear
        cout << vecPair[i].first << gap << vecPair[i].second << endl;
    }
```

```
    vector <pair <int, int>> vecPair[1000];
    vecPair[0].pb({5, 2});                    // Creating a tuple: O(1)
    vecPair[0].pb({2, 3});
    sort(vecPair[0].begin(), vecPair[0].end());
    for(auto [u, w] : vecPair[0]){            //Comparing two tuples: O(n)
        cout << u << gap << w << endl;
    }
```

2.3 Pointer

Code 2.3: Pointer

```
#include <bits/stdc++.h>
using namespace std;
void call(int *x){
   *x = 20;
   cout << "*x : " << *x << endl;
}
int main(){
   int n = 10;
   call(&n);
   cout << "n : " << n << endl;

   int a = 10, b = 20;
   vector <int*> v;
   v.push_back(&a);
   v.push_back(&b);
   *v[0] += 1;
   cout << "a : " << a << endl;

   return 0;
}
```

2.4 Set

Code 2.4: Set

```
set <int> mySet;    // Insert, Delete, Find in: O(logN)
mySet.insert(1);
mySet.erase(1);
for(auto it: mySet) cout << it << gap;  // Traverse in: O(NlogN)
int numOccurences = mySet.count(1);     // Count in O(logN)
int minValue = *mySet.begin();          // Minimum in O(1)
int maxValue = *mySet.rbegin();         // Maximum in O(1)
if(mySet.find(1) != mySet.end()) cout << "Found" << endl; // Find in O(logN)
```

2.5 Multiset

```
multiset <int> myMultiset;    // Insert, Delete, Find in: O(logN)
myMultiset.insert(1);
myMultiset.erase(1);
for(auto it: myMultiset) cout << it << gap;  // Traverse in: O(NlogN)
int numOccurences = myMultiset.count(1);     // Count in O(logN)
int minValue = *myMultiset.begin();          // Minimum in O(1)
int maxValue = *myMultiset.rbegin();         // Maximum in O(1)
if(myMultiset.find(1) != myMultiset.end()){
    cout << "Found" << endl;                 // Find in O(logN)
}
```

2.6 String

Code 2.5: Transform

```
    transform(s.begin(), s.end(), s.begin(), ::tolower); // O(n), n = length
    transform(s.begin(), s.end(), s.begin()+1, ::toupper);
```

Code 2.6: String concatenation

```
    string str1 = "Hello", str2 = "World";
    string str3 = str1 + str2;        //Time Complexity: O(m+n)
```

Code 2.7: Substring extraction

```
    string str1 = "Hello World";    //Complexity is: O(K), k is length
    string s1 = str1.substr(0, 7);  //0 = index_start , 7 = length
```

Code 2.8: String search

```
    string str1 = "Hello World";
    string find = "Wor";      //O(n*m), where n = str1.size(), m = find.size()
    int pos = str1.find(find);
    if(pos != string::npos){
        cout << "Found at " << pos << endl;
    }
```

2.6.1 String manipulation (e.g. reverse, replace, split):

Code 2.9: String manipulation

```
#include <iostream>
#include <string>

int main() {
    std::string str = "Hello World";
    std::reverse(str.begin(), str.end());
    std::cout << str << std::endl;
    return 0;
}
```

2.6.2 String compression (e.g. Run-length encoding, Huffman coding):

Code 2.10: String compression

```
#include <iostream>
#include <string>

std::string runLengthEncoding(std::string str) {
    std::string result;
    int count = 1;
    for (int i = 1; i < str.length(); i++) {
        if (str[i] == str[i - 1]) {
            count++;
        } else {
            result += str[i - 1] + std::to_string(count);
            count = 1;
        }
    }
    result += str[str.length() - 1] + std::to_string(count);
    return result;
}

int main() {
    std::string str = "aaaabbcccdde";
    std::cout << runLengthEncoding(str) << std::endl;
    return 0;
}
```

2.6.3 String hashing:

Code 2.11: String hashing

```
#include <iostream>
#include <string>

int hashFunction(std::string str) {
    int hash = 0;
    for (char c : str) {
        hash = hash + c;
    }
    return hash % 10;
}

int main() {
    std::string str = "Hello World";
    std::cout << hashFunction(str) << std::endl;
    return 0;
}
```

2.6.4 String sorting (e.g. radix sort):

Code 2.12: String sorting

```
#include <iostream>
#include <vector>
#include <string>

int getMaxLength(const std::vector<std::string> &arr) {
    int maxLength = 0;
    for (const std::string &s : arr) {
        maxLength = std::max(maxLength, (int)s.length());
    }
    return maxLength;
}

void countingSort(std::vector<std::string> &arr, int position) {
    std::vector<std::vector<std::string>> buckets(256);
    for (const std::string &s : arr) {
        int index = (position < s.length()) ? (int)s[position] : 0;
        buckets[index].push_back(s);
    }
    int k = 0;
    for (const std::vector<std::string> &bucket : buckets) {
        for (const std::string &s : bucket) {
            arr[k++] = s;
        }
    }
}

void radixSort(std::vector<std::string> &arr) {
    int maxLength = getMaxLength(arr);
    for (int position = maxLength - 1; position >= 0; position--) {
        countingSort(arr, position);
    }
}

int main() {
    std::vector<std::string> arr = {"caterpillar", "bee", "ant", "dog", "cat"};
    radixSort(arr);
    for (std::string s : arr) {
        std::cout << s << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

2.6.5 String comparison:

Code 2.13: String comparisonn

```
#include <iostream>
#include <string>

int main() {
    std::string s1 = "hello";
    std::string s2 = "world";
    if (s1 == s2) {
        std::cout << "The strings are equal." << std::endl;
    } else if (s1 < s2) {
        std::cout << "s1 is less than s2." << std::endl;
    } else {
        std::cout << "s1 is greater than s2." << std::endl;
    }
    return 0;
}
```

2.6.6 Palindrome detection:

Code 2.14: Palindrome detection

```
#include <iostream>
#include <string>

bool isPalindrome(const std::string &s) {
    int start = 0;
    int end = (int)s.length() - 1;
    while (start < end) {
        if (s[start] != s[end]) {
            return false;
        }
        start++;
        end--;
    }
    return true;
}
```

```
int main() {
    std::string s = "racecar";
    if (isPalindrome(s)) {
        std::cout << "The string is a palindrome." << std::endl;
    } else {
        std::cout << "The string is not a palindrome." << std::endl;
    }
    return 0;
}
```

2.7 Big Mod

Code 2.15: Big Mode

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long int
int power(ll a, ll b, ll mod){
    if(b == 0) return 1%mod;
    ll tmp = power(a, b/2, mod);
    ll result =( tmp*tmp)%mod;
    if(b%2 == 1) result = (result*a) % mod;
    return result;
}
int main(){
    cout << power(5, 5, 10) << endl;
    return 0;
}
```

2.8 Binary Exponentiation Power

Code 2.16: Binary Exponentiation

```
#include <bits/stdc++.h>
using namespace std;
const int MOD = 1e9 + 7;
int power(int a, int b){
    int res = 1;
    while(b > 0){
        if(b%2 == 1){
            res = res * a;
            res %= MOD;
            b--;
        }
        a = a * a;
        a %= MOD;
        b /= 2;
    }
    return res;
}
int main(){
    int a = 2, b = 3;
    cout << power(a, b) << endl;

    return 0;
}
```

2.9 Sieve Prime Factorization

Code 2.17: Sieve-Prime Factorization

```
const int n = 100000
bool prime[1000000];
vector <int> p;
void sieve(int n){
   for(ll i = 3; i*i <= n; i+=2){
       if(prime[i] == false){
           for(ll j = i*i; j <= n; j+=i)prime[j] = true;
       }
   }
   p.push_back(2);
   for(int i = 3; i <= n; i+=2) if(prime[i] == false) p.push_back(i);
   for(int i = 0; i < p.size(); i++)  cout << p[i] << " ";
}
```

2.10 Binary Search

Code 2.18: Binary Search

```
#include <bits/stdc++.h>
using namespace std;
const int MOD = 1e9 + 7;
int binary_search(int arr[], int size, int item){
    int left = 0, right = size - 1, mid;
    while(left <= right){
        mid = left + (right - left)/2;
        if(arr[mid] == item) return mid;
        if(arr[mid] < item) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
int main(){
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int ans = binary_search(arr, 10, 8);
    if(ans){
        cout << "Index of : " << ans << endl;
    }
    else cout << "Not found it" << endl;

    return 0;
}
```

2.11 Dynamic Programing

Code 2.19: Fibonacci Series

```cpp
#include <bits/stdc++.h>
using namespace std;
int dp[105];
int f(int n){
    if(n == 0) return 0;
    if(n == 1) return 1;
    if(dp[n] >= 0) return dp[n];
    dp[n] = f(n-1) + f(n-2);
    return dp[n];
}
int main(){
    for(int i = 0; i < 105; i++) dp[i] = -1;
    cout << f(7) << endl;
}
```

2.12 Euclidean Algorithm

Code 2.20: Euclidean Algorithm GCD LCM

```
#include <bits/stdc++.h>
using namespace std;
int gcd(int a, int b) {
    if (a == 0) return b;
    return gcd(b % a, a);
}
int main(){
    int a = 4, b = 10;
    int lcm = a*b/(__gcd(a, b));
    cout << lcm << " " << __gcd(a, b) << endl;

    return 0;
}
```

2.13 Lamda Function

Code 2.21: Lamda Function

```
#include <bits/stdc++.h>
using namespace std;
auto func = [] (int i, int j){
        return i+j;
};
int main(){
    cout << func(10, 20) << endl;

    return 0;
}
```

Code 2.22: Lamda Function

```

#include <bits/stdc++.h>
using namespace std;
function <int(int)> fib = [](int i){
       if(i <= 1) return 1;
       return fib(i-1) + fib(i-2);
};
int main(){
    cout << fib(5) << endl;

    return 0;
}
```

2.14 Bitwise Operation

Code 2.23: Bitwise Operation

```
Bitwise XOR:
    5 = 0 1 0 1
    6 = 0 1 1 0
   5ˆ6 = 0 0 1 1
   Left Shift: left_shift * 2
     x << = 0 1 1 0 1 0 1 1
   x << 2 = 1 1 0 1 0 1 0 0
   Right Shift: Right_shift / 2
     x >> = 0 1 1 0 1 0 1 1
   x >> 2 = 0 0 0 1 1 0 1 0
```

Chapter 3

Graph Theoury

3.1 Knuth–Morris–Pratt (KMP)

KMP time complexity: O(n+m). where, n = length of text, m = length of pattern

Code 3.1: KMP Algorithm

```
vector <int> createTemArray(string pattern){
    vector <int> lps(pattern.size());
    int ind = 0;
    for(int i = 1; i < pattern.size();){
        if(pattern[i] == pattern[ind]){
            lps[i] = ind + 1;
            ind++; i++;
        }
        else{
            if(ind != 0) ind = lps[ind - 1];
            else{
                lps[i] = ind;
                i++;
            }

        }
    }
    return lps;
}
void kmp(string pattern, string text){
    vector <int> lps = createTemArray(pattern);
    int i = 0, j = 0;   // i for text, j for pattern
    while(i < text.size()){
        if(text[i] == pattern[j]){
            i++; j++;
        }
        else{
            if(j != 0) j = lps[j-1];
            else i++;
        }
        if(j == pattern.size()){
            cout << "Pattern found at index " << i - j << endl;
            j = lps[j-1];   // for overlapping pattern
        }
    }
}
void solve(int tt){
    string pattern = "abc";
    string text = "ababcababcabc";
    kmp(pattern, text);
}
```

3.2 Rabin Karp Algorithm

Rabin Karp time complexity: O(n+m). where, n = length of text, m = length of pattern

Code 3.2: Rabin Karp Algorithm

```
void Rabinkarp(string pattern, string text){
    int m = pattern.length();
    int n = text.length();
    int q = INT_MAX;
    int x = 11;     // typically prime - multiplier
    int x_m = 1;    // x^(m-1) used for next hash computation
    int hash_p = 0, hash_t = 0;
    for(int i = 0; i < m - 1; i++){
        x_m = (x_m * x) % q;
    }
    // compute hash for pattern and first window of text
    for(int i = 0; i < m; i++){
        hash_p = (x * hash_p + pattern[i]) % q;
        hash_t = (x * hash_t + text[i]) % q;
    }
    for(int i = 0; i <= n - m; i++){
        if(hash_p == hash_t){   // check for character by character match
            /*bool flag = true;
            for(int j = 0; j < m; j++){
                if(text[i + j] != pattern[j]){
                    flag = false;
                    break;
                }
            }
            if(flag){
                cout << "Pattern found at index " << i << endl;
            }*/
            cnt++;
        }
        //if it was not last window, then continue hash computation
        if(i < n - m){
            hash_t = (x * (hash_t - text[i] * x_m) + text[i + m]) % q;
            if(hash_t < 0){
                hash_t += q;
            }
        }
    }
}
void solve(int tt){
    string pattern = "abc";
    string text = "ababcababcabc";
    Rabinkarp(pattern, text);
}
```

```
int Hash(const string &s, int m, int B, int M){
    int h = 0, power = 1;
    for(int i = m-1; i >= 0; i--){
        h = (h + (s[i] * power) % M) % M;
        h = h % M;
        power = (power * B) % M;
    }
    return h;
}
int Rabinkarp(const string &text, const string &pattern){
    int n = text.size();
    int m = pattern.size();
    if(n < m) return -1;
    if(m == 0 || n == 0) return -1;
    int B = 347, M = MOD;
    int power = 1;
    for(int i = 1; i < m; i++) power = (power * B) % M; // Calculate B^m-1
    // Find hash value of fisrt m characters of text
    // Find hash value of pattern[0...m-1]
    int hash_text = Hash(text, m, B, M);
    int hash_pattern = Hash(pattern, m, B, M);

    if(hash_text == hash_pattern) return 0;
    for(int i = m; i < n; i++){
        // Update Rolling Hash
        hash_text = (hash_text - (text[i-m] * power) % M) % M;
        hash_text = (hash_text + M) % M;
        hash_text = (hash_text * B) % M;
        hash_text = (hash_text + text[i]) % M;

        if(hash_text == hash_pattern) return i-m+1;
    }
    return -1;
}
void solve(int tt){
    string pattern = "abc";
    string text = "abaabdabcjkhabcabca";
    cout << Rabinkarp(text, pattern) << endl;  // Return First index of pattern
       in text
}
```

3.3 Breadth First Search (BFS)

BFS time complexity: O(V + E) where V is no. of vertices and E is no. of edges

Code 3.3: Breadth First Search of C++ programming language

```cpp
const int MAX = 2e5+5;
vector <int> adj[MAX];
vector <bool> vis(MAX, false);
vector <int> dis(MAX, 0);
void bfs(int s){
    queue <int> q;
    q.push(s);
    vis[s] = true;
    while(!q.empty()){
        int node = q.front();
        q.pop();
        for(auto child : adj[node]){
            if(!vis[child]){
                q.push(child);
                vis[child] = true;
                dis[child] = dis[node] + 1;
            }
        }
    }
}
void solve(int tt){
    int n, m;       // n = no. of nodes, m = no. of edges
    cin >> n >> m;
    for(int i = 1; i <= m; i++){
        int x, y;
        cin >> x >> y;
        adj[x].pb(y);
        adj[y].pb(x);
    }
    bfs(1);
    for(int i = 1; i <= n; i++){
        cout << dis[i] << gap;
    }
}
```

3.4 Depth First Search (DFS)

Code 3.4: Depth First Search of C++ programming language

```
bool vis[MAX];
vector <int> adj[MAX];
vector <int> ans;
void dfs(int node){
    if(vis[node]) return;
    ans.push_back(node);
    vis[node] = true;
    for(auto child : adj[node]){
        dfs(child);
    }
    return;
}
void solve(int tt){
    int n, m;        // n = no. of nodes, m = no. of edges
    cin >> n >> m;
    memset(vis, false, sizeof(vis));
    for(int i = 1; i <= m; i++){
        int x, y;
        cin >> x >> y;
        adj[x].pb(y);
        adj[y].pb(x);
    }
    int start; cin >> start;
    dfs(start);
    for(auto x : ans){
        cout << x << gap;
    }
}
```

3.5 Dijkstra's Algorithm

Dijkstra's Algorithm time complexity: O(E Log V) where, E is the number of edges and V is the number of vertices.
Space Complexity: O(V)

Code 3.5: Dijkstra's algorithm of C++ programming language

```
const int N = 1e5+5;
void dijkstra(vector <vector <pair <int, int>>> &adj, int source, int n){
    vector <int> dist(N, INT_MAX);
    vector <bool> vis(N, false);
    priority_queue <pair <int, int>, vector <pair <int, int>>, greater <pair <
        int, int>>> pq;
    pq.push({0, source});
    dist[source] = 0;
    while(!pq.empty()){
        int nd = pq.top().second;
        pq.pop();
        if(vis[nd]) continue;
        vis[nd] = true;
        for(auto [v, w] : adj[nd]){
            if(dist[v] > dist[nd] + w){
                dist[v] = dist[nd] + w;
                pq.push({dist[v], v});
            }
        }
    }
    for(int i = 1; i <= n; i++){
        cout << dist[i] << gap;
    }
}
void solve(int tt){
    int n;
    cin >> n;
    vector <vector <pair <int, int>>> adj(N);
    for(int i = 1; i < n; i++){
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        dbg(adj);
    }
    int source;
    cin >> source;
    dijkstra(adj, source, n);
}
```

3.6 Graham Scan Algorithm

Graham Scan Algorithm time complexity: O(n log n) where, n is the number of input points.in the best and average case,and O(n^2)*intheworstcase*.

Code 3.6: Graham Scan algorithm of C++ programming language

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Point {
    int x, y;
};
long long cross_product(Point a, Point b, Point c) {
    long long x1 = b.x - a.x;
    long long y1 = b.y - a.y;
    long long x2 = c.x - a.x;
    long long y2 = c.y - a.y;
    return x1 * y2 - x2 * y1;
}
bool compare_points(Point a, Point b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
vector<Point> convex_hull(vector<Point>& points) {
    int n = points.size();
    if (n <= 1) return points;

    sort(points.begin(), points.end(), compare_points);

    vector<Point> hull;

    for (int i = 0; i < n; ++i) {
        while (hull.size() >= 2 && cross_product(hull[hull.size()-2], hull[hull.
            size()-1], points[i]) < 0) {
            hull.pop_back();
        }
        hull.push_back(points[i]);
    }

    int lower_hull_size = hull.size();

    for (int i = n-2; i >= 0; --i) {
        while (hull.size() > lower_hull_size && cross_product(hull[hull.size()
            -2], hull[hull.size()-1], points[i]) < 0) {
            hull.pop_back();
        }
        hull.push_back(points[i]);
    }

    hull.pop_back();
    return hull;
}
int main() {
    int n;
    cin >> n;
    vector<Point> points(n);
    for (int i = 0; i < n; ++i) {
        cin >> points[i].x >> points[i].y;
    }
    vector<Point> hull = convex_hull(points);
    cout << hull.size() << endl;
    for (int i = 0; i < hull.size(); ++i) {
        cout << hull[i].x << " " << hull[i].y << endl;
    }
    return 0;
}
```

Code 3.7: Graham Scan algorithm of C++ programming language

```
class Solution {
public:
    vector<vector<int>> FindConvexHull(vector<vector<int>> &points) {
        int n = points.size();
        if(n <= 2) return vector <vector<int>> (1, vector <int> {-1});
        int leftmost = 0, mostMin = 1e9;
        for(int i = 0; i < n; ++i){
            if(points[i][0] < mostMin){
                mostMin = points[i][0];
                leftmost = i;
            }
            if(points[i][0] == mostMin && points[i][1] < points[leftmost][1]){
                leftmost = i;
            }
        }
        vector<vector<int>> hull;
        int p = leftmost;
        do{
            hull.push_back(points[p]);
            int q = (p + 1) % n;
            for(int i = 0; i < n; ++i){
                if(crossProduct(points[p], points[i], points[q]) == 2){
                    q = i;
                }
                else if(crossProduct(points[p], points[i], points[q]) == 0){
                    int64_t dist = distance(points[p], points[q], points[i]);
                    if(dist < 0) q = i;
                }
            }
            p = q;
        }while(p != leftmost);

        sort(hull.begin(), hull.end());
        return hull;
    }

private:
    int64_t crossProduct(vector<int>& a, vector<int>& b, vector<int>& c) {
        int64_t x1 = b[0] - a[0];
        int64_t y1 = b[1] - a[1];
        int64_t x2 = c[0] - a[0];
        int64_t y2 = c[1] - a[1];
        int64_t val = x1 * y2 - x2 * y1;
        if(val == 0) return 0;
        return val > 0 ? 1 : 2;
    }
    long long distance(vector<int>& a, vector<int>& b, vector<int>& c) {
        int64_t x1 = a[0] - b[0];
        int64_t y1 = a[1] - b[1];
        int64_t dist1 = (y1*y1) + (x1*x1);
        int64_t x2 = a[0] - c[0];
        int64_t y2 = a[1] - c[1];
        int64_t dist2 = (y2*y2) + (x2*x2);
        return dist1 - dist2;
    }
};
```

3.7 Meet in the middle

Time Complexity: $O(2^{\frac{n}{2}} \cdot \log(2^{\frac{n}{2}}))$
Auxiliary Space: $O(2^{\frac{n}{2}})$

Code 3.8: Meet in the middle

```
void solve(int tt){
    int n, A, B;
    cin >> n >> A >> B;
    vector<int> a(n);
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }
    vector <int> sum1, sum2;
    int half = n/2;
    for(int i = 0; i < (1<<half); i++){
        int sum = 0;
        for(int j = 0; j < half; j++){
            if(i & (1<<j)){
                sum += a[j];
            }
        }
        sum1.pb(sum);
    }
    for(int i = 0; i < (1<<(n-half)); i++){
        int sum = 0;
        for(int j = 0; j < (n-half); j++){
            if(i & (1<<j)){
                sum += a[j+half];
            }
        }
        sum2.pb(sum);
    }
    sor(sum1);
    int ans = 0;
    for(auto it:sum2){
        int x = A - it;
        int y = B - it;
        int l = lower_bound(sum1.begin(), sum1.end(), x) - sum1.begin();
        int r = upper_bound(sum1.begin(), sum1.end(), y) - sum1.begin();
        ans += (r-l);
    }
    cout << ans << endl;
}
```

Chapter 4

Common Mistakes in Competitive Programming

By YouKn0wWho,

4.0.1 Mistake 1

Check out the following code:

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

int main() {
  int a = 1000'000'000,b = 1000'000'000;
  long long product = a * b;
  cout << product << '\n';
  return 0;
}
```

The output should be 10^{18}. But if you run the code, you will get a different output. Why? Overflow!

Even if you declared the product variable as long long, what's happening, in reality, is, first the compiler is multiplying a with b and then it is assigning the output to the product variable and as a and b both are int, that's why the output is also int, but 10^{18} is too much to contain for an int variable.

So how to solve this?

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

int main() {
  int a = 1000'000'000,b = 1000'000'000;
  long long product = a * b;
  cout << product << '\n';
  return 0;
}
```

Code 4.1: Basic Knowledge of C++ programming language

```
//=================================================
//Basic Knowledge
//=================================================

cout << fixed << setprecision(10) << sum << endl;srand(clock());
value = (pow(3, 4) + 0.5)
int min  = INT_MIN, max = INT_MAX;
char ch[26];
```

```
while (scanf("%s",ch) != EOF)
 < input.txt > output.txt
### C & C++:
+) Loop:  i) Initialization (i=0);   ii) Terminate if(i<10);  iii) Increment/
   Decrement (i=i+1;)
+) Library Function:  pow(), strlen(), strcpy(), strrev(), strupr(), strlwr(),
   strcmp(), strcat()
+) int num;  scanf("%4d", & num); printf("%d\n",num);
+) double num;  scanf("%lf", & num);  printf("sum is= %0.3lf\n",sum);
+) float num; scanf("%f", & num);   printf("%f\n",num);
+) char n;    scanf("%c", & n); printf("%c\n",n);
+) char str[]; int l;     l = strlen(str);   scanf("%s", & str);  printf("%s\n",
   str);
+) char s[100];  scanf("%s",s); printf("%s\n",s);
+) long long int n; scanf("%lld", &n);  printf("%lld", n);
+) scanf("%[^\n]",str);
+) fflush(stdin);  gets(person[i].name);
//====================================================
```